

VAX DOCUMENT

User Manual, Volume 2

Order Number: AA-JT86A-TE

This manual describes each of the VAX DOCUMENT doctypes and the tags that are specific to each of these doctypes. These tags are described by the doctype in which they can be used.

Revision/Update Information: This is a new manual.

Operating System and Version: VMS Version 4.4 or higher.
MicroVMS Version 4.4 or higher.

Software Version: VAX DOCUMENT Version 1.0.

digital equipment corporation maynard, massachusetts

July 1987

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 ©1987 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	

The logo for Digital Equipment Corporation, featuring the word "digital" in a stylized, lowercase font with a trademark symbol, set against a dark background.

ZK4459

This document was prepared using VAX DOCUMENT, Version 1.0

VAX DOCUMENT User Manual, Volume 2

Order Number: AA-JT86A-TE

This manual describes each of the VAX DOCUMENT doctypes and the tags that are specific to each of these doctypes. These tags are described by the doctype in which they can be used.

Revision/Update Information: This is a new manual.

Operating System and Version: VMS Version 4.4 or higher.
MicroVMS Version 4.4 or higher.

Software Version: VAX DOCUMENT Version 1.0.

July 1987

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 ©1987 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	digital ™

ZK4459

This document was prepared using VAX DOCUMENT, Version 1.0

Contents

PREFACE

xiii

CHAPTER 1 OVERVIEW OF THE DOCTYPE-SPECIFIC TAGS 1-1

CHAPTER 2 USING THE ARTICLE DOCTYPE 2-1

2.1	ARTICLE DOCTYPE-SPECIFIC TAGS	2-2
2.1.1	Titles and Subtitles _____	2-3
2.1.2	Author Information _____	2-3
2.1.3	Abstracts, Source Notes, and Acknowledgments _____	2-4
2.1.4	Headings _____	2-5
2.1.5	Running Titles _____	2-5
2.1.6	Quotations _____	2-5
2.1.7	Back Notes and Reference Notes _____	2-6
	2.1.7.1 Back Notes • 2-6	
	2.1.7.2 Reference Notes • 2-7	
2.1.8	Bibliographies _____	2-7

2.2	IMPROVING THE FORMAT OF A TWO-COLUMN DOCTYPE	2-8
2.2.1	Line Breaks in Columns _____	2-8
2.2.2	Wide Tables and Examples _____	2-9
2.2.3	Final Adjustment of Column and Page Breaks _____	2-9

2.3	A SAMPLE USE OF THE ARTICLE DOCTYPE TAGS	2-11
------------	---	-------------

CHAPTER 3 USING THE LETTER DOCTYPE 3-1

3.1	SAMPLE USES OF THE LETTER DOCTYPE TAGS	3-2
3.1.1	A Sample Memo _____	3-3
3.1.2	A Sample Letter _____	3-5

Contents

CHAPTER 4	USING THE MANUAL DOCTYPE	4-1
4.1	A SAMPLE OF THE MANUAL DOCTYPE	4-1
CHAPTER 5	USING THE MILSPEC DOCTYPE	5-1
5.1	WRITING A DOCUMENT ACCORDING TO MIL-STD-490A	5-2
5.2	WRITING DATA ITEM DESCRIPTION DOCUMENTS	5-3
5.3	A SAMPLE USE OF THE MILSPEC DOCTYPE TAGS	5-6
CHAPTER 6	USING THE OVERHEADS DOCTYPE	6-3
6.1	SAMPLE USE OF THE OVERHEADS DOCTYPE TAGS	6-3
CHAPTER 7	USING THE REPORT DOCTYPE	7-1
7.1	A SAMPLE USE OF THE REPORT DOCTYPE TAGS	7-2
CHAPTER 8	USING THE SOFTWARE DOCTYPE	8-1
8.1	SOFTWARE DOCTYPE-SPECIFIC TAGS	8-2
8.1.1	Documenting Terminal Keys and Keypads	8-2
8.1.1.1	Describing Individual Keys • 8-2	
8.1.1.2	Describing Keypads and Keypad Keys • 8-5	
8.1.2	Documenting Code Fragments	8-8
8.1.3	Documenting Software Messages	8-10
8.1.4	Documenting Arguments, Parameters and Qualifiers	8-14
8.1.5	Creating A Series of Interactive or Code Examples	8-18
8.2	USING THE REFERENCE TEMPLATES	8-19
8.2.1	Documenting Reference-Elements within Templates	8-23

8.3	CREATING YOUR OWN REFERENCE TEMPLATE TAGS	8-25
8.3.1	Creating Your Own Template Lists _____	8-25
8.3.2	Creating Your Own Template Sections _____	8-27
8.3.3	Creating Your Own Template Tables _____	8-29
<hr/>		
8.4	MODIFYING THE REFERENCE TEMPLATES	8-31
8.4.1	Modifying Default Headings Within a Template _____	8-31
8.4.2	Using the Template-enabling Tags _____	8-33
8.4.3	Using the <SET_TEMPLATE_<templatename> Tags _____	8-34
<hr/>		
8.5	USING THE COMMAND TEMPLATE	8-35
8.5.1	Sample SDML File of the Command template _____	8-37
8.5.2	Sample Output File of the Command Template _____	8-39
<hr/>		
8.6	USING THE ROUTINE TEMPLATE	8-42
8.6.1	Sample SDML File of the Routine Template _____	8-43
8.6.2	Sample Output File of the Routine Template _____	8-44
<hr/>		
8.7	USING THE STATEMENT TEMPLATE	8-48
8.7.1	Sample SDML File of the Statement Template _____	8-49
8.7.2	Sample Output File of the Statement Template _____	8-50
<hr/>		
8.8	USING THE TAG TEMPLATE	8-53
8.8.1	Sample SDML File of the Tag Template _____	8-54
8.8.2	Sample Output File of the Tag Template _____	8-55

Contents

CHAPTER 9	ARTICLE DOCTYPE TAG REFERENCE	9-1
	<ABSTRACT>	9-2
	<ACKNOWLEDGMENTS>	9-3
	<AUTHOR>	9-4
	<AUTHOR_ADDR>	9-6
	<AUTHOR_AFF>	9-8
	<AUTHOR_LIST>	9-9
	<BACK_NOTE>	9-10
	<BACK_NOTES>	9-12
	<BIBLIOGRAPHY>	9-13
	<BIB_ENTRY>	9-14
	<COLUMN>	9-15
	<DOCUMENT_ATTRIBUTES>	9-17
	<QUOTATION>	9-20
	<REF_NOTE>	9-21
	<REF_NOTES>	9-23
	<RUNNING_FEET>	9-24
	<RUNNING_TITLE>	9-25
	<SOURCE_NOTE>	9-27
	<SUBTITLE>	9-28
	<TITLE>	9-29
	<TITLE_SECTION>	9-30
	<VITA>	9-31

CHAPTER 10	LETTER DOCTYPE TAG REFERENCE	10-1
	<CC>	10-2
	<CCLIST>	10-4
	<CHEAD>	10-5
	<CLOSING>	10-7
	<DISTLIST>	10-8
	<FROM_ADDRESS>	10-9
	<HEAD>	10-11
	<MEMO_DATE>	10-13
	<MEMO_FROM>	10-15
	<MEMO_HEADER>	10-17
	<MEMO_LINE>	10-18
	<MEMO_TO>	10-20
	<SALUTATION>	10-22
	<SUBJECT>	10-23
	<TO_ADDRESS>	10-24

CHAPTER 11	MILSPEC DOCTYPE TAG REFERENCE	11-1
	<SET_APPENDIX_NUMBER>	11-2
	<SIGNATURE_LINE>	11-4
	<SIGNATURE_LIST>	11-6
	<SPECIFICATION_INFO>	11-8
	<SPEC_TITLE>	11-10
	<SUBTITLE>	11-11

CHAPTER 12	OVERHEADS DOCTYPE TAG REFERENCE	12-1
	<AUTHOR_INFO>	12-2
	<AUTO_NUMBER>	12-3
	<INTRO_SUBTITLE>	12-4
	<INTRO_TITLE>	12-5
	<RUNNING_FEET>	12-6
	<RUNNING_TITLE>	12-8
	<SLIDE>	12-10
	<SUBTITLE>	12-12
	<TEXT_SIZE>	12-13
	<TITLE>	12-15
	<TOPIC>	12-16

CHAPTER 13	REPORT DOCTYPE TAG REFERENCE	13-1
	<AUTHOR>	13-2
	<BYLINE>	13-4
	<CHEAD>	13-6
	<COLUMN>	13-8
	<DOCUMENT_ATTRIBUTES>	13-10
	<HEAD>	13-13
	<LEVEL>	13-15
	<OUTLINE>	13-16
	<RUNNING_FEET>	13-18
	<RUNNING_TITLE>	13-19
	<SECTION>	13-21
	<SHOW_LEVELS>	13-23
	<SIGNATURES>	13-25

CHAPTER 14 SOFTWARE DOCTYPE TAG REFERENCE **14-1**

14.1	THE SOFTWARE DOCTYPE TAGS	14-1
	<ARGDEFLIST>	14-2
	<ARGDEF>	14-5
	<ARGITEM>	14-6
	<ARGUMENT>	14-8
	<AUTHOR>	14-9
	<BYLINE>	14-11
	<CPOS>	14-13
	<DELETE_KEY>	14-14
	<DISPLAY>	14-15
	<DOCUMENT_ATTRIBUTES>	14-17
	<EXAMPLE_SEQUENCE>	14-20
	<EXAMPLES_INTRO>	14-22
	<EXC>	14-24
	<EXI>	14-25
	<EXTTEXT>	14-27
	<GRAPHIC>	14-28
	<KEY>	14-29
	<KEY_NAME>	14-31
	<KEYPAD>	14-32
	<KEYPAD_ENDROW>	14-35
	<KEYPAD_ROW>	14-36
	<KEYPAD_SECTION>	14-38
	<KEY_PLUS>	14-41
	<KEY_SEQUENCE>	14-42
	<KEY_TYPE>	14-44
	<MESSAGE_SECTION>	14-45
	<MESSAGE_TYPE>	14-48
	<MSG>	14-49
	<MSGS>	14-51
	<MSG_TEXT>	14-53
	<PARAMDEF>	14-54
	<PARAMDEFLIST>	14-55
	<PARAMITEM>	14-58
	<QPAIR>	14-60
	<QUAL_LIST>	14-61
	<QUAL_LIST_DEFAULT_HEADS>	14-64
	<QUAL_LIST_HEADS>	14-65
	<QUALDEF>	14-66

	<QUALDEFLIST>	14-67
	<QUALITEM>	14-70
	<RUNNING_FEET>	14-72
	<RUNNING_TITLE>	14-73
	<SIGNATURES>	14-75
	<SYNTAX>	14-76
	<SYNTAX_DEFAULT_HEAD>	14-78
<hr/>		
14.2	TAGS COMMON TO ALL REFERENCE TEMPLATES	14-80
	<DESCRIPTION>	14-81
	<OVERVIEW>	14-84
	<SET_TEMPLATE_HEADING>	14-85
	<SET_TEMPLATE_LIST>	14-87
	<SET_TEMPLATE_PARA>	14-89
	<SET_TEMPLATE_TABLE>	14-91
<hr/>		
14.3	THE COMMAND REFERENCE TEMPLATE TAGS	14-94
	<COMMAND>	14-95
	<COMMAND_SECTION>	14-97
	<FCMD>	14-101
	<FORMAT>	14-104
	<FPARM>	14-106
	<FPARMS>	14-108
	<PROMPT>	14-110
	<PROMPTS>	14-112
	<RESTRICTIONS>	14-113
	<RETURN_VALUE>	14-115
	<RITEM>	14-116
	<SET_TEMPLATE_COMMAND>	14-117
	<SET_TEMPLATE_SUBCOMMAND>	14-119
	<SUBCOMMAND>	14-121
	<SUBCOMMAND_SECTION>	14-123
	<SUBCOMMAND_SECTION_HEAD>	14-124
<hr/>		
14.4	THE ROUTINE REFERENCE TEMPLATE TAGS	14-125
	<ARGTEXT>	14-126
	<FARG>	14-127
	<FARGS>	14-129
	<FFUNC>	14-131
	<FORMAT>	14-133

Contents

<FRTN>	14-135
<RETTEXT>	14-137
<RETURNS>	14-138
<ROUTINE>	14-140
<ROUTINE_SECTION>	14-142
<RSDEFLIST>	14-146
<RSITEM>	14-148
<SET_TEMPLATE_ROUTINE>	14-149

14.5 THE STATEMENT REFERENCE TEMPLATE TAGS 14-151

<CONSTRUCT>	14-152
<CONSTRUCT_LIST>	14-154
<FCMD>	14-157
<FFUNC>	14-160
<FORMAT_SUBHEAD>	14-162
<FPARM>	14-164
<FPARMS>	14-166
<FUNCTION>	14-168
<SET_TEMPLATE_STATEMENT>	14-169
<STATEMENT>	14-171
<STATEMENT_FORMAT>	14-172
<STATEMENT_LINE>	14-174
<STATEMENT_SECTION>	14-177

14.6 THE TAG REFERENCE TEMPLATE TAGS 14-181

<FORMAT>	14-182
<FTAG>	14-183
<RELATED_ITEM>	14-185
<RELATED_TAG>	14-186
<RELATED_TAGS>	14-187
<RESTRICTIONS>	14-189
<RITEM>	14-191
<SDML_TAG>	14-192
<SET_TEMPLATE_TAG>	14-193
<TAG_SECTION>	14-195
<TERMINATING_TAG>	14-199

INDEX

TABLES

2-1	Tags Available in the ARTICLE Doctype _____	2-1
3-1	Tags Available in the LETTER Doctype _____	3-1
5-1	Tags Available in the MILSPEC Doctype _____	5-1
5-2	MILSPEC Doctype Data Item Description Templates _____	5-5
6-1	Tags Available in the OVERHEADS Doctype _____	6-2
7-1	Tags Available in the REPORT Doctype _____	7-1
8-1	Default Headings of Reference Template Tags _____	8-32
8-2	Tags Available in the Command Template in their Standard Order _____	8-36
8-3	Tags Available in the Routine Template in their Standard Order _____	8-42
8-4	Tags Available in the Statement Template in their Standard Order _____	8-48
8-5	Tags Available in the Tag Template in their Standard Order _____	8-53
9-1	Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> Tag _____	9-18
13-1	Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> tag _____	13-11
14-1	Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> tag _____	14-18

Preface

Document Structure

This manual describes the VAX DOCUMENT doctypes and the tags available within each doctype.

- Chapter 1 provides an overview of all the doctypes discussed in this manual.
- Chapters 2 through 8 provide tutorial information on each of the doctypes and their tags.
- Chapters 9 through 14 provide the reference information on the tags available within each of these doctypes.

The *VAX DOCUMENT User Manual, Volume 1* provides a complete description of the global tags as well as tutorial information on how to use them to create a document. Global tags are tags that can be used in any doctype.

The *VAX DOCUMENT Design Samples* manual provides information on each of the doctype designs available in VAX DOCUMENT and provides samples of each design.

Intended Audience

This book is intended for writers, editors, and general users who wish to produce articles, business letters, military specifications, technical manuals, or overhead slides using VAX DOCUMENT. Familiarity with a text editor is presumed, as is a basic knowledge of the VMS operating system.

Conventions

Capitalized words within syntax statements indicate specific commands or keywords to be entered. Lowercase words indicate user-specified parameters or arguments. Optional items within syntax statements are enclosed in brackets.

Within reference chapters, the discussion of each tag follows a fixed order. First, the name of the tag is followed by a brief overview that describes the purpose of the tag. Following the overview is a format section that displays the syntax of the tag: any optional or required arguments and the information required, any related tags, any restrictions on the use of the tag, and any required terminators, if needed.

The category of "related tag" has been defined broadly. A tag is related to the tag under discussion if one of the following criteria are met:

- It is required for use of the tag under discussion.
- It marks a text element of the same kind as the tag under discussion.
- It is commonly used with the tag under discussion.

Preface

Following the format section is an optional description section. The description expands the overview and presents more detailed information on the use of the tag. Not every tag requires a description section.

The discussion of a tag concludes with at least one example, or a reference to an example. The example shows how the tag is used in a source file and what the formatted result is when the file is processed for printing.

Output examples within this manual may vary depending on the doctype you processed the example under, or depending on whether any doctype modifications have been made to your local installation of VAX DOCUMENT. Each output example is introduced by a form of the sentence "This example may produce the following output" to remind you that the output examples may vary.

Associated Documents

The reader is presumed to be familiar with the following documents:

- *Step-by-Step: Writing with VAX DOCUMENT*
- *VAX DOCUMENT User Manual, Volume 1*

Other books in the documentation set for VAX DOCUMENT Version 1.0 are as follows:

- *VAX DOCUMENT Design Samples*
- *VAX DOCUMENT Doctype Designer's Guide*

1

Overview of the Doctype-Specific Tags

This manual describes the VAX DOCUMENT doctypes and their associated tags. These *doctype-specific* tags are restricted to certain doctypes and so are not available in all doctypes as are the global tags described in the *VAX DOCUMENT User Manual, Volume 1*.

For example, the <SALUTATION> tag is a doctype-specific tag available only in the LETTER doctype. This tag is restricted to the LETTER doctype because it is only when you write a letter or memo that you want the output format associated with this tag. VAX DOCUMENT limits tags to specific doctypes to make the system more modular and to reduce the number of tags you must learn to use.

Using this Manual

The doctypes and doctype-specific tags are described in this manual in tutorial chapters (Chapters 2 through 8) and in reference chapters (Chapters 9—14).

Each doctype and its designs is explained in a tutorial chapter, which describes how to use that doctype and what doctype-specific tags and global tags are available within it (some doctypes do not use all of the global tags). These tutorial chapters are ordered alphabetically by doctype name. Each tutorial chapter includes a short sample of at least one SDML file that uses these tags. Each of these sample input files is followed by an output sample of that SDML file. See *VAX DOCUMENT Design Samples* for detailed samples of each doctype design.

The reference chapters describe the syntax and restrictions associated with each doctype-specific tag. These chapters are also arranged alphabetically by doctype. The following table lists the supported doctypes, explains what each doctype is generally used for, and indicates which chapters in this manual describe them.

Doctype Keyword	Used to Create	Tutorial Chapter	Reference Chapter
ARTICLE	Articles	Chapter 2	Chapter 9
LETTER	Letters and memos	Chapter 3	Chapter 10
MANUAL	User manuals	Chapter 4	Because this doctype uses only the global tags, no reference chapter is needed.
MILSPEC	Military specifications	Chapter 5	Chapter 11
OVERHEADS	Overhead slides for transparencies	Chapter 6	Chapter 12
REPORT	General-purpose documents or formal outlines	Chapter 7	Chapter 13
SOFTWARE	User manuals containing software-specific information	Chapter 8	Chapter 14



2

Using the ARTICLE Doctype

The ARTICLE doctype lets you create two-column articles in an $8\frac{1}{2} \times 11$ inch format with numbered or unnumbered headings. The ARTICLE doctype has a single design. You process files under this doctype by using the ARTICLE doctype keyword on the DOCUMENT command line.

This doctype supports all of the VAX DOCUMENT global tags with the following exceptions:

- <CHAPTER>
- <FRONT_MATTER>
- <REVISION> , <MARK> , and <UPDATE_RANGE>
- The MULTIPAGE keyword is not available as an argument to the <EXAMPLE_ATTRIBUTES> , <FIGURE_ATTRIBUTES> , and <TABLE_ATTRIBUTES> tags. This means that formal examples, figures, and tables are restricted to a length of one column or less.

See the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags. See *VAX DOCUMENT Design Samples* for additional information on the design of the ARTICLE doctype.

Table 2-1 summarizes the tags available in the ARTICLE doctype and provides a brief description of each tag. Chapter 9 contains the reference information on these tags.

Table 2-1 Tags Available in the ARTICLE Doctype

Tag Name	Description
<ABSTRACT>	Creates an article abstract.
<ACKNOWLEDGMENTS>	Creates an acknowledgments section in an article.
<AUTHOR>	Specifies an author of an article.
<AUTHOR_ADDR>	Specifies the address of the author.
<AUTHOR_AFF>	Specifies information about the organizational affiliation of the author.
<AUTHOR_LIST>	Creates a list of authors for an article with multiple authors.
<BACK_NOTE>	Creates a back note entry, and creates a superscript reference number in the article text.
<BACK_NOTES>	Causes any accumulated back notes to be output.
<BIBLIOGRAPHY>	Begins a bibliography.
<BIB_ENTRY>	Specifies a single entry in a bibliography.
<COLUMN>	Specifies that a new column of output should begin.
<DOCUMENT_ATTRIBUTES>	Enables doctype-specific tags that override the default design format of the ARTICLE doctype.

Using the ARTICLE Doctype

Table 2–1 (Cont.) Tags Available in the ARTICLE Doctype

Tag Name	Description
<QUOTATION>	Begins a quotation in which the spacing is retained and the text is not filled or justified.
<REF_NOTE>	Specifies the text of a reference note, and creates a bracketed reference number in the article text.
<REF_NOTES>	Causes all accumulated reference notes to be output.
<RUNNING_FEET>	Creates a heading at the bottom of each page.
<RUNNING_TITLE>	Creates a one- or two-line heading at the top of each page.
<SOURCE_NOTE>	Provides information pertaining to the original source of information for an article.
<SUBTITLE>	Specifies a subtitle for an article.
<TITLE>	Specifies the main title line for an article.
<TITLE_SECTION>	Begins the title section of an article that spans both columns of the article.
<VITA>	Provides information about the author's professional history.

2.1 ARTICLE Doctype-specific Tags

The ARTICLE doctype provides tags that let you perform the following functions:

- Create an article title and subtitle in either a one- or two-column format.
- Specify an author (or list of authors) for the article, and provide information about the author's professional history, address, affiliation or other information.
- Create abstracts, source notes, and acknowledgments.
- Specify whether primary headings should be numbered or unnumbered.
- Create running titles and running feet.
- Create quotations in which spacing and line breaks are retained as entered.
- Create back notes or reference notes which are automatically numbered and collated.
- Create bibliographies.

2.1.1 Titles and Subtitles

You can create a title for an article using the `<TITLE_SECTION>`, `<TITLE>`, and `<SUBTITLE>` tags. The `<TITLE>` and `<SUBTITLE>` tags are used to create titles and subtitles for an article. By default, these tags create a title or subtitle that is restricted to the first column of the article.

The `<TITLE_SECTION>` tag begins a title section that spans both columns of the article and enlarges the type faces output by the `<TITLE>` and `<SUBTITLE>` tags. Typically you would want to use only the `<TITLE>` and `<SUBTITLE>` tags in the context of the `<TITLE_SECTION>` tag, but you may also want to use the `<AUTHOR>` tag as well to have the author's name span both columns. Note that the output of the `<AUTHOR>` tag will not be enlarged.

In the following example, a title and subtitle are created that use the full page width of the article by using the `<TITLE_SECTION>` tag.

```
<TITLE_SECTION>
<TITLE>(A Guide to Instrument Care)
<SUBTITLE>(A Professional's View)
<ENDTITLE_SECTION>
```

You can have your title and subtitle appear only in the first column of the article by using them without the `<TITLE_SECTION>` tag.

2.1.2 Author Information

VAX DOCUMENT provides several tags to specify authors and author information in the ARTICLE doctype.

Use the `<AUTHOR>` and `<AUTHOR_LIST>` tags to specify one or more authors for an article. Use the following tags to specify information about an author:

- `<AUTHOR_ADDR>` specifies the address of the author
- `<AUTHOR_AFF>` specifies information about the professional or educational affiliations of the author
- `<VITA>` specifies the professional history of the author

These tags will appear in your output file in whatever order you place them in your SDML file, with the exception of the `<VITA>` tag, which places its text argument at the bottom of the current text column of output.

In the following example, a title, subtitle and author's name and professional history are created using the `<TITLE>`, `<SUBTITLE>`, `<AUTHOR>`, and `<VITA>` tags.

The output from this example would place the title, subtitle, and author's name at the top of the first column, and would place the text specified as the argument to the `<VITA>` tag at the bottom of the first column. The text of this example is assumed to be in the first paragraph of the article.

```
<TITLE>(Computer Graphics)
<SUBTITLE>(Everyone Likes It)
<AUTHOR>(G. R. Edwards)
<VITA>(G.R. Edwards has used graphics editors extensively.)
<P>Computer Graphics really are not for everyone, yet...
```

Using the ARTICLE Doctype

You can also use the author information tags within the context of the <TITLE_SECTION> tag. In the following example, the title, subtitle, and author's name are output using the full page width because these tags are used in the context of the <TITLE_SECTION> tag. The information on the author's affiliations, and then the text of the article, are output in a single column because those tags were used outside of the context of the <TITLE_SECTION> tag.

```
<TITLE_SECTION>
<TITLE>(Computer Graphics)
<SUBTITLE>(Everyone Likes It)
<AUTHOR>(G.R. Edwards)
<ENDTITLE_SECTION>
<AUTHOR_AFF>(G.R. Edwards is a senior consultant for Terminals, Inc.)
<P>Computer Graphics really are not for everyone, yet...
```

2.1.3 Abstracts, Source Notes, and Acknowledgments

Abstract, source note, and acknowledgment sections are special formats that typically occur at the beginning or end of an article, depending on your preference.

Abstracts

Use the <ABSTRACT> tag to specify an abstract for an article. You can specify the <ABSTRACT> tag either within the context of the <TITLE_SECTION> tag or following it.

In the following example, a short abstract is created within the context of the <TITLE_SECTION> tag. This causes the abstract to be formatted using the full page width rather than just a single column. The <AUTHOR> tag occurs outside of the context of the <TITLE_SECTION> tag and so is formatted using a single column.

```
<TITLE_SECTION>
<TITLE>(A Guide to Instrument Care)
<ABSTRACT>(A summary of brass and keyboard instrument care fundamentals by
a professional musician.)
<ENDTITLE_SECTION>
<AUTHOR>(Dan Dover)
```

Source Notes

The <SOURCE_NOTE> tag lets you specify the origination of material for an article. In the following example, the output from the <SOURCE_NOTE> tag is printed at the bottom of the current column of output.

```
<SOURCE_NOTE>(From the Boston Globe
<LINE>(<MCS>(COPYRIGHT) 1986 by the Boston Globe))
```

You can specify source information at the beginning or end of an article.

Acknowledgments

You can create a section for any acknowledgments you feel are necessary for your article using the <ACKNOWLEDGMENTS> tag. The text of the acknowledgment is entered as an argument to the <ACKNOWLEDGMENTS> tag. This tag outputs the heading "Acknowledgments" using the same format as the headings used by the <BACK_NOTES> and <REF_NOTES> tags.

The following example shows an acknowledgments section created using the `<ACKNOWLEDGMENTS>` tag. Note how it is used near the end of the SDML file with the `<BACK_NOTES>` and `<REF_NOTES>` tags.

```
<REF_NOTES>(Bibliography)
<BACK_NOTES>(References)
<ACKNOWLEDGMENTS>(I am deeply indebted to my doctor
for her support in this task.)
```

2.1.4 Headings

The ARTICLE doctype allows the global numbered heading tags (`<HEAD1>`, `<HEAD2>`, and so on) to provide a logical structure for your text. However, by default these headings are not numbered. You can specify that these headings be numbered by using the `<DOCUMENT_ATTRIBUTES>` tag as shown in the following example:

```
<DOCUMENT_ATTRIBUTES>
<SET_HEADINGS>(NUMBERED)
<ENDDOCUMENT_ATTRIBUTES>
```

In addition to the primary headings, you can use the global `<SUBHEAD1>` and `<SUBHEAD2>` tags to specify unnumbered paragraph topics or side headings as in the following example:

```
<SUBHEAD1>(Rationale.)<P>The purpose of this experiment...
```

2.1.5 Running Titles

You can place a title at the top or bottom of all the pages of your article using the `<RUNNING_TITLE>` and `<RUNNING_FEET>` tags.

The `<RUNNING_FEET>` tag accepts a single text argument, which it uses to create a title at the bottom of the page. The `<RUNNING_TITLE>` tag accepts one or two text arguments, which it uses to create a one- or two-line title at the top of the page.

The following example shows a two-line running title being set for the top of the page using the `<RUNNING_TITLE>` tag and a single line running title being set for the bottom of the page using the `<RUNNING_FEET>` tag.

```
<RUNNING_TITLE>(Mr. A. Author and\Mrs. B. Author)
<RUNNING_FEET>(The Story of Our Life Together)
```

2.1.6 Quotations

You can use either the ARTICLE doctype `<QUOTATION>` tag or the global `<SAMPLE_TEXT>` tag to place extended quotations in an article.

Use the `<QUOTATION>` tag to format text that you want to appear exactly as it is entered into the SDML file. The following example shows a Haiku poem formatted using the `<QUOTATION>` tag:

```
<P>A similar Haiku follows.
<QUOTATION>
    All lights are frozen;
    The cursor box blinks blandly.
    Soon, I see the dump.
<ENDQUOTATION>
```

Using the ARTICLE Doctype

Use the global `<SAMPLE_TEXT>` tag to create an extended quotation that is to be filled and justified in the text. You must supply any quotation marks, paragraph spacing, and so on. The following example shows how you can use the `<SAMPLE_TEXT>` tag to create an extended quotation:

As in the following text fragment:

```
<SAMPLE_TEXT>
<P><QUOTE>
Many are the ways of mankind. As some strive for recognition, others seek
obscurity. Surely, we are the strangest of creatures.
<ENDQUOTE>
<ENDSAMPLE_TEXT>
```

2.1.7 Back Notes and Reference Notes

You can use two types of automatically numbered notes in the ARTICLE doctype: back notes and reference notes. Back notes, sometimes referred to as end notes, are referenced in the text of an article using superscript numbers. Reference notes are similar to back notes, except that the references in the text are output using normal-sized numbers enclosed in brackets.

References to each type of note are accumulated while the article is being processed, and are output at the end of the article. You should use only one of these two types of note in your article.

You can also use the global `<FOOTNOTE>` tag to place footnotes at the bottom of a column of text. However, you should not use `<FOOTNOTE>` tag in an article in which you are using back notes. Both the `<FOOT_NOTE>` and the `<BACK_NOTE>` tags create superscript numbers for references, and that output would be extremely misleading and confusing.

2.1.7.1 Back Notes

Use the `<BACK_NOTE>` and the `<BACK_NOTES>` tags to create a set of back notes. The `<BACK_NOTE>` tag specifies the text of a note that is to be printed at the end of a document. You enter the `<BACK_NOTE>` tag in your SDML file wherever you want to provide a back note citation. VAX DOCUMENT sequentially numbers each of the back note entries and places the appropriate sequential number as a superscript in the output file.

For example, if you want to cite the book *Training Seagulls* as a back note, and this back note was the third in your document, the text where you cited the book would appear as follows:

These techniques are outlined in *Training Seagulls*³.

Place the `<BACK_NOTES>` tag in your SDML file at the point you want the accumulated back notes to be printed. When the `<BACK_NOTES>` tag is processed, all the accumulated notes are output, with their correct numbers and with the text you specified as arguments to the `<BACK_NOTE>` tags. The back notes are not automatically output at the end of the article so that you can control their position in the article.

The following example shows how to use the `<BACK_NOTE>` tag. The `<BACK_NOTE>` tag would be replaced by a superscript number in the output, and the note produced by that tag would be output near the end of the article using the `<BACK_NOTES>` tag.

```
As Ms. Roma so clearly stated <BACK_NOTE>(P.A. Roma,  
<QUOTE>(Computer-Chart Making  
from the Graphic Editor's Perspective,)  
<EMPHASIS>(ACM Computer Graphics, SIGGRAPH '99 Conf. Proc.),  
Vol 45. No. 3, July 1999, pp. 247-253.).
```

```
<BACK_NOTES>
```

2.1.7.2 Reference Notes

You can create bibliographic reference notes by using the `<REF_NOTE>`, `<REF_NOTES>` and optionally the global `<REFERENCE>` tags. Place the `<REF_NOTE>` tag in your SDML file at the point at which you want the reference to appear. This tag is replaced in the output by a number in brackets, which corresponds to the number assigned to the note text, for example, [4].

Use the `<REF_NOTES>` tag to cause the text of the reference notes you have created using the `<REF_NOTE>` tag to be output along with its assigned numbers. Typically this tag is placed at the end of the SDML file, but you can have the references appear earlier.

If you want to reference a source that you have already referenced using the `<REF_NOTE>` tag, you need to specify the *symbol-name* argument to that `<REF_NOTE>` tag and use the global `<REFERENCE>` tag to refer to that symbol.

The following example shows a reference note created using the `<REF_NOTE>` tag, a referral to that note using the global `<REFERENCE>` tag, and the printing of all the accumulated reference notes using the `<REF_NOTES>` tag. Note how the `<REF_NOTE>` tag was coded with the symbol `CHICAGO_MAN`, so that the subsequent `<REFERENCE>` tag could reference that symbol and use that same reference note number.

```
Sorting entries word by word is preferred  
<REF_NOTE>(<EMPHASIS>(A Manual of Style,)  
The University of Chicago Press, 1969.\CHICAGO_MAN).
```

```
<P>Overuse of emphasis can cause confusion <REFERENCE>(CHICAGO_MAN).
```

```
<REF_NOTES>(References)
```

If you want to create a bibliography that is not tied to references in the text of an article, you should use the `<BIBLIOGRAPHY>` tag.

2.1.8 Bibliographies

Use the `<BIBLIOGRAPHY>` tag to create a bibliography of related reading when you do not use numbered reference notes to reference other works in the text of the article. The `<BIBLIOGRAPHY>` tag enables the `<BIB_ENTRY>` tag and allows you to specify a heading for the bibliography as an argument to the `<BIBLIOGRAPHY>` tag.

Each of the entries in the bibliography are created by specifying the entry as an argument to the `<BIB_ENTRY>` tag. When you use this tag, you should use the `<EMPHASIS>` and `<QUOTE>` tags to specify the entry.

Using the ARTICLE Doctype

In the following example a bibliography is shown with two entries:

```
<BIBLIOGRAPHY>(Bibliography)
<p>The following may also be of interest:
<BIB_ENTRY>(EMPHASIS)(Molecular Connectivity in Chemistry
and Drug Research.)
Lamont B. Kier and Lowell H. Hall. Academic Press, 1983.)
<BIB_ENTRY>(Arhnhheim, Rudolph, EMPHASIS)(Visual Thinking).
University of California Press, Berkeley, 1984.)
<ENDBIBLIOGRAPHY>
```

Use the <REF_NOTE> and <REF_NOTES> tags to create numbered reference notes.

2.2 Improving the Format of a Two-Column Doctype

The ARTICLE and REPORT.TWOCOL doctypes let you create a two-column document. While these doctypes allow you to visualize what your document will look like when published, they are somewhat less flexible in terms of how they format SDML tags than the single-column doctypes. This section summarizes how you can improve the format of your two-column document.

2.2.1 Line Breaks in Columns

The width of the text column for paragraphs is much smaller in the two-column doctype than in the single-column doctypes. Furthermore, the left column is formatted right-justified. As you enter the text for your document into the SDML file, you need not be overly concerned about text paragraphs that exceed the right margin during text formatting. The text formatter issues the following message when a text line exceeds the right margin:

```
%TEX-W-LINETOOLONG_P, line too long ... in paragraph ...
```

As you near completion of the edits on your document, you may want to use the global <HYPHENATE> and <KEEP> tags to improve the line breaks in your printed document.

Use the global <HYPHENATE> tag to specify possible points of hyphenation in words the text formatter does not know how to hyphenate, but that you want to allow to hyphenate. This increases the number of places the text formatter can hyphenate the text, and so creates more even line breaks.

Use the global <KEEP> tag to specify text that you do not want hyphenated (broken across a line) by the text formatter. Use this tag as sparingly as possible because it decreases the number of places the text formatter can hyphenate the text, and so makes it difficult for the text formatter to create well-placed line breaks.

The text formatter constructs more well-formatted text lines in each column when it has more places at which it can hyphenate words in the text. The more places you allow the text formatter to hyphenate your text, the better your final output will be formatted.

2.2.2 Wide Tables and Examples

When developing examples and tables using a two-column doctype, you should be careful of the following conditions:

- Monospaced or unformatted output created using the `<CODE_EXAMPLE>` or `<QUOTATION>` tags that exceed the column width
- The width of tables and figures (if your figures include monospaced examples or art)

If you have a table, figure, or a monospaced example that is wider than the text column width, you can use the `WIDE` argument when you specify attributes for the tag.

When the `<TABLE_ATTRIBUTES>` or `<FIGURE_ATTRIBUTES>` tag is specified with the `WIDE` keyword to create a wide table or figure, that table or figure causes the two-column output to be suspended and the text entered before that table or figure will be placed in the two columns above the table or figure.

The table or figure then will be output using the full page width, as if they occurred in a single-column doctype. Two-column formatting is restored after the table or figure ends, and the text after the table or figure begins again in the first column under the table or figure.

2.2.3 Final Adjustment of Column and Page Breaks

When you are working with a two-column doctype (such as `ARTICLE` or `REPORT.TWOCOL`), you may need to make some final adjustments to your paged output when your text is complete. This is because it is difficult to create good-looking pages within the constraints of a two-column document. Occasionally, a two-column document will need to have explicit line, column, and page breaks inserted into it to improve its appearance.

You can use the `<FINAL_CLEANUP>` tag in all doctypes to specify where text lines and pages should break by using the `LINE_BREAK` and `PAGE_BREAK` keywords as arguments. In two-column doctypes, you can begin a new column of text by using the `COLUMN_BREAK` keyword to the `<FINAL_CLEANUP>` tag, or the `<COLUMN>` tag, which is available in the `ARTICLE` and `REPORT.TWOCOL` doctypes.

Column Breaks

When the text formatter creates a two-column page, it breaks the text into two columns so as to create a page in which the columns are of as nearly equal length as possible (this is sometimes called “bottoming out”). Certain text elements (such as tables and figures) cannot be easily broken across columns. The text formatter uses vertical space to adjust the length of the columns, therefore, you may see large amounts of vertical white space preceding and following those text elements that accept a variable amount of white space (for example, headings, lists and tables).

You can specify that columns be explicitly broken by using the `<COLUMN>` or `<FINAL_CLEANUP>` (`COLUMN_BREAK`) tags. You should use the `<COLUMN>` tag only when you want the subsequent text to always begin a new column, regardless of any changes you make to the text. You should use the `<FINAL_CLEANUP>` (`COLUMN_BREAK`) tag only after your text is finished

Using the ARTICLE Doctype

and you want to improve the appearance of your document by specifying a new column of text.

In either case, if the current text is in the first column of a page, the request to start a new column places the next text in the second column. If the current text is in the second column of a page, the request results in a new page of output.

In some circumstances, the output of a two-column page may appear to have lost vertical space before a text element; for example, a heading tag may have no space before it. When this occurs in processing a two-column doctype, you should ignore the occurrence until you are ready to give your document a final go-over. If the space is still being lost, use the `<FINAL_CLEANUP>` (`SPECIAL_BREAK`) tag. For example, suppose the following lines represent fragments of a two-column page:

```
-----
Major Heading
-----
-----
-----
-----
-----
-----
-----
-----
-----
Next Heading
-----
-----
-----
-----
end of text in this
```

In the previous example, the spacing appears to be lost above the heading "Next Heading." You can correct this by placing the `<FINAL_CLEANUP>` (`SPECIAL_BREAK`) tag in the SDML file between the words that are output on the final line of the first column as in the following example:

```
<P>An example shows what happens to the end of
<FINAL_CLEANUP> (SPECIAL_BREAK)
text in this column.
```

You should need to use this special column break only in rare instances.

Page Breaks

A new page of output is explicitly started whenever the following conditions exist:

- A `<COLUMN>` or `<FINAL_CLEANUP>` (`COLUMN_BREAK`) occurs in the right text column and so results in a new page of output
- A `<FINAL_CLEANUP>` (`PAGE_BREAK`) tag is used to request that text start on a new page

In either of these situations, the current page is set in two columns, without balancing the columns. The length of the text in either column may be less than that of the regular balanced page.

2.3 A Sample Use of the ARTICLE Doctype Tags

This section contains an example of an article created using the ARTICLE doctype tags. You may find this sample useful in understanding how the tags can be used to create two-column articles.

The SDML code for the article is shown first, followed by the output from that SDML code.

```
<TITLE_SECTION>
<TITLE>(MUSIC, MAESTRO)
<SUBTITLE>(A Guide to Instrument Care)
<ENDTITLE_SECTION>

<AUTHOR_LIST>(By)
<AUTHOR>(Dan Dover)
<AUTHOR>(Dunstan Frobisher)
<ENDAUTHOR_LIST>

<P>Musical instruments of any kind can bring years of enjoyment to both the
player, and hopefully the listener, as long as the musical instrument is cared
for properly. Where diligence and discipline can make one a proficient
player, care and cleaning can also make one an efficient player.

<P>This article discusses the basic care of several musical instruments which
can be carried over to other musical instruments that will not be discussed
here.

<CHEAD>(Keyboard Instruments)

<P>The first rule in caring for any keyboard instrument is <EMPHASIS>(Are your
hands clean?) <REF_NOTE>( <EMPHASIS>(Tickling the Ivories: Piano for Beginners),
Architect Press, 1982.). Peanut butter and jelly tends to make the keys stick.
Also, grime and dirt will scratch the keys and get between them as well.

<P>Even the natural oils of your hand have a detrimental effect on the
keyboard. It is always a good idea to wash your hands before playing the piano,
organ, or other keyboard instrument. And after you are through playing, take a
warm, damp cloth and wipe down the keyboard. This removes any residual hand
oil from the keys.

<P>The second rule for keyboard care is <EMPHASIS>(tuning). Like Mary Edith
Whiteout of the Hanscom Music Company says:

<QUOTATION>
  You can tell the quality of a pianist
  by the pitch of their instrument.
  A well-tuned piano is as much a joy,
  as a badly-tuned piano is a horror.
<ENDQUOTATION>

<P>Have your piano tuned every six months (for the average piano player); if
you play more than four hours a day, a tune up every three to four months is
recommended.

<P>If your organ, or your accordion goes out of tune, take it to a repairman
and get the offending note fixed. In summary, basic care for your keyboard
instrument entails:

<LIST>(NUMBERED)
  <LE>Clean hands and a clean instrument; wash your hands before, wash the
  keyboard after
  <LE>Tune your instrument regularly; 6 months - average use, 3 to 4 months
  for heavy use
<ENDLIST>

<CHEAD>(Brass Instruments)
```

Using the ARTICLE Doctype

<P>The first rule in caring for any brass instrument is <EMPHASIS>(Is your mouth clean?) Be sure to brush your teeth and rinse your mouth if you are going to play the trumpet <REF_NOTE>(<EMPHASIS>(Trumpeter Lullaby: Caring for Your Horn), County Ecks Press, 1985), trombone <REF_NOTE>(<EMPHASIS>(Trombone Exercises) Emerald Books, 1983), or other brass instrument. Food particles left in your mouth will foul up the valves and slides. It may even cause a restriction of the air flow, the instrument to go out of tune, or even damage it permanently.

<P>The second rule is <EMPHASIS>(Oil your valves and slides regularly.) Use the recommended oil for your instrument. This will ensure that part movement happens smoothly and quickly.

<P>The third rule is to <EMPHASIS>(Polish your instrument after each use) with a warm, damp cloth. This will help keep it from tarnishing from the natural oils in your hand. In addition to this, you should use a recommended brass polish every month. In summary, basic care for your brass instrument entails:

<LIST>(NUMBERED)

<LE>A clean mouth

<LE>Oiled valves and slides

<LE>Polishing on a regular basis

<ENDLIST>

<REF_NOTES>(Additional Reading)

This example may produce the following output if processed using the ARTICLE keyword.

Sample Output of Article Doctype

MUSIC, MAESTRO

A Guide to Instrument Care

By

Dan Dover

Dunstan Frobisher

Musical instruments of any kind can bring years of enjoyment to both the player, and hopefully the listener, as long as the musical instrument is cared for properly. Where diligence and discipline can make one a proficient player, care and cleaning can also make one an efficient player.

This article discusses the basic care of several musical instruments which can be carried over to other musical instruments that will not be discussed here.

Keyboard Instruments

The first rule in caring for any keyboard instrument is *Are your hands clean?* [1]. Peanut butter and jelly tends to make the keys stick. Also, grime and dirt will scratch the keys and get between them as well.

Even the natural oils of your hand have a detrimental effect on the keyboard. It is always a good idea to wash your hands before playing the piano, organ, or other keyboard instrument. And after you are through playing, take a warm, damp cloth and wipe down the keyboard. This removes any residual hand oil from the keys.

The second rule for keyboard care is *tuning*. Like Mary Edith Whiteout of the Hanscom Music Company says:

You can tell the quality of a pianist
by the pitch of their instrument.

A well-tuned piano is as much a joy,
as a badly-tuned piano is a horror.

Have your piano tuned every six months (for the average piano player); if you play more than four hours a day, a tune up every three to four months is recommended.

If your organ, or your accordion goes out of tune, take it to a repairman and get the offending note fixed. In summary, basic care for your keyboard instrument entails:

1. Clean hands and a clean instrument; wash your hands before, wash the keyboard after
2. Tune your instrument regularly; 6 months - average use, 3 to 4 months for heavy use

Brass Instruments

The first rule in caring for any brass instrument is *Is your mouth clean?* Be sure to brush your teeth and rinse your mouth if you are going to play the trumpet [2], trombone [3], or other brass instrument. Food particles left in your mouth will foul up the valves and slides. It may even cause a restriction of the air flow, the instrument to go out of tune, or even damage it permanently.

The second rule is *Oil your valves and slides regularly*. Use the recommended oil for your instrument. This will ensure that part movement happens smoothly and quickly.

The third rule is to *Polish your instrument after each use* with a warm, damp cloth. This will help keep it from tarnishing from the natural oils in your hand. In addition to this, you should use a recommended brass polish every month. In summary, basic care for your brass instrument entails:

1. A clean mouth
2. Oiled valves and slides
3. Polishing on a regular basis

Additional Reading

- [1] *Tickling the Ivories: Piano for Beginners*, Architect Press, 1982.
- [2] *Trumpeter Lullaby: Caring for Your Horn*, County Ecks Press, 1985
- [3] *Trombone Exercises* Emerald Books, 1983

3

Using the LETTER Doctype

The LETTER doctype lets you create various types of correspondence such as business letters, personal letters, and memos in an $8\frac{1}{2} \times 11$ inch format. The LETTER doctype has a single design. You process files under this doctype by using the LETTER doctype keyword on the DOCUMENT command line.

The LETTER doctype does not support the full set of VAX DOCUMENT global tags. The following tags are not available in the LETTER doctype:

- <APPENDIX>
- <CHAPTER>
- <FRONT_MATTER>
- <HEAD1> through <HEAD6>
- <PART_PAGE>

Even though the LETTER doctype does not support the global numbered heading tags (<HEAD1> , <HEAD2> , and so on), it does support the global unnumbered heads (<SUBHEAD1> and <SUBHEAD2>). See the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags.

The LETTER doctype-specific tags allow you to label and format the text elements of letters and memos. In general, the LETTER doctype tags whose tag names are prefixed with "MEMO_" (for example, the <MEMO_TO> tag) are used to create memos; the other tags in this doctype are used to create letters or headings in either memos or letters. You can use the LETTER doctype tags in whatever order you choose. No one tag is restricted to either a memo or a letter format.

Table 3-1 summarizes the tags available in the LETTER doctype. Chapter 10 contains the reference information on the tags listed in this table.

Table 3-1 Tags Available in the LETTER Doctype

Tag Name	Description
<CC>	Labels the name of one person who is to receive a copy of the memo or letter. This tag places the heading "cc:" on the left margin, and then places the name of the person to the right of this heading on the same line.
<CCLIST>	Begins a list of one or more persons' names who are to receive a copy of the memo or letter. This tag places the heading "cc:" on the left margin. The <CC> tag is used to label each of the names in the list within the context of the <CCLIST> tag.
<CHEAD>	Creates an unnumbered centered heading in a memo or letter.
<CLOSING>	Labels the closing text of a letter and formats the closing text flush left just to the right of the middle of the page. This text is aligned with the output of the <FROM_ADDRESS> tag. This text is typically a closing line such as "Yours Truly," followed by the name of the sender.

Using the LETTER Doctype

Table 3–1 (Cont.) Tags Available in the LETTER Doctype

Tag Name	Description
<DISTLIST>	Begins a list of people to whom the memo or letter is to be distributed. This list is formatted on the left margin beneath a heading of "Distribution:."
<FROM_ADDRESS>	Identifies the name and address of the sender of a letter and formats that information flush left just to the right of the middle of the page. This text is aligned with the output of the <CLOSING> tag.
<HEAD>	Creates an unnumbered heading on the left margin of a memo or letter.
<MEMO_DATE>	Labels the date of a memo and formats that date flush left near the left margin. This tag places the heading "Date:" on the left margin.
<MEMO_FROM>	Identifies the name and address of the sender of a memo and formats that information flush left near the left margin. This tag places the heading "From:" on the left margin.
<MEMO_HEADER>	Centers the heading "Interoffice Memorandum" on the current line of the output page.
<MEMO_LINE>	Lets you specify your own information and headings in a format similar to the format used by the <MEMO_TO> or <MEMO_FROM> tags. This tag places your heading on the left margin and then places the first line of information text to the right of that heading on the same line; additional lines of information are formatted under the first.
<MEMO_TO>	Identifies the name and address of the sender of a memo and formats that information flush left near the left margin. This tag places the heading "To:" on the left margin.
<SALUTATION>	Labels the greeting portion of a letter and formats that greeting on the left margin.
<SUBJECT>	Labels the subject of a memo or letter and formats that information flush left near the left margin. This tag places the heading "Subject:" on the left margin. The subject text is formatted on the same line and to the right of the heading.
<TO_ADDRESS>	Identifies the name and address of the receiver of a letter and formats that information flush left on the left margin.

3.1 Sample Uses of the LETTER Doctype Tags

This section contains two examples of the LETTER doctype tags. The first example shows a sample memo and the second example shows a sample letter. You may find these sample files useful in understanding how the tags all fit together to create memos and letters.

3.1.1 A Sample Memo

The SDML code for a memo is shown first, followed by the output from that SDML code.

```

<MEMO_HEADER>
<MEMO_FROM>(Mr. Smith\Corporate Company Accounting)
<MEMO_LINE>(Phone\181-1546)
<MEMO_TO>(Jack Jones\Payroll Accounting)
<MEMO_DATE>(March 17, 1987)
<CCLIST>
<CC>(Jim Walker)
<CC>(John Beam)
<CC>(D. M. Bones)
<ENDCCLIST>
<CC>(Departmental Distribution)
<SUBJECT>(Conference Report)
<P>This conference was hosted by Numbers Inc. in Seattle, Wash.,
March 4 through 7. The goal of the conference was to stimulate the
development of accounting technology.
<P>My goals for attending the conference were to learn as much as
I could about accounting technology and to find out about existing
products or projects related to accounting methodology.
<SUBHEAD1>(Summary of Presentations Attended)
<P>
Major opening and closing presentations were directed at all conference
attendees. In between, there were choices between technical sessions and
general sessions, and I almost always felt a conflict. It was especially
annoying because there were not many clues as to what the differences were.
Sometimes technical was way too technical and general was way
too general.
<DISTLIST>
Bert Tom
Harry Lisa
Jim Melinda
Walter Jess
*All Trainees*
<ENDDISTLIST>

```

This example may produce the following output.

Using the LETTER Doctype

Sample Memo Created Using Letter Doctype

INTEROFFICE MEMORANDUM

From: Mr. Smith
Corporate Company Accounting
Phone: 181-1546

To: Jack Jones
Payroll Accounting
Date: March 17, 1987

cc: Jim Walker
John Beam
D. M. Bones

cc: Departmental Distribution

Subject: Conference Report

This conference was hosted by Numbers Inc. in Seattle, Wash., March 4 through 7. The goal of the conference was to stimulate the development of accounting technology.

My goals for attending the conference were to learn as much as I could about accounting technology and to find out about existing products or projects related to accounting methodology.

Summary of Presentations Attended

Major opening and closing presentations were directed at all conference attendees. In between, there were choices between technical sessions and general sessions, and I almost always felt a conflict. It was especially annoying because there were not many clues as to what the differences were. Sometimes technical was way too technical and general was way too general.

DISTRIBUTION:

Bert Tom
Harry Lisa
Jim Melinda
Walter Jess
All Trainees

3.1.2 A Sample Letter

The SDML code for a letter is shown first, followed by the output from that SDML code.

```
<FROM_ADDRESS>(Harvard University\Cambridge, MA\January 1, 1987, 10:00am EST)
<TO_ADDRESS>(Carol Jones\World Wide Wickets Co.\Seattle, WA)
<SALUTATION>(Hi Carol,)
<P>
This is an excerpt from a symposium I went to on letter writing.
I thought you might find it interesting. We really ought to have lunch
some time.
<P>
The excerpt follows:
<P>
There are generally two kinds of letters:
<LIST>(UNNUMBERED)
<LE> Business letters
<LE> Personal letters
<ENDLIST>

<HEAD>(Writing a Business Letter)
<P>
When writing a business letter, form can be very important.
In many cases, the form of the letter can be nearly as important as the content
of the letter.

<CHEAD>(Writing a Letter to Request Information)
<P>
A business letter is often used to request information from
an official source. It is important to specify very clearly what
information you need, and for what you need it. If your information needs
are unclear, your request may not be filled.

<CLOSING>(Best Wishes,\Bob Smith\Chairman, CZZA Committee)
```

This example may produce the following output.

Using the LETTER Doctype

Sample Letter Created Using Letter Doctype

Harvard University
Cambridge, MA
January 1, 1987, 10:00am EST

Carol Jones
World Wide Wickets Co.
Seattle, WA

Hi Carol,

This is an excerpt from a symposium I went to on letter writing. I thought you might find it interesting. We really ought to have lunch some time.

The excerpt follows:

There are generally two kinds of letters:

- Business letters
- Personal letters

Writing a Business Letter

When writing a business letter, form can be very important. In many cases, the form of the letter can be nearly as important as the content of the letter.

Writing a Letter to Request Information

A business letter is often used to request information from an official source. It is important to specify very clearly what information you need, and for what you need it. If your information needs are unclear, your request may not be filled.

Best Wishes,

Bob Smith
Chairman, CZZA Committee

7

Using the REPORT Doctype

The REPORT doctype lets you create general-purpose documents such as reports and formal outlines. The REPORT doctype has two designs. These doctype designs are listed by their doctype keyword:

- **REPORT**
Creates an $8\frac{1}{2} \times 11$ inch format with unruled numbered and unnumbered headings.
- **REPORT.TWOCOL**
Creates the same format as REPORT, but places the text in two columns.

This doctype accepts the full range of VAX DOCUMENT global tags (see the *VAX DOCUMENT User Manual, Volume 1* for more information on the global tags). The REPORT doctype also provides additional tags that let you perform the following functions:

- Create headings or modify the default attributes of the REPORT doctype.
- Create signature lines and list author information within the context of the global <FRONT_MATTER> tag
- Create formal outlines within the context of the <OUTLINE> tag

You process a file with the REPORT doctype by using the REPORT or REPORT.TWOCOL doctype keyword on the VAX DOCUMENT command line. See Section 2.2 for information on the special formatting considerations of a two-column doctype.

See *VAX DOCUMENT Design Samples* for more information on the format of the REPORT designs.

Table 7-1 summarizes the tags available in the REPORT doctype and provides a brief description of each tag. Chapter 13 contains the reference information on the tags listed in this table.

Table 7-1 Tags Available in the REPORT Doctype

Tag Name	Description
Tags Available in the Front Matter	
<AUTHOR>	Places the name of an author and up to two additional lines of information about the author on the output page.
<BYLINE>	Creates a rule to be used as a signature line, and places the name of the signatory beneath the line.
<SIGNATURES>	Begins a listing of signature lines created by the <BYLINE> tag. Optionally, you can use this tag to begin the listing of signature lines on a new page.

Using the REPORT Doctype

Table 7-1 (Cont.) Tags Available in the REPORT Doctype

Tag Name	Description
Tags Available Throughout the Document	
<CHEAD>	Creates an unnumbered centered heading.
<COLUMN>	Specifies that a new column of output should begin in a two-column doctype.
<DOCUMENT_ATTRIBUTES>	Modifies the numbering of pages and formal elements in the document.
<HEAD>	Creates an unnumbered heading, which is placed on the left margin.
<RUNNING_FEET>	Places a heading at the bottom of each page.
<RUNNING_TITLE>	Places a heading at the top of each page.
<SECTION>	Begins a new page and places an unnumbered heading at the top of the new page on the left margin.
Tags Available to Create Outlines	
<LEVEL>	Specifies an entry in an outline.
<OUTLINE>	Enables the <LEVEL> and <SHOW_LEVELS> tags and lets you specify a title for the outline.
<SHOW_LEVELS>	Emphasizes text within the outline using either bolding or italics.

7.1 A Sample Use of the REPORT Doctype Tags

This section contains an example of the first few pages of a report created using the REPORT doctype tags. This report includes a front matter section and an outline in the body of the report. Note how the outline and front matter tags are used in this example. You may find this sample useful in understanding how the tags all fit together to create reports and other general-purpose documents.

The SDML code for the report is shown first, followed by the output from that SDML code.

```
<FRONT_MATTER>
<TITLE_PAGE>
<TITLE>(Equipment Usage in this Company)
<ABSTRACT>
This is an internal report on equipment usage during
the period (May 1986 - November 1986).
<ENDABSTRACT>
<AUTHOR>(Thomas A. Smith\Comptroller\Eastern Division)
<SIGNATURES>
<BYLINE>(T. A. Smith)
<BYLINE>(John Whorfin\Accounting Consultant)
<DATE>
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
<CHAPTER>(Equipment Usage Summary)
<P>Equipment usage is a very important quantity to monitor.
If equipment is not used, it is a wasted resource. If equipment is over-used,
it tends to break sooner, and means that people must wait to use it. If people
are waiting, they are not being as productive as they might otherwise be.
<P>The following sections summarize equipment usage in various departments.
```

Using the REPORT Doctype

```
<HEAD1>(Usage of Official Vehicles)
<P>
Official vehicle usage is listed in a separate report CORP-AUTO-1439u2.
This report is organized as in the following outline. Note that there are two
new categories in the report. These categories are italicized in the following
outline.
<OUTLINE>(Outline of Report\CORP-AUTO-1439u2\Motor Vehicle Usage)
<LEVEL>(1\Four wheeled Vehicles)
<LEVEL>(2\Cars)
<LEVEL>(2\Trucks)
<SHOW_LEVELS>(ITALIC)
<LEVEL>(3\Heavy trucks)
<LEVEL>(3\Light trucks (less than 2 ton))
<SHOW_LEVELS>(OFF)
<LEVEL>(2\Vans)
<ENDOUTLINE>
```

This example may produce the following output if processed using the REPORT keyword.

Using the REPORT Doctype

Sample Output of Report Doctype

Equipment Usage in this Company

This is an internal report on equipment usage during the period (May 1986 - November 1986).

Thomas A. Smith
Comptroller
Eastern Division

T. A. Smith

John Whorfin—Accounting Consultant
26-November-1986

Sample Output of Report Doctype

CHAPTER 1
EQUIPMENT USAGE SUMMARY

Equipment usage is a very important quantity to monitor. If equipment is not used, it is a wasted resource. If equipment is over-used, it tends to break sooner, and means that people must wait to use it. If people are waiting, they are not being as productive as they might otherwise be.

The following sections summarize equipment usage in various departments.

1.1 Usage of Official Vehicles

Official vehicle usage is listed in a separate report CORP-AUTO-1439u2. This report is organized as in the following outline. Note that there are two new categories in the report. These categories are italicized in the following outline.

Outline of Report
CORP-AUTO-1439u2
Motor Vehicle Usage

- I. Four wheeled Vehicles
 - A. Cars
 - B. Trucks
 - 1. *Heavy trucks*
 - 2. *Light trucks (less than 2 ton)*
 - C. Vans

9

ARTICLE Doctype Tag Reference

This chapter provides the reference information on the tags available within the ARTICLE doctype. These tags, which allow you to create articles in a two-column format, are described alphabetically in this chapter. Chapter 2 provides a tutorial on how to use these tags.

The ARTICLE doctype supports all of the VAX DOCUMENT global tags with the following exceptions:

- <CHAPTER>
- <FRONT_MATTER>
- <REVISION> , <MARK> , and <UPDATE_RANGE>
- The MULTIPAGE keyword is not available as an argument to the <EXAMPLE_ATTRIBUTES> , <FIGURE_ATTRIBUTES> and <TABLE_ATTRIBUTES> tags. This means that formal examples, figures and tables are restricted to a length of one page or less.

See the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags. See *VAX DOCUMENT Design Samples* for more information on the page layout of ARTICLE doctype.

ARTICLE Doctype Tag Reference

<ABSTRACT>

<ABSTRACT>

Creates an article abstract and can also specify a heading for that abstract.

FORMAT **<ABSTRACT>** [(*abstract-heading*)]

ARGUMENTS *abstract-heading*
Specifies a heading for the abstract. If no heading is specified, none is output.

related tags

- The global <ABSTRACT> tag
- <TITLE_SECTION>

restrictions *None.*

required terminator <ENDABSTRACT>

DESCRIPTION The <ABSTRACT> tag is used to create an abstract of an article. This tag can be used within the context of the <TITLE_SECTION> tag, or following the <TITLE_SECTION> tag within the ARTICLE doctype.

This tag is used in a different manner from the global <ABSTRACT> tag, which is used in any doctype other than the ARTICLE doctype within the context of the <TITLE_PAGE> tag.

EXAMPLE The following example shows how to create an abstract. It is headed with the title "Editor's Note:."

```
<ABSTRACT>(Editor's Note:)
The following is an excerpt from the paper,
<QUOTE>(Creating a Lasting Impression.)
See the bibliography at the end of this article for information on how to
locate this paper.
<ENDABSTRACT>
```

<ACKNOWLEDGMENTS>

Creates an acknowledgments section in an article.

FORMAT <ACKNOWLEDGMENTS> (*acknowledgment-text*)

ARGUMENTS *acknowledgment-text*
Specifies the text of one or more acknowledgments.

related tags • <BACK_NOTES>
 • <REF_NOTES>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <ACKNOWLEDGMENTS> tag outputs the heading "Acknowledgments" using the same format as the <BACK_NOTES> and <REF_NOTES> tags. The text of the acknowledgment is entered as an argument to the <ACKNOWLEDGMENTS> tag.

The <ACKNOWLEDGMENTS>, <BACK_NOTES>, and <REF_NOTES> tags are typically entered at the end of an SDML file to be processed using the ARTICLE doctype.

EXAMPLE The following example shows a sample use of the <ACKNOWLEDGMENTS> tag. Note how it is used near the end of the SDML file with the <BACK_NOTES> and <REF_NOTES> tags.

```
<REF_NOTES>(Bibliography)
<BACK_NOTES>(References)
<ACKNOWLEDGMENTS>(I am deeply indebted to my doctor for her support in this task.)
```

ARTICLE Doctype Tag Reference

<AUTHOR>

<AUTHOR>

Specifies an author of an article.

FORMAT <AUTHOR> (*author-name*[\ *optional-information*])

ARGUMENTS *author-name*

Specifies the name of the author. If you want the word “By” included with the author’s name, you must specify it as part of the author-name text.

optional-information

Specifies additional optional information about the author. This information is formatted below the author’s name. This text should be approximately one line in length.

related tags

- <AUTHOR_ADDR>
- <AUTHOR_AFF>
- <AUTHOR_LIST>
- <VITA>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <AUTHOR> tag to place the name of the author in an article. The name of the author is given as the first argument to the <AUTHOR> tag; additional information about the author can be specified as the second argument to the <AUTHOR> tag.

The following list includes the other tags that allow you to specify information about the author.

- | | |
|---------------|--|
| <AUTHOR_ADDR> | Specifies an address for the author. |
| <AUTHOR_AFF> | Specifies the organizational affiliation of the author. |
| <AUTHOR_LIST> | Lists multiple authors of an article. |
| <VITA> | Provides information about an author’s professional history. |

See the appropriate tag description in this chapter for more information on any of these tags.

ARTICLE Doctype Tag Reference

<AUTHOR>

EXAMPLE

The following example shows how to use the <AUTHOR> tag in a file processed using the ARTICLE doctype. Note how the optional second argument to the <AUTHOR> tag is used to give additional information about the author.

```
<AUTHOR>(By A.B. Roma\Publisher)
<AUTHOR_AFF>(<emphasis>(Disco Monthly) staff)
<AUTHOR_ADDR>(Top-Ten Corporation,\
5300 Westlake Boulevard,\
Los Angeles, California 09945)
```


ARTICLE Doctype Tag Reference <AUTHOR_ADDR>

EXAMPLE

The following example shows how to provide address information about the author of an article.

```
<AUTHOR>(A. B. Roma)  
<AUTHOR_AFF><emphasis>(Disco Monthly) staff)  
<AUTHOR_ADDR>(Top-Ten Corporation,\  
5300 Westlake Boulevard,\  
Los Angeles, California 09945)
```

ARTICLE Doctype Tag Reference

<AUTHOR_AFF>

<AUTHOR_AFF>

Specifies information about the organizational affiliation of the author.

FORMAT <AUTHOR_AFF> (*affiliation-information*)

ARGUMENTS *affiliation-information*
Specifies information about the affiliation of the author.

related tags

- <AUTHOR>
- <AUTHOR_ADDR>

restrictions *None.*

required terminator *None.*

DESCRIPTION Use the <AUTHOR_AFF> tag to describe the professional or educational affiliation of the author of an article. This tag is typically entered after the <AUTHOR> tag in the SDML file.

The following list includes the other tags that allow you to specify information about the author.

<AUTHOR>	Specifies an author for an article.
<AUTHOR_ADDR>	Specifies an address for the author.
<AUTHOR_LIST>	Lists multiple authors of an article.
<VITA>	Provides information about an author's professional history.

See the appropriate tag description in this chapter for more information on any of these tags.

EXAMPLE The following example shows how to provide information about the affiliation of the author of an article.

```
<AUTHOR>(A. B. Roma)
<AUTHOR_AFF>(<emphasis>(Disco Monthly) staff)
<AUTHOR_ADDR>(Top-Ten Corporation,\
5300 Westlake Boulevard,\
Los Angeles, California 09945)
```

<AUTHOR_LIST>

Creates a list of authors for an article with multiple authors.

FORMAT <AUTHOR_LIST> [(*heading-text*)]

ARGUMENTS *heading-text*

Provides an introductory heading for a list of authors. A sample heading might be "By:."

related tags • <AUTHOR>

restrictions *None.*

required terminator <ENDAUTHOR_LIST>

DESCRIPTION You can use the <AUTHOR_LIST> tag to place the name of multiple authors in an article. You can also optionally specify a heading for the list of authors as an argument to the <AUTHOR_LIST> tag. The name of each author is given as the first argument to the <AUTHOR> tag, and additional information about the author can be specified as the second argument to the <AUTHOR> tag.

The following list includes the other tags that allow you to specify information about the author.

<AUTHOR>	Specifies an author for an article.
<AUTHOR_ADDR>	Specifies an address for the author.
<AUTHOR_AFF>	Specifies the organizational affiliation of the author.
<VITA>	Provides information about an author's professional history.

See the appropriate tag description in this chapter for more information on any of these tags.

EXAMPLE The following example shows a list of two authors introduced by the word "By:."

```
<AUTHOR_LIST>(By:)
<AUTHOR>(Mary Smith,\User Publications)
<AUTHOR>(Tom Jones,\Software Engineering)
<ENDAUTHOR_LIST>
```

ARTICLE Doctype Tag Reference

<BACK_NOTE>

<BACK_NOTE>

Creates a back note entry and a superscript reference number to that entry in the article text.

FORMAT <BACK_NOTE> (*back-note-text*)

ARGUMENTS *back-note-text*

Specifies the text to be associated with the back note entry.

related tags

- <BACK_NOTES>
- <REF_NOTE>
- The global <FOOTNOTE> tag

restrictions

The <BACK_NOTE> tag should not be used in the same document as the global <FOOTNOTE> tag when the <FOOTNOTE> tag is being used to create numbered footnotes, since both tags would place superscript numbers into a document, and references to those numbers could be confusing.

required terminator

None.

DESCRIPTION

The <BACK_NOTE> tag allows you to create an entry for a back note (also referred to as an "end note") in your document. You enter the <BACK_NOTE> tag in your SDML file wherever you want to provide a back note citation. VAX DOCUMENT sequentially numbers each of the back note entries and places the appropriate sequential number as a superscript in the output file.

For example, if you want to cite the book *Training Seagulls* as a back note, and this back note is the third in your document, the text where you cited the book would appear as follows:

These techniques are outlined in *Training Seagulls*³.

VAX DOCUMENT collects all the automatically numbered back note entries and outputs them together in their sequential order wherever you use the <BACK_NOTES> tag. Typically, you will want to use the <BACK_NOTES> tag at the end of your article.

ARTICLE Doctype Tag Reference <BACK_NOTE>

EXAMPLE

The following example shows a reference to a back note and the text of the note as it is to appear at the end of the document. The <BACK_NOTES> tag at the conclusion of the article causes this note, and any others, to be output.

As Ms. Roma so clearly stated.<BACK_NOTE>(P.A. Roma,
<QUOTE>(Computer-Chart Making from the Graphic Editor's Perspective,)
<EMPHASIS>(ACM Computer Graphics, SIGGRAPH '99 Conf. Proc.),
Vol 45. No. 3, July 1999, pp. 247-253.)

<BACK_NOTES>

ARTICLE Doctype Tag Reference

<BACK_NOTES>

<BACK_NOTES>

Causes any accumulated back notes to be output.

FORMAT **<BACK_NOTES>** [(*heading-text*)]

ARGUMENTS ***heading-text***

Specifies the text that is output above the back note section. This heading will have the default heading format of the <HEAD1> tags in a document using unnumbered heads. If you do not specify the *heading-text* argument, no heading is output.

related tags • <BACK_NOTE>

restrictions *None.*

required terminator *None.*

DESCRIPTION You can use the <BACK_NOTES> tag to create a list of back note entries that were created using the <BACK_NOTE> tag. The <BACK_NOTES> tag causes these entries to be placed in the output file at the point where the <BACK_NOTES> tag occurs in the SDML file.

You can specify a heading for these back notes as an argument to the <BACK_NOTES> tag. Alternatively, you can specify your own heading, with the heading tag (<HEAD1>, <HEAD2>, and so on) that is appropriate for your document. The heading level tag must precede the <BACK_NOTES> tag as shown in the second of the examples below.

EXAMPLES

The following example shows how you can create the heading "References" for a list of back notes by coding that heading as an argument to the <BACK_NOTES> tag.

1 <BACK_NOTES>(References)

The following example shows how you can create the second level heading "Articles" for a list of back notes by coding that heading as an argument to the <HEAD2> tag, and placing that tag immediately before the <BACK_NOTES> tag.

2 <HEAD2>(Articles)
 <BACK_NOTES>

<BIBLIOGRAPHY>

Begins a bibliography.

FORMAT <BIBLIOGRAPHY> [(*heading-text*)]

ARGUMENTS *heading-text*

Specifies the heading for a bibliography. This heading will appear in the format used by <HEAD1> tags in an article with unnumbered heads. By default, no heading is output.

related tags

- <BIB_ENTRY>
- <REF_NOTE>
- <REF_NOTES>

restrictions

None.

required terminator

<ENDBIBLIOGRAPHY>

DESCRIPTION

You can use the <BIBLIOGRAPHY> tag to create a bibliography of related reading when you do not use numbered reference notes to reference other works in the text of the article. Each of the entries in the bibliography is created using the <BIB_ENTRY> tag. You can specify a heading for the bibliography either as an argument to the <BIBLIOGRAPHY> tag or by using an unnumbered heading tag immediately before the <BIBLIOGRAPHY> tag.

If you wish to create numbered reference notes, you should use the <REF_NOTE> and <REF_NOTES> tags.

EXAMPLE

The following example shows how to format a bibliography using the <BIBLIOGRAPHY> tag.

```
<BIBLIOGRAPHY>(Bibliography)
<BIB_ENTRY>( <EMPHASIS>(Molecular Connectivity in Chemistry and Drug Research.)
Lamont B. Kier and Lowell H. Hall. Academic Press, 1983.)
<BIB_ENTRY>(Arhnheim, Rudolph, <EMPHASIS>(Visual Thinking).
University of California Press, Berkeley, 1984.)
<ENDBIBLIOGRAPHY>
```

ARTICLE Doctype Tag Reference

<BIB_ENTRY>

<BIB_ENTRY>

Specifies a single entry in a bibliography.

FORMAT <BIB_ENTRY> (*bibliography-text*)

ARGUMENTS *bibliography-text*
Specifies the text of the bibliographic entry.

related tags • <BIBLIOGRAPHY>

restrictions Valid only in the context of the <BIBLIOGRAPHY> tag.

required terminator *None.*

DESCRIPTION You can use the <BIB_ENTRY> tag to create a single entry for a bibliography. The text of the bibliography entry is passed as an argument to the <BIB_ENTRY> tag. This text can be of any length.

EXAMPLE The following example shows how to use a <BIB_ENTRY> tag to create a bibliographic entry within the context of the <BIBLIOGRAPHY> tag.

```
<BIBLIOGRAPHY>(Bibliography)
<BIB_ENTRY>(<EMPHASIS>(Molecular Connectivity in Chemistry and Drug Research.)
Lamont B. Kier and Lowell H. Hall. Academic Press, 1983.)
<BIB_ENTRY>(Arhnheim, Rudolph, <EMPHASIS>(Visual Thinking).
University of California Press, Berkeley, 1984.)
<ENDBIBLIOGRAPHY>
```

<COLUMN>

Specifies that output should begin in a new column in a two-column format.

FORMAT <COLUMN>

ARGUMENTS *None.*

related tags • The global <FINAL_CLEANUP> tag

restrictions Valid only in a two-column doctype such as REPORT.TWOCOL or ARTICLE.

required terminator *None.*

DESCRIPTION Use the <COLUMN> tag to cause the text immediately following it to be started in a new column. If this tag occurs in the left text column, the text immediately following it begins in the right text column. If this tag occurs in the right text column, the text immediately following it begins in the left column of the next page.

Use the <COLUMN> tag when you always want to begin a new column at that point in your text. You can use the COLUMN_BREAK argument to the global <FINAL_CLEANUP> tag to also specify a column break; however, this should be used only during the final processing of the two-column document.

See Section 2.2 for more information on improving the formatting of a two-column doctype such as ARTICLE.

EXAMPLE The following example shows how to use the <COLUMN> tag to begin a new text column. In this example, the writer wants the two descriptions to appear side by side, one in each column.

ARTICLE Doctype Tag Reference

<COLUMN>

<SUBHEAD1>(Woodwind Instruments)

<P>Woodwind instruments have the following attributes:

<LIST>(UNNUMBERED)

<LE>They are often made of wood, hence their name.

<LE>Musicians create sound using these instruments by causing a reed to vibrate.

.

.

<ENDLIST>

<COLUMN>

<SUBHEAD1>(Brass Instruments)

<P>Brass instruments have the following attributes:

<LIST>(UNNUMBERED)

<LE>They are often made of brass, hence their name.

<LE>Musicians create sound using these instruments by vibrating (buzzing) their lips into a steel mouthpiece.

.

.

<ENDLIST>

<DOCUMENT_ATTRIBUTES>

Enables doctype-specific tags that override the default design format of the ARTICLE doctype.

FORMAT <DOCUMENT_ATTRIBUTES>

ARGUMENTS *None.*

related tags *None.*

restrictions *None.*

required terminator <ENDDOCUMENT_ATTRIBUTES>

DESCRIPTION The <DOCUMENT_ATTRIBUTES> tag can be used in three doctypes:

- ARTICLE
- REPORT
- SOFTWARE

The <DOCUMENT_ATTRIBUTES> tag enables a group of tags in each of these doctypes that allow you to modify the default format of that doctype. These tags are recognized only within the context of the <DOCUMENT_ATTRIBUTES> tag. If other VAX DOCUMENT tags occur in this context, they are ignored, as if they had occurred within the context of a <COMMENT> tag.

Typically, you use the <DOCUMENT_ATTRIBUTES> tag at the beginning of an input file (or in a file specified using the DOCUMENT /INCLUDE qualifier) to alter the default format of a doctype for the processing of that entire file.

Table 9-1 summarizes the formatting tags enabled by the <DOCUMENT_ATTRIBUTES> tag in each of the three supported doctypes.

ARTICLE Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

Table 9–1 Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> Tag

Formatting Tags	Description
Tags Enabled in the ARTICLE Doctype	
<SET_HEADINGS> (UNNUMBERED) <SET_HEADINGS> (NUMBERED)	<p>The <SET_HEADINGS> tag specifies whether numbered or unnumbered headings are produced by the heading-level tags (<HEAD1> , <HEAD2> , and so on). By default, headings are not numbered in a document processed using the ARTICLE doctype.</p> <p>Use the <SET_HEADINGS> (NUMBERED) tag to specify that your headings should be numbered.</p>
Tags Enabled in the REPORT Doctype	
<SET_PAGE_NUMBERING> (BY_CHAPTER) <SET_PAGE_NUMBERING> (SEQUENTIAL)	<p>The <SET_PAGE_NUMBERING> tag specifies how pages are to be numbered in a document processed using the REPORT doctype. Pages are numbered sequentially by default.</p> <p>If the BY_CHAPTER argument is used, the pages are numbered by chapter. For example, the second page in Chapter 3 would be numbered 3-2. If the BY_CHAPTER argument is used in a document that contains no chapters, the page numbering is sequential.</p> <p>The SEQUENTIAL argument specifies that the pages should be numbered sequentially, as they are by default.</p>
<SET_FORMAL_ELEMENT_NUMBERING> (BY_CHAPTER) <SET_FORMAL_ELEMENT_NUMBERING> (SEQUENTIAL)	<p>The <SET_FORMAL_ELEMENT_NUMBERING> tag specifies how formal tables, figures, and examples are to be numbered in a document processed using the REPORT doctype. By default, formal tables, figures, and examples are numbered sequentially.</p> <p>The BY_CHAPTER argument indicates that formal elements are to be numbered using the chapter prefix; for example, the first formal table in Chapter 4 would be numbered as Table 4-1.</p> <p>The SEQUENTIAL argument indicates that formal elements are to be numbered sequentially throughout the document; for example, the eighth formal table in the document, regardless of what chapter it occurs in, would be numbered as Table 8.</p> <p>If the BY_CHAPTER argument is used in a document that contains no chapters, VAX DOCUMENT uses the default sequential formal element numbering.</p>

ARTICLE Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

Table 9–1 (Cont.) Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> Tag

Formatting Tags	Description
Tags Enabled in the SOFTWARE Doctype	
<SET_RUNNING_TITLES> (BY_HEADONE)	The <SET_RUNNING_TITLES> tag specifies that the running title at the top of each page is composed of two lines: the first line is the title of the chapter, and the second line is the heading text of the current <HEAD1> tag. Note that the argument BY_HEADONE is required. By default, the chapter title is used as a single-line running title.

EXAMPLE

In the following example of a file to be processed under the ARTICLE doctype, numbered headings are specified using the <SET_HEADINGS> tag with the NUMBERED argument. This will cause any subsequent heading tags (<HEAD1>, <HEAD2> and so on) to be numbered.

```
<DOCUMENT_ATTRIBUTES>  
<SET_HEADINGS>(NUMBERED)  
<ENDDOCUMENT_ATTRIBUTES>
```

ARTICLE Doctype Tag Reference

<QUOTATION>

<QUOTATION>

Begins a quotation in which the spacing is retained and the text is not filled or justified.

FORMAT <QUOTATION>

ARGUMENTS *None.*

related tags

- The global <CODE_EXAMPLE> tag
- The global <SAMPLE_TEXT> tag

restrictions *None.*

required terminator <ENDQUOTATION>

DESCRIPTION Use the <QUOTATION> tag to format a quotation in which you want the spacing to be retained and the text to not be filled or justified.

The <QUOTATION> tag differs from the global <CODE_EXAMPLE> tag in that the text of the quotation is not set in a monospaced font. It allows you to quote poetry or passages of text that you do not want formatted.

For long, block-style quotations, you should use the global <SAMPLE_TEXT> tag.

EXAMPLE The following example shows how you can use the <QUOTATION> tag to set off a quotation in text and have the spacing retained.

```
It is often instructive to remember the words of our founder:
<QUOTATION>
  It is better to try again
  than to fail;
  And better still to succeed.
<ENDQUOTATION>
```

<REF_NOTE>

Specifies the text of a reference note, and creates a bracketed reference number in the article text.

FORMAT <REF_NOTE> (*text-of-note*[\ *symbol-name*])

ARGUMENTS *text-of-note*

Specifies the text of the reference note.

symbol-name

Specifies the symbolic name of the note. Specify this argument when you need to refer to the same note more than once in the same article. Subsequent referrals to the same note are made by referring to this symbol name with the global <REFERENCE> tag.

The *symbol-name* argument can be any text string of 31 characters or less without spaces. This text must contain only the letters A through Z, the numbers 0 through 9 and the underscore (_) character. The *symbol-name* may not begin with an underscore character.

related tags

- <BACK_NOTE>
- <BIBLIOGRAPHY>
- <REF_NOTES>
- The global <FOOTNOTE> tag
- The global <REFERENCE> tag

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <REF_NOTE> tag to create a reference note. Place the <REF_NOTE> tag in the SDML file at the point at which you are referencing text for which you want to provide the note. This tag is replaced in the output file by a number in brackets that corresponds to the number assigned to the note text; for example, [4].

Use the <REF_NOTES> tag to cause the text of this note and any other reference notes you have created using the <REF_NOTE> tag to be output. The <REF_NOTES> tag is typically placed at the end of the SDML file.

To reference a source that you have already referenced using the <REF_NOTE> tag, you need to specify the *symbol-name* argument to that <REF_NOTE> tag and use the global <REFERENCE> tag to refer to that symbol.

ARTICLE Doctype Tag Reference

<REF_NOTE>

Use the <BIBLIOGRAPHY> tag to create a bibliography that is not tied to references in the text of an article.

EXAMPLE

The following example shows how to use the <REF_NOTE> tag both for a single reference and for two references to the same source. Note how the second reference to the Hopkins and Johnson article is made using the <REFERENCE> tag.

These notes and any others would be output by the <REF_NOTES> tag that occurs at the end of the article.

Hopkins and Johnson (1968)

<REF_NOTE>(A.A Hopkins and B.B. Johnson,<QUOTE>(An Eye for an Eye,) Proceedings of the American Journal of Comparative Biology, 163:1145-1152.\EYE_ARTICLE) noted the preponderance of short cones in type A subjects.

The research of J.Dobbs (1972)

<REF_NOTE>(J. Dobbs,<quote>(Cones in Type A and B Subjects), Proceedings of the American Journal of Comparative Biology, 167:201-227.) corroborated this observation.

<P>

In 1978 the research of Hopkins and Johnson (1968) <REFERENCE>(EYE_ARTICLE), was shown to have been misleading.

<REF_NOTES>

<REF_NOTES>

Causes all accumulated reference notes to be output.

FORMAT <REF_NOTES> [(*heading-text*)]

ARGUMENTS *heading-text*

Specifies a heading for the reference notes. This heading has the default heading format used by <HEAD1> tags in an article with unnumbered heads.

If you do not specify the *heading-text* argument, no heading is output. You can specify your own heading, with the heading tag (<HEAD1> , <HEAD2> and so on) that is appropriate to your document.

related tags

- <REF_NOTE>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <REF_NOTES> tag to output a list of all the references you created using the <REF_NOTE> tag. These references are numbered and correspond to the number placed in your document by the <REF_NOTE> tag (for example, [4]). Typically, the <REF_NOTES> tag is placed at the end of the SDML file so that the accumulated references appear at the end of the article.

Use the <BIBLIOGRAPHY> tag to create a bibliography that is not tied to references in the text of an article.

EXAMPLES

The following example shows how you can create a list of references with the heading "References."

1 <REF_NOTES>(References)

The following example shows how you can use a heading tag as an alternative heading for the list of references. In this example, the <HEAD2> tag was used.

2 <HEAD2>(References)
 <REF_NOTES>

ARTICLE Doctype Tag Reference

<RUNNING_FEET>

<RUNNING_FEET>

Creates a single line heading at the bottom of each page.

FORMAT <RUNNING_FEET> (*title-text*)

ARGUMENTS *title-text*
Specifies the text to be used as a running heading at the foot of the page.

related tags • <RUNNING_TITLE>

restrictions *None.*

required terminator *None.*

DESCRIPTION Use the <RUNNING_FEET> tag to place a heading at the bottom of every page. This heading is called a "footer" because it appears at the foot of the page. When the same footer is used for several pages, the footers are collectively called "running feet."

This tag accepts one argument that is the text heading that appears at the bottom of the page. This text is output exactly as entered, including spacing and capitalization.

Use the <RUNNING_TITLE> tag to create a heading at the top of the page. See the reference description of the <RUNNING_TITLE> tag for more information on that tag.

EXAMPLE The following example shows how to use the <RUNNING_FEET> tag to place the heading "Getting the Piece of Paper" at the bottom of each page. The running footer will be output exactly as entered.

```
<RUNNING_FEET>(Getting the Piece of Paper)
<HEAD2>(Getting the Piece of Paper)
<P>
```

You can buy clean paper in most major supermarkets, department stores, and hardware stores. You should try to get ruled paper so that your letter will be neat and easy to read.

<RUNNING_TITLE>

Creates a one- or two-line running heading at the top of each page.

FORMAT

<RUNNING_TITLE> ({ *OFF*
title-1 [\ *title-2*]
[\ *FIRST_PAGE*] })

ARGUMENTS

title-1

Specifies the text of a running title. If a two-line title is specified, this title is output on the upper title line.

title-2

Specifies the bottom line of a running title that has two lines; this argument is optional.

FIRST_PAGE

Specifies that the running title is to begin output on the first output page. If this keyword is not specified, the running title is output on the page after the current page.

OFF

Specifies that any existing running titles created using the <RUNNING_TITLE> tag should be disabled for the page on which this tag occurs and on any subsequent pages.

related tags

- <RUNNING_FEET>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <RUNNING_TITLE> tag to specify a one- or two-line title at the top of the page. Use the *FIRST_PAGE* argument to the <RUNNING_TITLE> tag to begin the title lines on the first page of output, rather than on the page after the current page as is the default.

Use the *OFF* argument to disable any existing running titles created using the <RUNNING_TITLE> tag. These titles will then be disabled for the page on which this tag occurs and on any subsequent pages.

Use the <RUNNING_FEET> tag to create a heading that appears at the bottom of the page. See the reference description of the <RUNNING_FEET> tag for more information on that tag.

ARTICLE Doctype Tag Reference

<RUNNING_TITLE>

EXAMPLES

The following examples show how to use the <RUNNING_TITLE> tag to create titles and how to disable them.

The following example shows how to use the <RUNNING_TITLE> tag to create the two line running title "An E. B. Bartz Course:" and "Writing Quality Correspondence." Note that because the FIRST_PAGE argument is used, the two-line running title will appear at the top of the first page.

1 <RUNNING_TITLE>(An E. B. Bartz Course:\Writing Quality Correspondence\FIRST_PAGE)
<HEAD>(How to Write a Letter)
<P>
The first thing that you should do in writing a letter is to get
a clean piece of paper and a well-sharpened pencil.

The following example shows how you can disable a running title by using the OFF argument to the <RUNNING_TITLE> tag.

2 <COMMENT>(turn off running titles for the following example page)
<RUNNING_TITLE>(OFF)
<HEAD>(An Example of a Letter)
.
.
.

<SOURCE_NOTE>

Provides information pertaining to the original source of information for an article.

FORMAT <SOURCE_NOTE> (*source-text*)

ARGUMENTS *source-text*

Specifies the text that describes the source of the article. The text can be any length and can include any tags not listed in the restrictions section. The source text is positioned at the bottom of the column of output in which the tag is specified.

related tags

- <AUTHOR>
- <VITA>

restrictions

The following tags should not be used as part of the source note text:
<CODE_EXAMPLE>, <EXAMPLE>, <FIGURE>, <FORMAT>, <HEAD1>
through <HEAD6>, <INTERACTIVE>, <MATH>, or <NOTE>.

required terminator

None.

DESCRIPTION

Typically, information provided in the <SOURCE_NOTE> tag is placed either at the beginning of the first column on the first page of an article or at the end of the last column on the last page. You should place the <SOURCE_NOTE> tag in your SDML file to correspond to where you want the output to appear.

If you want the text to appear on the first page, specify the tag following the <AUTHOR> tag. If you want the text to appear on the last page, specify the tag at the end of the SDML file.

EXAMPLE

The following example shows how to create a note describing the original source of an article. Note that the global <LINE> tag is used to produce a line beginning with the copyright symbol to indicate the copyright date. The copyright symbol is created by using the global <MCS> tag with the argument COPYRIGHT.

```
<SOURCE_NOTE>(Reprinted from <EMPHASIS>(Visible Discs,) Volume V,  
Number 3, Summer 1971. c/o The Top-Ten Museum of Art, Cleveland,  
Ohio, U.S.A., 44106  
<LINE>  
<MCS>(COPYRIGHT) 1971 by <EMPHASIS>(Visible Discs.))
```

ARTICLE Doctype Tag Reference

<SUBTITLE>

<SUBTITLE>

Specifies a subtitle for an article.

FORMAT <SUBTITLE> (*title-line-1*[\ *title-line-2*[\ *title-line-3*]])

ARGUMENTS *title-line-n*
Specifies up to three lines of text for subtitle of an article. The text of each argument is centered on a new line of output.

related tags • <TITLE>
 • <TITLE_SECTION>

restrictions *None.*

required terminator *None.*

DESCRIPTION Use the <SUBTITLE> tag to create a subordinate title for an article that has up to three separate lines. Each of the subtitle lines is centered in the column in which the tag occurs (typically the first column). Use the <TITLE> tag to create a main title for an article.

If you want the subtitle (or title) to span both columns of the article, you should use the <TITLE_SECTION> tag in your SDML file before the <SUBTITLE> (or <TITLE>) tag.

EXAMPLE The following example illustrates an article that begins with a main title followed by a subtitle that will span both columns. Note that the <AUTHOR> tag occurs outside of the context of the <TITLE_SECTION> tag so it will not span both columns in the output.

```
<TITLE_SECTION>
<TITLE>(FILE PROCESSING)
<SUBTITLE>(CONCEPTS AND INSTRUCTIONS)
<ENDTITLE_SECTION>
<AUTHOR>(A.B. Roma\Contributing Editor)
```

<TITLE>

Specifies the main title line for an article.

FORMAT <TITLE> (*title-line-1*[\ *title-line-2*[\ *title-line-3*]])

ARGUMENTS *title-line-n*
Specifies up to three lines of text for the title of the article. The text of each argument is centered on a new line of output.

related tags • <SUBTITLE>
 • <TITLE_SECTION>

restrictions *None.*

required terminator *None.*

DESCRIPTION Use the <TITLE> tag to create a main title for an article that has up to three separate lines. Each of the title lines is centered in the column in which the tag occurs (typically the first column). Use the <SUBTITLE> tag to create a subordinate title for an article.

If you want the title (or subtitle) to span both columns of the article, use the <TITLE_SECTION> tag in your SDML file before the <TITLE> (or <SUBTITLE>) tag.

EXAMPLE The following example illustrates an article that begins with a main title followed by a subtitle that spans both columns. Note that the <AUTHOR> tag occurs outside of the context of the <TITLE_SECTION> tag so it will not span both columns in the output.

```
<TITLE_SECTION>
<TITLE>(FILE PROCESSING\USING THE CALL INTERFACE)
<SUBTITLE>(CONCEPTS AND INSTRUCTIONS)
<ENDTITLE_SECTION>
<AUTHOR>(A.B. Roma\Contributing Editor)
```

ARTICLE Doctype Tag Reference

<TITLE_SECTION>

<TITLE_SECTION>

Begins the title section of an article that spans both columns of an article.

FORMAT <TITLE_SECTION>

ARGUMENTS *None.*

related tags • <SUBTITLE>
 • <TITLE>

restrictions *None.*

required terminator <ENDTITLE_SECTION>

DESCRIPTION Use the <TITLE_SECTION> tag to create a section at the beginning of an article in which you want a title, subtitle, or other information to span the full page. If the <TITLE> or <SUBTITLE> tag is used in the context of the <TITLE_SECTION> tag, the typeface that is output by those tags will be larger than the typeface that is output outside of the context of the <TITLE_SECTION> tag.

If you do not use the <TITLE_SECTION> tag, any titles, subtitles, or additional information is output in the appropriate column and does not span the full page.

EXAMPLES In the following example, the <TITLE> and <SUBTITLE> tags are specified within the context of the <TITLE_SECTION> tag. The <AUTHOR> tag appears after the <ENDTITLE_SECTION> , so it will be formatted in the first column of the article.

1 <TITLE_SECTION>
 <TITLE>(Optical Discs)
 <SUBTITLE>(The New Documentation Frontier)
 <ENDTITLE_SECTION>
 <AUTHOR>(A. B. Roma)

In the following example, the <TITLE> and <SUBTITLE> tags are used without the <TITLE_SECTION> tag so the title and subtitle text will be set in the first column with the author information.

2 <TITLE>(Optical Discs)
 <SUBTITLE>(The New Documentation Frontier)
 <AUTHOR>(A. B. Roma)

<VITA>

Provides information about the author's professional history.

FORMAT <VITA> (*vita-text*)

ARGUMENTS *vita-text*

Specifies information describing the professional history of the author. The text can be any length and can include any tags that are not listed in the restrictions section. The vita text is positioned at the bottom of the column of output in which the tag is specified.

related tags

- <AUTHOR>
 - <SOURCE_NOTE>
-

restrictions

The following tags should not be used as part of the vita text: <CODE_EXAMPLE>, <EXAMPLE>, <FIGURE>, <FORMAT>, <HEAD1> through <HEAD6>, <INTERACTIVE>, <MATH>, or <NOTE>.

required terminator

None.

DESCRIPTION

Use the <VITA> tag to describe the professional history of an article's author. Typically, information provided in the <VITA> tag is placed either at the beginning of the first column of the first page of an article, or at the end of the last column on the last page. You should place the <VITA> tag in your SDML file to correspond to where you want the output to appear.

If you want the text to appear on the first page, specify the <VITA> tag following the <AUTHOR> tag. If you want the text to appear on the last page, specify the <VITA> tag at the end of the SDML file.

The following list includes the other tags that allow you to specify information about the author.

<AUTHOR>	Specifies an author for an article.
<AUTHOR_ADDR>	Specifies an address for the author.
<AUTHOR_AFF>	Specifies the organizational affiliation of the author.
<AUTHOR_LIST>	Lists multiple authors of an article.

See the appropriate tag description in this chapter for more information on any of these tags.

ARTICLE Doctype Tag Reference

<VITA>

EXAMPLE

The following example illustrates how to create a description of the author to accompany the <AUTHOR> tag. This coding positions the text at the bottom of the first column of the first page of the article.

<AUTHOR>(A.B. Roma)

<VITA>(A.B. Roma is program director for
information processing and distribution for the Top-Ten Corporation.
She has published numerous magazine articles.)

4 Using the MANUAL Doctype

The MANUAL doctype lets you create users' manuals in one of three designs:

- **MANUAL.GUIDE**
Creates a users' manual in a 7 × 9 inch format with numbered headings. This design is intended for chapter-oriented tutorial material.
- **MANUAL.PRIMER**
Creates a users' manual in a 7 × 9 inch format with unnumbered headings. This design is intended for chapter-oriented primer material.
- **MANUAL.REFERENCE**
Creates a users' manual in an 8½ × 11 inch format with numbered headings. This design is intended for reference material.

The MANUAL doctype provides no doctype-specific tags, but accepts the full range of VAX DOCUMENT global tags. See the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags.

You process a file with the MANUAL doctype by using one of the doctype keywords in the preceding list on the DOCUMENT command line. The following example shows how to process a file named MYMANUAL.SDML with the MANUAL doctype to create a reference manual.

```
$ DOCUMENT mymanual MANUAL.REFERENCE LN03
```

See *VAX DOCUMENT Design Samples* for more information on the format of the MANUAL designs.

4.1 A Sample of the MANUAL Doctype

This section contains an example of the first few pages of a hardware manual created using the MANUAL.REFERENCE doctype. This manual sample includes a title page and some numbered headings and a table in the body of the manual. You may find this sample useful in understanding how global tags can be used in the MANUAL doctype to create various kinds of manuals.

The SDML code for the manual is shown first, followed by the output from that SDML code.

```
<FRONT_MATTER>  
<TITLE_PAGE>  
<TITLE>(Series III Overthruster\User Manual)  
<ABSTRACT>  
This book describes the Series III Overthruster, the Series III  
manual-override mode and the Overthruster monitor utilities.  
<ENDABSTRACT>  
<ENDTITLE_PAGE>  
<ENDFRONT_MATTER>
```

Using the MANUAL Doctype

<CHAPTER>(Introduction to the Overthruster\INTRO_CHAP)
<P>

The Series III Overthruster is an intelligent mass thrust device, designed to deliver controlled thrust from a ZX-300 type drive unit. The Series III family provides enhanced data collection and control utilities far superior to those present in the Series II.

<HEAD1>(Series III Overthruster Models)

<p>

The Series III Overthruster family has three models.

<REFERENCE>(MODEL_TAB) lists these models and the salient features of each.

<TABLE>(Comparison of Series III Overthruster Models\MODEL_TAB)

<TABLE_SETUP>(4\25\10\10)

<TABLE_HEADS>(Feature\(3\LEFT)Models\ \)

<TABLE_HEADS>(\SIII-030\SIII-050\SIII-070)

<TABLE_ROW>(Data Channels Supported\2\4\4)

<TABLE_ROW>(I/O Control Processor \A-L35\J19\J19)

<TABLE_ROW>(Drive Unit\ZX-301\ZX-301\ZX-310A)

<TABLE_ROW>(Power Source\TY-100\TY-100A \TY-200)

<TABLE_ROW>(Control Memory\128Kb\128Kb\512Kb)

<TABLE_ROW>(Data Memory\ -- \ -- \512Kb)

<ENDTABLE>

<HEAD1>(Series III Overthruster Functional Description)

<p>

The Series III Overthruster is an intelligent mass thrust device in accordance with the DS8 architecture.

This example may produce the following output if processed using the MANUAL doctype keyword.

Sample Output of Manual Doctype

Series III Overthruster User Manual

This book describes the Series III Overthruster, the Series III manual-override mode and the Overthruster monitor utilities.

Using the MANUAL Doctype

Sample Output of Manual Doctype

Chapter 1

Introduction to the Overthruster

The Series III Overthruster is an intelligent mass thrust device, designed to deliver controlled thrust from a ZX-300 type drive unit. The Series III family provides enhanced data collection and control utilities far superior to those present in the Series II.

1.1 Series III Overthruster Models

The Series III Overthruster family has three models. Table 1-1 lists these models and the salient features of each.

Table 1-1: Comparison of Series III Overthruster Models

Feature	Models		
	SIII-O30	SIII-O50	SIII-O70
Data Channels Supported	2	4	4
I/O Control Processor	A-L35	J19	J19
Drive Unit	ZX-301	ZX-301	ZX-310A
Power Source	TY-100	TY-100A	TY-200
Control Memory	128Kb	128Kb	512Kb
Data Memory	—	—	512Kb

1.2 Series III Overthruster Functional Description

The Series III Overthruster is an intelligent mass thrust device in accordance with the DS8 architecture.

5

Using the MILSPEC Doctype

The MILSPEC doctype lets you create standard military specifications in accordance with the United States Department of Defense Military Specification Standard MIL-STD-490A, published June 4, 1985.

The MILSPEC doctype has a single design. You process files under this doctype by using the MILSPEC doctype keyword on the DOCUMENT command line.

This doctype accepts the full range of VAX DOCUMENT global tags, with the exception of the <PART> and <PART_PAGE> tags (see the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags).

It also provides doctype-specific tags that let you create a title page in accordance with the MIL-STD-490A standard, and a tag that allows you to override the default appendix numbering used by VAX DOCUMENT. Chapter 11 provides the reference information on these tags.

Table 5-1 briefly describes the doctype-specific tags available in the MILSPEC doctype.

Table 5-1 Tags Available in the MILSPEC Doctype

Tag Name	Description
<SET_APPENDIX_NUMBER>	Overrides the default appendix Roman number assigned to an appendix by VAX DOCUMENT.
<SIGNATURE_LIST>	Begins a two-column listing of signature lines on the title page and places a heading above each list. Each row of signature lines is created using the <SIGNATURE_LINE> tag within the context of the <SIGNATURE_LIST> tag.
<SIGNATURE_LINE>	Creates up to two rules on a line (one in each signature column) and places a name below each rule; each rule is used as a signatory line for the person listed below it.
<SPECIFICATION_INFO>	Creates a listing of information about the specification document on the title page and creates a two-line running heading listing the specification number and date for the rest of the document.
<SPEC_TITLE>	Creates a title with up to seven centered lines on the title page.
<SUBTITLE>	Creates a subtitle with up to seven centered lines on the title page.

VAX DOCUMENT provides a template SDML file for general usage of the MIL-STD-490A format. It also provides 24 template SDML files for documents that conform to the Department of Defense standard DOD-STD-2167 for Data Item Descriptions (DID) published June 4, 1985. Data item descriptions are MIL-STD-490A documents that are specialized for a particular kind of information. The exact form of each data item description is specified by DOD-STD-2167.

You can use these files to create military specifications. If you have VAX Language-Sensitive Editor Version 2.0 (LSE) installed on your system, you can use that editor to access the LSE templates for the data item description documents and expand the appropriate placeholders. See the

Using the MILSPEC Doctype

VAX DOCUMENT User Manual, Volume 1 for more information on using LSE with VAX DOCUMENT.

Each template file contains all the headings, titles, and so on required by the DOD-STD-2167 specification for that document. The template input files contain comments that guide you in placing your information into the file. Table 5-2 lists the data item description templates files that VAX DOCUMENT provides.

Documents produced using the MILSPEC doctype conform to the MIL-STD-490A specification and have the following general attributes:

- Pages, formal figures, and formal tables are numbered sequentially throughout the document and are not numbered by chapter, section, or appendix.
- Paragraphs are numbered using the global numbered heading tags (<HEAD1>, <HEAD2>, and so on) to create the Arabic numerals. A period (.) is automatically placed at the end of the numbered paragraph headings.
- Formal tables are numbered using Roman numerals.
- The global <PREFACE> tag automatically generates a preface section heading of "Foreword" rather than "Preface." All other VAX DOCUMENT doctypes use the heading "Preface."
- The table of contents begins on page ii rather than on page iii. All other VAX DOCUMENT doctypes that support the automatic creation of tables of contents begin the table of contents on page iii.
- Appendixes are numbered using Roman numerals. Sections and paragraphs within an appendix are numbered in Arabic numbers that correspond to the appendix number multiplied by 10. For example, in Appendix II, the first paragraph would be paragraph 20, and the second paragraph would be 20.1.

See *VAX DOCUMENT Design Samples* for additional information on the format of the MILSPEC design.

5.1 Writing a Document According to MIL-STD-490A

You can create a document in accordance with MIL-STD-490A either by copying the template file MILSPEC_SAMPLE.SDML from the directory DOC\$TEMPLATES into your work directory and then editing it to insert your text, or by creating your own file. There are comments in the template file to guide you in its use. If you create your own new file, you may still want to look at the template SDML input file as a guide.

You can use the MILSPEC doctype to create documents that conform to MIL-STD-490A by tagging them in the following manner:

- Create the required cover page for the document using the <SPEC_TITLE> and <SPECIFICATION_INFO> tags within the context of the global <TITLE_PAGE> tag. The <SPEC_TITLE> tag creates a title for the document and the <SPECIFICATION_INFO> tag places additional information on the cover page of the document, as well as setting the running headings for the rest of the document (including the table of contents).

- Create major sections (such as Section 1, "Scope,") in your document using the global <CHAPTER> tag. Each <CHAPTER> tag that has paragraph text immediately following it should have a <P> tag placed between the heading and the text for correct formatting. Chapters that contain no pertinent information should be coded with the <CHAPTER> tag, complete with the appropriate title text and the symbol name argument. Such chapters should contain only the following standard disclaimer paragraph as specified in MIL-STD-490A.

This section is not applicable to this specification.

Because the disclaimer text does not vary from use to use, you may want to symbolize it using the global <DEFINE_SYMBOL> tag as follows:

```
<DEFINE_SYMBOL>(This section is not applicable to this specification.\na_text)
```

You could then use the global <REFERENCE> tag to include this text as follows:

```
<comment>(This section must be included for DID 80031 documents,  
but requires no specification details.)
```

```
<CHAPTER>(Qualification Requirements\qualifications_sec)
```

```
<P>
```

```
<REFERENCE>(NA_TEXT)
```

- Create the numbered paragraphs within each major section using the appropriate global numbered heading tags (<HEAD1>, <HEAD2>, and so on). Each heading tag that has paragraph text immediately following it should have a <P> tag placed between the heading and the text for correct formatting. The MILSPEC doctype automatically inserts a period at the end of all titled numbered headings.

If you need to cross reference numbered paragraphs within your document, you should specify them using the <REFERENCE> tag with the VALUE argument so that the default text "Section" is not automatically output before the paragraph number.

5.2 Writing Data Item Description Documents

VAX DOCUMENT provides 24 template files for data item description (DID) documents and a general military specification template file, which can be customized for other documents. The framework for each of the supported DID specifications is supplied in the template files so that all you need supply is the text for your particular document. All required tags, including section and paragraph headings, are provided in the document template.

Each template file is a collection of individual element files coded for a specific DID or military document. These files are concatenated into a single file to simplify usage and storage. When you use one of these files, separate it into several files, one file for each major section.

Each section of the template input file is labeled with comments that contain directions for its use. Placing each major section in a separate file makes it easier to maintain your document, and lets you use the book building features of VAX DOCUMENT. You then list these files in a profile file and process the profile list through a book build.

Using the MILSPEC Doctype

For example, if you were to use the file MILSPEC_DID_80025.SDML, you would place each of the six major sections in a separate file, and also place the front matter section and the symbol definition section in separate files. Each of these files would be placed in your document when you process book build profile file.

The following is a list of steps you should follow to use the MILSPEC template files.

- 1 Select the template you want by referring to Table 5-2. If the template you want is not listed, use the file MILSPEC_SAMPLE.SDML as the basis for creating your own template. You may want to read through the template to make sure it is the one you want.
- 2 Copy the template file into your working directory. Do not modify the VAX DOCUMENT templates in DOC\$TEMPLATES. Other users of VAX DOCUMENT may also want to use them.
- 3 Separate the templated file into separate files, placing each major section (begun using the <CHAPTER> tag) in a separate file. The front matter (begun with the <FRONT_MATTER> tag) and the symbol definitions (the <DEFINE_SYMBOL> and <DEFINE_BOOK_NAME> tags) should also be placed in separate files.

References to symbols created using the <DEFINE_SYMBOL> tag occur in each template file. Comments in the template file identify which symbols are used by all the templates and which symbols are used only in that particular template.

You will need to modify the text arguments to the <DEFINE_SYMBOL> tags so that the proper text (such as your product's name) is automatically inserted into the template.

- 4 Create a book building profile that lists each of the major sections and the front matter file as book elements. These book elements can then be built into a single book by VAX DOCUMENT. You may want to include the <CONTENTS_FILE> and <INDEX_FILE> tags in your profile file to automatically include your table of contents and index files into the final book.

The profile should not include the file that contains the symbol definitions in it. These symbol definitions should be included into your document by specifying the symbol definitions file as an argument to the DOCUMENT /SYMBOLS qualifier when you process your profile.

The following is a sample book building profile for a military specification document. See how the <CONTENTS_FILE> and <INDEX_FILE> tags are used to automatically include the table of contents and index files for the document.

```
<PROFILE>
<ELEMENT>(frontmatter.sdml)
<CONTENTS_FILE>
<ELEMENT>(scopechap.sdml)
<ELEMENT>(secondchap.sdml)
<ELEMENT>(thirdchap.sdml)
<ELEMENT>(fourthchap.sdml)
<ELEMENT>(fifthchap.sdml)
<ELEMENT>(noteschap.sdml)
<ELEMENT>(firstapx.sdml)
<INDEX_FILE>
<ENDPROFILE>
```

If you have a large book element, or a book element that contains sections that change a great deal, you may want to separate the book element into several files called book subelements that are included into the book element file using the global <INCLUDE> tag. The following sample shows the book element file THIRDCCHAP.SDML that includes several book subelement files.

```
<CHAPTER>(Testing Results\test_results_chap)
<INCLUDE>(testing_intro.sdml)
<INCLUDE>(test1_results.sdml)
<INCLUDE>(test2_results.sdml)
<INCLUDE>(test3_results.sdml)
<INCLUDE>(test4_results.sdml)
<INCLUDE>(test5_results.sdml)
<INCLUDE>(testing_conclusions.sdml)
```

Each of these book subelements can be processed individually and have all cross references correctly resolved after a bookbuild of the entire book has taken place (the book build process creates the XREF cross reference file that the subelement accesses to resolve the cross references). See the *VAX DOCUMENT User Manual, Volume 1* for more information about creating and using book build profiles, symbol definitions files, and processing book subelements.

- 5 Enter the appropriate information for your document into each of the major sections of the template input file.

Table 5-2 lists the data item descriptions for which VAX DOCUMENT provides templated SDML files. The names of the template files correspond to their data item description document number.

Table 5-2 MILSPEC Doctype Data Item Description Templates

Data Item Description Number	Template Description	Template Files Located in the Directory DOC\$TEMPLATES
None.	Sample template for any document conforming to MIL-STD-490A	MILSPEC_SAMPLE.SDML
DI-CMAN-80008 AMSC No. N3584	System/Segment Specification	MILSPEC_DID_80008.SDML
DI-MCCR-80009 AMSC No. N3585	Software Configuration Management Plan	MILSPEC_DID_80009.SDML
DI-MCCR-80010 AMSC No. N3586	Software Quality Evaluation Plan	MILSPEC_DID_80010.SDML
DI-MCCR-80011 AMSC No. N3587	Software Standards and Procedures Manual	MILSPEC_DID_80011.SDML
DI-MCCR-80012 AMSC No. N3588	Software Top Level Design Document	MILSPEC_DID_80012.SDML
DI-MCCR-80013 AMSC No. N3589	Version Description Document	MILSPEC_DID_80013.SDML
DI-MCCR-80014 AMSC No. N3590	Software Test Plan	MILSPEC_DID_80014.SDML
DI-MCCR-80015 AMSC No. N3591	Software Test Description	MILSPEC_DID_80015.SDML

Using the MILSPEC Doctype

Table 5–2 (Cont.) MILSPEC Doctype Data Item Description Templates

Data Item Description Number	Template Description	Template Files Located in the Directory DOC\$TEMPLATES
DI-MCCR-80016 AMSC No. N3592	Software Test Procedure	MILSPEC_DID_80016.SDML
DI-MCCR-80017 AMSC No. N3593	Software Test Report	MILSPEC_DID_80017.SDML
DI-MCCR-80018 AMSC No. N3594	Computer System Operator's Manual	MILSPEC_DID_80018.SDML
DI-MCCR-80019 AMSC No. N3595	Software User's Manual	MILSPEC_DID_80019.SDML
DI-MCCR-80020 AMSC No. N3596	Computer System Diagnostic Manual	MILSPEC_DID_80020.SDML
DI-MCCR-80021 AMSC No. N3597	Software Programmer's Manual	MILSPEC_DID_80021.SDML
DI-MCCR-80022 AMSC No. N3598	Firmware Support Manual	MILSPEC_DID_80022.SDML
DI-MCCR-80023 AMSC No. N3599	Operational Concept Document	MILSPEC_DID_80023.SDML
DI-MCCR-80024 AMSC No. N3600	Computer Resources Integrated Support Document	MILSPEC_DID_80024.SDML
DI-MCCR-80025 AMSC No. N3601	Software Requirements Specification	MILSPEC_DID_80025.SDML
DI-MCCR-80026 AMSC No. N3602	Interface Requirements Specification	MILSPEC_DID_80026.SDML
DI-MCCR-80027 AMSC No. N3603	Interface Design Document	MILSPEC_DID_80027.SDML
DI-MCCR-80028 AMSC No. N3604	Data Base Design Document	MILSPEC_DID_80028.SDML
DI-MCCR-80029 AMSC No. N3605	Software Product Specification	MILSPEC_DID_80029.SDML
DI-MCCR-80030 AMSC No. N3606	Software Development Plan	MILSPEC_DID_80030.SDML
DI-MCCR-80031 AMSC No. N3607	Software Detail Design Document	MILSPEC_DID_80031.SDML

5.3 A Sample Use of the MILSPEC Doctype Tags

This section contains a sample of the first few pages of a specification created using the MILSPEC doctype. This sample includes the title page of the specification and the first page of text after the title page. The text page illustrates numbered paragraphs and tables created using the MILSPEC doctype.

Using the MILSPEC Doctype

Note how the <SPECIFICATION_INFO> tag is used to create the running heads, which contain the date and specification number. Also note that all major paragraphs are numbered by using one of the numbered heading tags such as <HEAD1> and <HEAD2>. The SDML code for the specification is shown first, followed by the output from that SDML code.

```
<FRONT_MATTER>
<TITLE_PAGE>
<SPECIFICATION_INFO>(12345B\ai42-b4\<DATE>\Part I of Three Parts)
<SPEC_TITLE>(Preliminary Development Specification
\For the Overthrust Monitor System
\Series (Series Configuration Number)
\Order Number (Approved Order Number))
<SUBTITLE>(Submitted Under\Contract A00000--11--A--2222)
<SIGNATURE_LIST>(Authenticated by:\Approved by:)
<SIGNATURE_LINE>(Procurer\Program Manager)
<SIGNATURE_LINE>(Date\Technical Director)
<SIGNATURE_LINE>(\Consultant)
<ENDSIGNATURE_LIST>
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
<CHAPTER>(Scope\first_sec)
<HEAD1>(Scope\scope_head)
<P>
This document establishes all specifications for the design and
production of the Overthrust Monitor System (USN-122-233x) by our Corporation.
<HEAD1>(Purpose\purpose_sec)
<P>
The purpose of this document is to specify all design and production
dimensions of the Overthrust Monitor System. This will ensure that all
essential requirements are met and that all concerns are addressed.
<HEAD2>(Primary Purpose\Primary_purpose_sec)
<P>
The primary purpose is to enrich the quality dimension of our product.
<HEAD3>(Secondary Purpose\secondary_purpose_sec)
<P>
The secondary purpose is to create a corporate strategy for the product that
encompasses the goals established in <REFERENCE>(primary_purpose_sec\VALUE).
<P>
<ELLIPSIS>
<P>
Production of the Overthrust Monitor System will necessitate a reorganization
of our current production strategy. In order to produce the projected
quantities of the Overthrust Monitor System we will have to make the changes
summarized in <REFERENCE>(OMS_tab).
<TABLE>(Overthrust Monitor System <oparen>OMS<cparen> Production Line Impact\OMS_tab)
<TABLE_ATTRIBUTES>(WIDE)
<TABLE_SETUP>(2\18)
<TABLE_HEADS>(Production<LINE>Line Name\Production<LINE>Line Modification)
<TABLE_ROW>(Alpha<LINE> (System Units)
\100% conversion from Series II OMS production to Series III OMS production.)
<TABLE_ROW>(Beta<LINE> (Unit Stands)
\Increase production 30% and designate 50% of
that production for the Overthrust Monitor System sales.)
<TABLE_ROW>(Gamma<LINE> (Model I Power Supplies)
\Phase out production over 6 month time frame.
Will be modified to produce the new Model IIA Power Supply.)
<TABLE_ROW>(Omega<LINE> (Model IIA Power Supplies)
\Increase production by 35% until Gamma comes on-line in 6 months.)
<ENDTABLE>
```

This example may produce the following output if processed using the MILSPEC doctype keyword.

Using the MILSPEC Doctype

Sample Output of Milspec Doctype

12345B
a142-b4
22 June 1987
Part I of Three Parts

**Preliminary Development Specification
For the Overthrust Monitor System
Series (Series Configuration Number)
Order Number (Approved Order Number)**

**Submitted Under
Contract A00000-11-A-2222**

Authenticated by:

Approved by:

Procurer

Program Manager

Date

Technical Director

Consultant

Sample Output of Milspec Doctype

12345B
22 June 1987

1. SCOPE

1.1 *Scope.* This document establishes all specifications for the design and production of the Overthruster Monitor System (USN-122-233x) by our Corporation.

1.2 *Purpose.* The purpose of this document is to specify all design and production dimensions of the Overthruster Monitor System. This will ensure that all essential requirements are met and that all concerns are addressed.

1.2.1 *Primary Purpose.* The primary purpose is to enrich the quality dimension of our product.

1.2.1.1 *Secondary Purpose.* The secondary purpose is to create a corporate strategy for the product that encompasses the goals established in 1.2.1.

Production of the Overthruster Monitor System will necessitate a reorganization of our current production strategy. In order to produce the projected quantities of the Overthruster Monitor System we will have to make the changes summarized in Table I.

Table I. Overthruster Monitor System (OMS) Production Line Impact

Production Line Name	Production Line Modification
Alpha (System Units)	100% conversion from Series II OMS production to Series III OMS production.
Beta (Unit Stands)	Increase production 30% and designate 50% of that production for the Overthruster Monitor System sales.
Gamma (Model I Power Supplies)	Phase out production over 6 month time frame. Will be modified to produce the new Model IIA Power Supply.
Omega (Model IIA Power Supplies)	Increase production by 35% until Gamma comes on-line in 6 months.



6

Using the OVERHEADS Doctype

The OVERHEADS doctype lets you create pages with large bold text that copies well and is easy to see. These pages can then be used to create transparencies or 35mm photograph mounts for use on an overhead projector or similar display device.

The OVERHEADS doctype has two designs. These doctype designs are listed by their doctype keyword:

- OVERHEADS

Creates a page with an $8\frac{1}{2} \times 11$ inch format. This design can be used to create slides that fit on overhead projectors, or figures that fit into an $8\frac{1}{2} \times 11$ inch notebook.

- OVERHEADS.35MM

Creates a $6\frac{1}{2} \times 5\frac{1}{2}$ inch format that can be used to create 35mm slides by photographic reduction.

The OVERHEADS doctype supports all of the VAX DOCUMENT global tags with the following exceptions:

- <APPENDIX>
- <CHAPTER>
- <CONTENTS_FILE>
- <FRONT_MATTER>
- <GLOSSARY>
- <HEAD1> through <HEAD6>
- <INDEX_FILE>
- <PART>
- <PART_PAGE>
- <PROFILE>

See the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags.

Table 6-1 briefly describes the doctype-specific tags available in the OVERHEADS doctype. Chapter 12 contains the reference information on the tags listed in this table.

Using the OVERHEADS Doctype

Table 6-1 Tags Available in the OVERHEADS Doctype

Tag Name	Description
<AUTHOR_INFO>	Specifies from one to four lines of information about the slide presentation. This information is placed on the right side of the slide toward the bottom edge.
<AUTO_NUMBER>	Automatically numbers slides that occur in the same file. Additionally, you can specify an argument to this tag that places a text string at the beginning of each slide number, for example, "Vacation-1."
<INTRO_SUBTITLE>	Creates a secondary title of one to four lines on an introductory slide. Typically, titles created using the <INTRO_SUBTITLE> tag occupy a large amount of space on the slide.
<INTRO_TITLE>	Creates a main title of one to four lines on an introductory slide. Typically, titles created using the <INTRO_TITLE> tag occupy a large amount of space on the slide.
<RUNNING_FEET>	Places a heading at the bottom right of the current slide and all subsequent slides. If the <RUNNING_FEET> tag is used with the <AUTO_NUMBER> tag, the slide is numbered on the bottom right, and the text string is placed on the bottom left.
<RUNNING_TITLE>	Places a title at the top of all subsequent slides.
<SLIDE>	Begins a new overhead slide. Optionally, you can use this tag to specify text that is to be placed at the bottom of the slide.
<SUBTITLE>	Creates a secondary title with one to four title lines for a nonintroductory slide. The title lines produced by the <SUBTITLE> tag are output in a smaller type face with less space between lines than the title lines produced by the <INTRO_SUBTITLE> tag.
<TEXT_SIZE>	Modifies the size of the type face used in a topic, table, or list on a single slide.
<TITLE>	Creates a main title with one to four title lines for a nonintroductory slide. The title lines produced by the <TITLE> tag are output in the same type face with less space between lines than the title lines produced by the <INTRO_TITLE> tag.
<TOPIC>	Specifies a line of topic text for a slide. This text begins at the left margin and is in a smaller type font than the font used by the <TITLE> tag.

You process a file with the OVERHEADS doctype by using one of the doctype keywords in the preceding list on the DOCUMENT command line. The following example shows how to process a file named MYGRAPHIC.SDML with the OVERHEADS.35MM doctype to create a 35mm slide mount.

```
$ DOCUMENT mygraphic OVERHEADS.35MM LN03
```

You will probably want to use the LN03_LASER_PRINTER or POSTSCRIPT destinations when you create your overheads, because these destinations provide better quality output than the LINE_PRINTER destination. You may want to make xerographic copies of your printed VAX DOCUMENT files and use these copies to create the overhead transparencies. Xerographic copies of laser printer output often reproduce better than the original laser printer output.

See *VAX DOCUMENT Design Samples* for more information on the format of the OVERHEADS designs.

6.1 Sample Use of the OVERHEADS Doctype Tags

This section contains an example of two slides produced using the OVERHEADS doctype tags. The first slide is an introductory slide for a slide presentation. Note how the tags used to create this slide are particularly appropriate (for example the <INTRO_TITLE> and <INTRO_SUBTITLE> tags are used). You may find this sample useful in understanding how the tags all fit together to create overhead slides.

The SDML code for the two slides is shown first, followed by the output from that SDML code.

```
<SLIDE>(Presented 3/8/86)
<AUTO_NUMBER>(DMS)
<RUNNING_TITLE>(Pets Are People Too)
<RUNNING_FEET>(Pet selection seminar)
<INTRO_TITLE>(CHOOSING THE\RIGHT PET FOR\YOU)
<INTRO_SUBTITLE>(A Seminar on\Pet Selection)
<AUTHOR_INFO>(D. M. Smith\Veterinarian)

<SLIDE>
<TITLE>(Heart Rates\In Pets and Wild Mammals)
<SUBTITLE>(Physiological Data \on Selected Mammals)
<TOPIC>(Nondomesticated Mammal Heart Rates)
<TABLE>
<table_attributes>(KEEP)
<table_setup>(3\19\16)
<table_heads>(Common Name\Weight\Heart Rate)
<table_row>(European hedgehog\500-700 g.\246)
<table_row>(Gray shrew\3-4 g.\782)
<table_row>(Least chipmunk\40 g.\684)
<table_row>(Gray squirrel\500-600 g.\390)
<table_row>(Harbor porpoise\170 kg.\40-110)
<table_row>(Mink\0.7-1.4 kg.\272)
<table_row>(Harbor seal\20-25 kg.\18-25)
<endtable>
```

This example may produce the following output if processed using the OVERHEADS keyword.

CHOOSING THE RIGHT PET FOR YOU

A Seminar on
Pet Selection

D. M. Smith
Veterinarian

Sample Output of Overhead Doctype

Pets Are People Too

Heart Rates In Pets and Wild Mammals

Physiological Data on Selected Mammals

Nondomesticated Mammal Heart Rates

Table 1: Heart Rates of Selected Mammals

Common Name	Weight	Heart Rate
European hedgehog	500-700 g.	246
Gray shrew	3-4 g.	782
Least chipmunk	40 g.	684
Gray squirrel	500-600 g.	390
Harbor porpoise	170 kg.	40-110
Mink	0.7-1.4 kg.	272
Harbor seal	20-25 kg.	18-25

8

Using the SOFTWARE Doctype

The SOFTWARE doctype provides various formats, tags, and tag templates for those who wish to describe computer software. It is equally well-suited to describing tutorial or reference information and contains tags customized for describing software in either of these formats.

The SOFTWARE doctype lets you create documents in one of six designs:

- SOFTWARE.BROCHURE
Creates a brochure in a 7×9 inch format with unnumbered headings.
- SOFTWARE.GUIDE
Creates a users' guide in a 7×9 inch format with numbered headings.
- SOFTWARE.HANDBOOK
Creates a handbook in a 7×9 inch format with numbered headings.
- SOFTWARE.POCKET_REFERENCE
Creates a pocket reference in a $5\frac{1}{2} \times 7$ inch format with numbered headings.
- SOFTWARE.REFERENCE
Creates a reference manual in an $8\frac{1}{2} \times 11$ inch format with numbered headings.
- SOFTWARE.SPECIFICATION
Creates a specification in an $8\frac{1}{2} \times 11$ inch format with numbered headings.

The SOFTWARE doctype accepts the full range of VAX DOCUMENT global tags. See the *VAX DOCUMENT User Manual, Volume 1* for more information on global tags. It also enables a large number of tags specially suited for describing various kinds of software. Chapter 14 contains the reference information on these tags.

You process a file with the SOFTWARE doctype by using one of the doctype keywords in the preceding list on the DOCUMENT command line. The following example shows how to process a file named MYSOFTWARE.SDML with the SOFTWARE.GUIDE doctype.

```
$ DOCUMENT mysoftware SOFTWARE.GUIDE LNO3
```

See *VAX DOCUMENT Design Samples* for more information on the format of each of the SOFTWARE designs.

The SOFTWARE doctype contains doctype-specific tags that can be used throughout the SOFTWARE doctype designs and doctype-specific tags that can be used only within the context of the reference templates in the SOFTWARE doctype.

Using the SOFTWARE Doctype

The SOFTWARE doctype-specific tags are available only in the SOFTWARE doctype and allow you to create the basic writing elements that you need to describe computer software, such as terminal keys and keypads, command qualifiers and parameters, and programming language code fragments. See Section 8.1 for detailed information on these tags.

The reference template tags are doctype-specific tags that are available only in the reference templates of the SOFTWARE doctype. With these tags, you can specify the format of a software command, the restrictions on such commands, prompts used within an interactive environment, and so on. See Section 8.2 for detailed information on these tags.

8.1 SOFTWARE Doctype-Specific Tags

The SOFTWARE doctype-specific tags allow you to describe software elements in structured reference material and in less structured tutorial material. Using these tags, you can describe the following:

- Terminal keys and keypads
- Code fragments and their results
- Software messages
- Software arguments, parameters, and qualifiers
- Interactive terminal sessions

8.1.1 Documenting Terminal Keys and Keypads

The SOFTWARE doctype contains several tags that allow you to accurately represent terminal keys, keypads, and key names both in text and in examples. These tags fall into two groups:

- Tags that describe keys that can be used throughout the SOFTWARE doctype
- Tags used to create keypad diagrams that can be used only within the context of the <KEYPAD_SECTION> tag in the SOFTWARE doctype

8.1.1.1 Describing Individual Keys

Use the following tags to label and describe certain keys and key sequences that appear on keyboards and keypads.

<CPOS>	Creates a typographically distinct marking for the cursor position within an example.
<DELETE_KEY>	Creates a special character that represents the DELETE key used on keyboards.
<KEY>	Creates a label for a key from the keyboard or keypad.
<KEY_NAME>	Creates a label for a key name.
<KEY_SEQUENCE>	Creates a section in which you can depict a sequence of keys.
<GRAPHIC>	Creates a specially formatted character out of the two characters specified as arguments.

Using the <CPOS> Tag

Use the <CPOS> tag to mark the position of the cursor in a terminal example. The cursor is depicted as the underline character. The following example shows how the <CPOS> tag can be used.

```
<P>
Correct the directory specification to <QUOTE>([TEXTFILES]) by
moving the cursor to the misspelled letter in the directory
specification as in the following example:
<DISPLAY>
$ COPY ABC.TXT [T<CPOS>(R)XTFILES]
<ENDDISPLAY>
```

This example may produce the following output.

Correct the directory specification to "[TEXTFILES]" by moving the cursor to the misspelled letter in the directory specification as in the following example:

```
$ COPY ABC.TXT [TRXTFILES]
```

Using the <DELETE_KEY> Tag

Use the <DELETE_KEY> tag to create the character used on most terminal and typewriter keyboards for the DELETE key. The following example shows how the <DELETE_KEY> tag can be used.

```
<P>
Press the DELETE key (<DELETE_KEY>) to delete a character.
```

This example may produce the following output.

Press the DELETE key (X) to delete a character.

Using the <GRAPHIC> Tag

Use the <GRAPHIC> tag to create special graphic characters that appear on the terminal screen. The <GRAPHIC> tag accepts two characters as arguments and formats them next to each other with the second character formatted slightly below the first character. This tag allows you to create representations of the linefeed and formfeed characters as well as other similarly-formatted characters.

The following example shows how the <GRAPHIC> tag can be used.

```
<P>
Two special characters that will appear in your editing session
are the linefeed (<GRAPHIC>(L\F)) and the formfeed (<GRAPHIC>(F\F))
characters.
```

This example may produce the following output.

Two special characters that will appear in your editing session are the linefeed (L_F) and the formfeed (F_F) characters.

Using the <KEY_NAME> Tag

Use the <KEY_NAME> tag to differentiate the name of a key from the other output in an example or in text.

The following example shows how the <KEY_NAME> tag can be used.

```
<P>
Press the <KEY_NAME>(HELP) or the <KEY_NAME>(DO) key for more information.
```

Using the SOFTWARE Doctype

This example may produce the following output.

Press the HELP or the DO key for more information.

Using the <KEY> Tag

Use the <KEY> tag to represent a terminal key either in text or in an example. The <KEY> tag accepts a *key-label* argument, which specifies the name of the key.

If you want to represent a terminal key in text, you can use the TEXT keyword argument to the <KEY> tag to place angle brackets before and after the key label. Use the BOX keyword argument to the <KEY> tag to place the key label in a box that resembles a key to represent a terminal key in an example. If neither the TEXT nor the BOX keyword is specified, the default format is BOX.

The following example shows how the <KEY> tag can be coded as it would appear in text using the TEXT keyword argument, and as it would appear in an example using the BOX keyword argument.

Note that the <KEY> tag is used within the context of the <KEY_SEQUENCE> tag. See the description of the <KEY_SEQUENCE> tag in this section for more information on that tag.

```
<P>  
You should now press the <KEY>(RETURN\TEXT) key to insert a line.  
<KEY_SEQUENCE>  
<KEY>(RETURN\BOX)  
<ENDKEY_SEQUENCE>
```

This example may produce the following output.

You should now press the <RETURN> key to insert a line.

```
RETURN
```

Using the <KEY_SEQUENCE> Tag

Use the <KEY_SEQUENCE> tag to give an example of a sequence of keys. For example, you might wish to describe the sequence of keys needed to exit a text editor.

The <KEY_SEQUENCE> tag enables the <KEY_PLUS> and the <KEY_TYPE> tags to make creating such key sequences easier. The <KEY_PLUS> tag creates a plus sign (+) between two keys, and the <KEY_TYPE> tag lets you associate a key sequence with some textual information, such as a terminal type. In addition to these two tags, the <KEY> tag can also be used within the context of the <KEY_SEQUENCE> tag.

When the <KEY> tag is used within a key sequence, it accepts an additional argument, which lets you place two key labels rather than one inside a box or angle brackets. When two key labels are used, they are stacked together with the first argument placed on the top. This extra argument makes it possible to specify keys that use two stacked words as their label, such as the "Next Screen" key.

The following code fragment contains a series of key examples within a <KEY_SEQUENCE> section. Note that the <KEY> tag is used in two contexts in this example. Within the context of the <KEY_SEQUENCE> tag the <KEY> tag accepts two key label arguments. Outside of the context of the <KEY_SEQUENCE> tag the <KEY> tag accepts only a single-key label argument.

Using the SOFTWARE Doctype

- `<KEYPAD_ENDROW>`

Creates a special three-column keypad row for the larger keys on the bottom row of the keypad.

If you use the `DISPLAY` keyword argument with the `<KEYPAD>` tag, you can specify arguments to the `<KEYPAD_ROW>` and `<KEYPAD_ENDROW>` tags that will place text on the appropriate key in the keypad diagram.

If you do not use the `DISPLAY` keyword, you can specify only one of three keywords as the arguments to the `<KEYPAD_ROW>` and `<KEYPAD_ENDROW>` tags.

The `<KEYPAD_ROW>` and `<KEYPAD_ENDROW>` tags accept the same keyword arguments, which allow you to specify whether a key should be drawn, and whether a key that is drawn should be shaded in. If no keyword argument is specified for a particular keypad column, the key is drawn and it is left open (not shaded in).

The following keywords are accepted by the `<KEYPAD_ROW>` and `<KEYPAD_ENDROW>` tags:

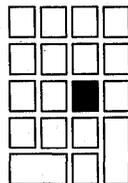
- `OPEN` — Specifies that the key in that column is to be left blank. `OPEN` is the default.
- `CLOSED` — Specifies that the key in that column is to be shaded in.
- `NONE` — Specifies that no key should be drawn in that column.

The following examples show how to use the keypad section tags to create various keypad diagrams. The first example shows a complete keypad with one key shaded in.

```
<KEYPAD_SECTION>
<KEYPAD>(A Complete Keypad Diagram)
<KEYPAD_ROW>( \ \ \ )
<KEYPAD_ROW>( \ \ \ )
<KEYPAD_ROW>( \ \ \CLOSED\ )
<KEYPAD_ROW>( \ \ \NONE )
<KEYPAD_ENDROW>( \ \ )
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output:

A Complete Keypad Diagram



The following example shows a single line from the previous keypad diagram:

```
<KEYPAD_SECTION>
<KEYPAD>(A Single Keypad Line)
<KEYPAD_ROW>( \ \ \CLOSED\ )
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output:

A Single Keypad Line

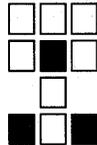


The following example shows the complete keypad with some keys eliminated from the diagram. You can use the NONE keyword argument to eliminate keys from the keypad diagram.

```
<KEYPAD_SECTION>
<KEYPAD>(An Irregular Keypad)
<KEYPAD_ROW>( \ \ \NONE)
<KEYPAD_ROW>( \CLOSED\ \NONE)
<KEYPAD_ROW>(NONE\ \NONE\NONE)
<KEYPAD_ROW>(CLOSED\ \CLOSED\NONE)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output:

An Irregular Keypad



The following example shows how the DISPLAY keyword argument to the <KEYPAD> tag allows you to specify text as the arguments to the <KEYPAD_ROW> and <KEYPAD_ENDROW> tags.

```
<KEYPAD_SECTION>
<KEYPAD>(The Editing Keypad With Key Labels\DISPLAY)
<KEYPAD_ROW>(PF1\PF2\PF3\PF4)
<KEYPAD_ROW>(7\8\9\-)
<KEYPAD_ROW>(4\5\6\.)
<KEYPAD_ROW>(1\2\3\ )
<KEYPAD_ENDROW>(0\.\ENTER)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output:

The Editing Keypad With Key Labels

PF1	PF2	PF3	PF4
7	8	9	-
4	5	6	.
1	2	3	ENTER
0		.	

Using the SOFTWARE Doctype

```
<KEYPAD_SECTION>
<KEYPAD>(The Editing Keypad With Key Labels and Blacked-in Keys\DISPLAY)
<KEYPAD_ROW>(CLOSED\PF2\PF3\PF4)
<KEYPAD_ROW>(7\8\9\-)
<KEYPAD_ROW>(CLOSED\5\6\,)
<KEYPAD_ROW>(1\2\3\ )
<KEYPAD_ENDROW>(0\.\ENTER)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output:

The Editing Keypad With Key Labels and Blacked-in Keys

	PF2	PF3	PF4
7	8	9	-
	5	6	,
1	2	3	ENTER
0	.		

8.1.2 Documenting Code Fragments

There are three tags available in the SOFTWARE doctype that let you describe software code fragments. These tags let you create syntax statements, emphasize arguments, and create samples of screen displays.

- <DISPLAY> Simulates a screen display.
- <SYNTAX> Formats text on the page exactly as entered to allow correct positioning of code or syntax statements.
- <ARGUMENT> Emphasizes arguments to functions, procedures, or tags within text using an altered text font (such as bold face or italic).

Using the <DISPLAY> tag

The <DISPLAY> tag lets you create an example that simulates a screen display. This example is terminated by the <ENDDISPLAY> tag. The <DISPLAY> tag accepts one of two keyword arguments.

The WIDE keyword argument extends the display format into the left margin. The KEEP keyword argument specifies that the entire display example should be placed on the next page if it does not fit on the existing page. This argument is used to prevent unnecessary page breaks in display examples.

The following example shows how you can use the <DISPLAY> tag to simulate a screen display. This example was coded using the WIDE argument due to the width of the example. Note that all the spacing in the display is retained as it was entered.

```
<DISPLAY>(WIDE)
VAX/VMS V4.4 on node XXXXXX 6-NOV-1986 17:43:18.03 Uptime 0 02:08:56
  Pid Process Name State Pri I/O CPU Page flts Ph.Mem
20400080 NULL COM 0 0 0 00:15:46.03 0 0
20400081 SWAPPER HIB 16 0 0 00:00:19.74 0 0
20400085 ERRFMT HIB 7 157 0 00:00:01.32 67 104
20400086 CACHE_SERVER HIB 16 6 0 00:00:00.12 57 92
20400087 CLUSTER_SERVER HIB 10 14 0 00:00:00.61 109 257
20400088 OPCOM LEF 7 126 0 00:00:01.06 332 77
20400089 JOB_CONTROL HIB 8 2081 0 00:00:22.95 190 322
2040008A CONFIGURE HIB 9 19 0 00:00:00.18 99 136
20400092 SYMBIONT_0001 HIB 6 312 0 00:00:06.34 2886 63
<ENDDISPLAY>
```

This example may produce the following output.

```
VAX/VMS V4.4 on node XXXXXX 6-NOV-1986 17:43:18.03 Uptime 0 02:08:56
  Pid Process Name State Pri I/O CPU Page flts Ph.Mem
20400080 NULL COM 0 0 0 00:15:46.03 0 0
20400081 SWAPPER HIB 16 0 0 00:00:19.74 0 0
20400085 ERRFMT HIB 7 157 0 00:00:01.32 67 104
20400086 CACHE_SERVER HIB 16 6 0 00:00:00.12 57 92
20400087 CLUSTER_SERVER HIB 10 14 0 00:00:00.61 109 257
20400088 OPCOM LEF 7 126 0 00:00:01.06 332 77
20400089 JOB_CONTROL HIB 8 2081 0 00:00:22.95 190 322
2040008A CONFIGURE HIB 9 19 0 00:00:00.18 99 136
20400092 SYMBIONT_0001 HIB 6 312 0 00:00:06.34 2886 63
```

Using the <SYNTAX> tag

The <SYNTAX> tag lets you distinguish the syntax of a programming language statement from regular text. The <SYNTAX> tag distinguishes the syntax statement by making the typeface of the statement different from the typeface used in the surrounding text. The <SYNTAX> tag is terminated by the <ENDSYNTAX> tag.

The <SYNTAX> tag accepts two optional arguments. The WIDE keyword argument allows more width for the syntax statement by letting the syntax statement extend into the left margin. The *alternate-heading* argument lets you specify a heading for the syntax statement.

The following example shows how you can use the <SYNTAX> tag to separate a syntax statement from the surrounding text. Note that all the spacing in the statement is retained as it was entered.

```
<SYNTAX>
IF condition THEN
  statement-list
[ELSE
  statement-list];
<ENDSYNTAX>
```

This example may produce the following output:

```
IF condition THEN
  statement-list
[ELSE
  statement-list];
```

Using the SOFTWARE Doctype

Using the <ARGUMENT> tag

The <ARGUMENT> tag lets you label an argument by displaying that argument in a typeface that differs from that of the surrounding text (for example, in some doctypes it may cause the argument to be displayed in a bold typeface). This tag accepts only the argument name as an argument.

The following example shows how you can use the <ARGUMENT> tag to separate the argument from the surrounding text.

The CHR\$ function converts the <ARGUMENT>(char-data) argument into a numeric value.

This example may produce the following output.

The CHR\$ function converts the **char-data** argument into a numeric value.

8.1.3 Documenting Software Messages

You can describe the messages put out by software programs by using the <MESSAGE_SECTION> tag and the tags it enables. The tags made available by the <MESSAGE_SECTION> tag allow you to label messages issued by an operating system or an application program, and to enter descriptions of those messages.

The tags enabled by the <MESSAGE_SECTION> tag are summarized in the following list.

<MESSAGE_TYPE>	Specifies the type of message being described using the <MSG> or <MSGS> tag.
<MSG>	Labels and formats up to two lines of message text within the message description section.
<MSGS>	Labels and formats up to nine lines of message text within the message description section.
<MSG_TEXT>	Specifies the explanatory text associated with the messages described by the <MSG> or the <MSGS> tag.

Messages generally come in one of three forms:

- A message text string only. An example of this type of string is "System Resources Unavailable."
- A message text string preceded by a text string identification code. An example of this type of string is "%DIRECT-W-NOFILES, no files found."
- A message text string preceded by a numeric string identification code. An example of this type of string is "%1288374 file lookup failed."

The <MESSAGE_SECTION> tag begins the message description section. This tag is terminated by the <ENDMESSAGE_SECTION> tag. Select the tags you need to use in the message description section based upon the following criteria:

- The format your messages most closely follows
- The number of lines each message occupies on the terminal display

Using the <MESSAGE_TYPE> tag

Use the <MESSAGE_TYPE> tag to specify the type of message labeled by the <MSG> or <MSGS> tags. The <MESSAGE_TYPE> tag accepts one of three keywords as the argument that determines the type of messages being described:

- NOIDENT

Specifies that the <MSG> and <MSGS> tags in the message section will not be passed a numeric or text identification string as the first argument. NOIDENT is the default keyword. Specifying NOIDENT as the keyword argument has the same effect as not specifying the <MESSAGE_TYPE> tag at all.

- TEXTIDENT

Specifies that the <MSG> and <MSGS> tags in the message section will be passed a text identification string as an argument.

Text identification strings and message text arguments should be specified as pairs. The TEXTIDENT keyword creates a comma (,) between text identification strings and message text arguments.

- NUMIDENT

Specifies that the <MSG> and <MSGS> tags in the message section will be passed a numeric identification string as an argument.

Numeric identification strings and message text arguments should be specified as pairs. The NUMIDENT keyword assumes that the numeric identification string specified will not exceed a total length of six picas (approximately ten characters).

The following is a sample use of the <MESSAGE_TYPE> tag. Note how the NUMIDENT keyword indicates that the message being described has a numeric identification code.

```
<MESSAGE_SECTION>  
<MESSAGE_TYPE>(NUMIDENT)  
<MSG>(%1288374\file lookup failed.)  
.  
.
```

Using the <MSG> tag

The <MSG> tag is used for messages that occupy only one or two lines when they are output. The arguments accepted by the <MSG> tag are based upon the type of message being described. The message type is determined by the keyword argument used with the <MESSAGE_TYPE> tag.

- NOIDENT keyword

The <MSG> tag requires a message text argument and accepts an optional second message text argument.

Using the SOFTWARE Doctype

- TEXTIDENT keyword

The <MSG> tag requires a text message identification string and a message text argument. It also accepts a second line of message text as an optional third argument. The TEXTIDENT keyword creates a comma (,) between text identification strings and message text arguments. The optional second message text argument is stacked under the first message text.

- NUMIDENT keyword

The <MSG> tag requires a numeric message identification string and a message text argument. It also accepts a second line of message text as an optional third argument. The numeric message identification string must be no longer than ten characters (six picas) in length. The optional second message text argument is stacked under the first message text.

The following is a sample use of the <MSG> tag. Note how the optional third argument is used for the additional message text.

```
<MESSAGE_SECTION>
<MESSAGE_TYPE>(NUMIDENT)
<MSG>(%1244374\file lookup failed\directory not found)
.
.
```

Using the <MSG> tag

The <MSG> tag is used for messages that occupy from one or nine lines when they are output. The arguments accepted by the <MSG> tag are based upon the type of message being described. The message type is determined by the keyword argument used with the <MESSAGE_TYPE> tag.

- NOIDENT keyword

The <MSG> tag requires one message text argument and accepts up to a total of nine message text arguments.

- TEXTIDENT keyword

The <MSG> tag requires text message identification strings as the first and optionally as the third, fifth, and seventh arguments. It also requires a message text argument as the second argument and optionally as the fourth, sixth and eighth arguments. The TEXTIDENT keyword creates a comma (,) between text identification strings and message text arguments. The optional arguments are stacked under the required arguments.

- NUMIDENT keyword

The <MSG> tag requires numeric message identification strings as the first and optionally as the third, fifth, and seventh arguments. It also requires a message text argument as the second argument and optionally as the fourth, sixth and eighth arguments. The text message identification strings must be no more than 10 characters (approximately six picas in length). The optional arguments are stacked under the required arguments.

The following is a sample use of the <MSG> tag. Note how the numeric identification codes and the text strings associated with them are specified in pairs.

```
<MESSAGE_SECTION>
<MESSAGE_TYPE>(NUMIDENT)
<MSG>(%133455\read write error%\0000221\disk not available
\6644544\file not found)
```

Using the <MSG_TEXT> tag

The <MSG_TEXT> tag labels the text that describes the messages listed by the <MSG> or <MSG> tags. By default the <MSG_TEXT> tag has a heading of "Explanation:." If you want the heading for your text to be more descriptive, you can specify an alternate heading for this tag (for example, "User Action"). Note that any alternate heading specified for the <MSG_TEXT> tag will have a colon (:) appended to the end of the heading when it is output.

The following example shows how you can use the tags enabled by the <MESSAGE_SECTION> tag to create a message description section. Note how using the <MESSAGE_TYPE> tag with the TEXTIDENT keyword results in a comma (,) being placed after each message identification string.

```
<MESSAGE_SECTION>
<MESSAGE_TYPE>(TEXTIDENT)
<MSG>(BACKLINK\Incorrect directory back link)
<MSG_TEXT>(Facility) VERIFY, Verify Utility
<MSG_TEXT>(Severity) BACKLINK-F-BADLINK
<MSG_TEXT>
The Verify Utility could not process your command, please check the
syntax of your statement.
<MSG>(UAF-E-NAOFIL\Unable to open file SYSUAF.DAT\-RMS-E-FNF\file not found)
<MSG_TEXT>(User Action)
Check the syntax and reenter the command.
<MSG_TEXT>
This is some explanatory text for the previous message.
<ENDMESSAGE_SECTION>
```

This example may produce the following output.

BACKLINK, Incorrect directory back link

Facility: VERIFY, Verify Utility

Severity: BACKLINK-F-BADLINK

Explanation: The Verify Utility could not process your command, please check the syntax of your statement.

UAF-E-NAOFIL, Unable to open file SYSUAF.DAT
-RMS-E-FNF, file not found

User Action: Check the syntax and reenter the command.

Explanation: This is some explanatory text for the previous message.

Using the SOFTWARE Doctype

8.1.4 Documenting Arguments, Parameters and Qualifiers

There are four sets of tags available in the SOFTWARE doctype for describing arguments, parameters, and qualifiers in a list.

<ARGDEFLIST>	Creates a definition list of arguments
<PARAMDEFLIST>	Creates a definition list of parameters
<QUALDEFLIST>	Creates a definition list of qualifiers
<QUAL_LIST>	Creates a summary list of qualifiers

The tags used to create these SOFTWARE doctype definition lists and the form of these lists are very similar to those used by the global <DEFINITION_LIST> tag. The qualifier summary list differs from the definition lists in that it is designed only as a summary list of qualifiers and contains no provision for definition text.

These SOFTWARE doctype-specific tags are used to label lists of arguments, parameters or qualifiers inside or outside the context of the reference templates.

Using the SOFTWARE Definition List Tags

The following list describes the software definition list tag sets. Note that tags that function in the same manner in each of the definition lists are described together.

Software Definition List Tags

<ARGDEFLIST>
<PARAMDEFLIST>
<QUALDEFLIST>

Begins each type of list and enables the following tags. Each of these tags allows an optional heading to be specified as an argument. The NONE keyword argument may also be used to indicate that the list contains no items.

If these tags are used inside of a reference template, default headings may be defined.

<ARGITEM>
<PARAMITEM>
<QUALITEM>

Labels the argument, parameter or qualifier to be listed. Each of these tags requires one argument and accepts up to a total of seven arguments to list any related arguments, parameters, or qualifiers.

<ARGDEF>
<PARAMDEF>
<QUALDEF>

Labels the text string that describes the appropriate argument, parameter or qualifier.

<ENDARGDEFLIST>
<ENDPARAMDEFLIST>
<ENDQUALDEFLIST>

Terminates each type of list and disables the contained tags.

Using the SOFTWARE Doctype

The following examples show how to create an argument definition list, a parameter definition list, and a qualifier definition list.

The first example shows a parameter definition list within the Command template. This coding of the <PARAMDEFLIST> tag uses the NOHEAD keyword to suppress the output of a default heading (in this case, the heading "parameters.") If this definition list were coded outside of the context of a reference template, no default heading would be defined and the NOHEAD argument would not be needed.

```
<P>The system maintains logical names and their
associated equivalence strings in two types of tables:
<PARAMDEFLIST>(NOHEAD)
<PARAMITEM>(process-private)
<PARAMDEF>These tables contain logical names that are available only
to your process.
<PARAMITEM>(shareable)
<PARAMDEF>These tables contain logical names that are available to other
processes on the system.
<ENDPARAMDEFLIST>
```

This example may produce the following output:

The system maintains logical names and their associated equivalence strings in two types of tables:

process-private

These tables contain logical names that are available only to your process.

shareable

These tables contain logical names that are available to other processes on the system.

The following example shows how to use the <ALIGN_AFTER> tag for additional formatting flexibility in an argument definition list outside of a reference template.

```
<ARGDEFLIST>
<ARGITEM>(STATUS:arg\
<ALIGN_AFTER>(STATUS:)COMMAND\
<ALIGN_AFTER>(STATUS:)TASK)
<ARGDEF>Specifies whether exit status is to be returned
from the RUN command.
<ENDARGDEFLIST>
```

This example may produce the following output:

```
STATUS:arg
COMMAND
TASK
```

Specifies whether exit status is to be returned from the RUN command.

Using the SOFTWARE Doctype

The following example shows a qualifier definition list within the Command template. Note how the related qualifiers are stacked.

```
<COMMAND_SECTION>
<QUALDEFLIST>(Qualifiers)
<QUALITEM>( /HERE\ /THERE)
<QUALDEF>
The /HERE qualifer specifies the location of your workplace;
the /THERE qualifier specifies the location of your home;
/THERE is the default.
<QUALITEM>( /TIME\ /NOTIME)
<QUALDEF>
Specifies the amount of free time you have available.
If the /TIME qualifier is used, an amount must be given;
/NOTIME is the default.
<ENDQUALDEFLIST>
<ENDCOMMAND_SECTION>
```

This example may produce the following output:

Qualifiers

/HERE
/THERE

The /HERE qualifier specifies the location of your workplace; the /THERE qualifier specifies the location of your home; /THERE is the default.

/TIME
/NOTIME

Specifies the amount of free time you have available. If the /TIME qualifier is used, an amount must be given; /NOTIME is the default.

Using the Software Qualifier Summary List Tags

VAX DOCUMENT provides several tags to create a summary list of command qualifiers. The list generated by these tags creates two default headings, and places one heading over each qualifier listed by the <QPAIR> tag. Although qualifier summary lists are most often used following the <FORMAT> tag in the Command template, they may be used either inside or outside of the context of the reference templates.

Use the following tags to create a qualifier summary list:

- `<QUAL_LIST>`

Begins the list and enables the `<QUAL_LIST_HEADS>`, `<QUAL_LIST_DEFAULT_HEADS>`, `<QPAIR>`, and `<ENDQUAL_LIST>` tags. This tag places two default headings on the page. The heading "Command Qualifiers" is placed over the first list column and the heading "Defaults" is placed over the second list column. These headings may be overridden by the `<QUAL_LIST_HEADS>` or `<QUAL_LIST_DEFAULT_HEADS>` tags.

The NONE keyword argument can be used to indicate that there are no qualifiers. If the NONE keyword is used, the `<ENDQUAL_LIST>` tag should not be used. This tag is terminated by the `<ENDQUAL_LIST>` tag.
- `<QUAL_LIST_HEADS>`

Creates alternate headings for a single qualifier summary list. This tag requires two arguments, which specify the alternate headings. If either argument is null, the associated heading is not output.
- `<QUAL_LIST_DEFAULT_HEADS>`

Creates new default headings for all subsequent qualifier summary lists. This tag requires two arguments, which specify the new default headings. If either argument is null, the associated heading is not output.
- `<QPAIR>`

Specifies the pair of qualifiers to be listed. This tag requires two qualifier arguments; the first argument is listed under the default heading "Command Qualifiers," the second argument is listed under the default heading "Defaults."

The following example shows a qualifier summary list. Note how no description text is used. Note also how each list has a default heading.

```
<QUAL_LIST>
<QPAIR>( /HERE\ /THERE)
<QPAIR>( /TIME\ /NOTIME)
<ENDQUAL_LIST>
```

This example may produce the following output.

Command Qualifiers	Defaults
<i>/HERE</i>	<i>/THERE</i>
<i>/TIME</i>	<i>/NOTIME</i>

8.1.5 Creating A Series of Interactive or Code Examples

Often, it is useful to explain concepts through the use of examples. You can create a series of numbered interactive and code examples by using the `<EXAMPLE_SEQUENCE>` tag.

Interactive and code examples within an example sequence have the same form and accept the same tags as examples created using the global `<INTERACTIVE>` and `<CODE_EXAMPLE>` tags.

You can use the following tags to construct a sequence of examples:

- `<EXAMPLE_SEQUENCE>` tag

Begins a sequence of numbered informal examples. This tag accepts two optional arguments, which are used to alter the heading and suppress the numbering of examples.

The *heading-info* argument alters the example sequence heading. This argument can be one of the following:

- An alternate heading for the example sequence
- The `NOHEAD` keyword, which suppresses the output of a heading for the example sequence
- The `EXAMPLE` keyword, which suppresses numbering of the examples in the sequence and outputs the heading "Example"; this keyword should be used when you have a single informal example rather than a series of examples.

The `NONUMBER` keyword argument suppresses the numbering of examples in an example sequence. When the keyword `EXAMPLE` is used as the *heading-info* argument, the `NONUMBER` keyword argument is unnecessary.

The `<EXAMPLE_SEQUENCE>` tag enables the `<EXAMPLES_INTRO>`, `<EXC>`, `<EXI>`, and `<EXTTEXT>` tags. This tag is terminated by the `<ENDEXAMPLE_SEQUENCE>` tag.

- `<EXAMPLES_INTRO>`

Specifies introductory text for the example sequence.

- `<EXC>`

Specifies the beginning of a code example within an example sequence. This code example is formatted exactly the same as those created using the global `<CODE_EXAMPLE>` tag. The code example is terminated by the `<EXTTEXT>` tag.

- `<EXI>`

Specifies the beginning of an interactive example within an example sequence. This tag uses the global `<S>` and `<U>` tags within the example in the same manner as the global `<INTERACTIVE>` tag. The interactive example is terminated by the `<EXTTEXT>` tag.

- `<EXTTEXT>`

Specifies text that explains the previous example in the sequence. This tag is required to terminate examples begun using the `<EXC>` and `<EXI>` tags.

The following example shows how the example sequence tags can be used. Note the alternate heading specified as an argument to the <EXAMPLE_SEQUENCE> tag.

```
<EXAMPLE_SEQUENCE>(An Interactive Example and a Code Example)
<EXAMPLES_INTRO>
This is introductory text for the sample examples.
<exi><S>($ )<U>(SET WORK/NOTIME)
<EXTEXT>
This command sets the /NOTIME qualifier to the SET WORK command.
<EXC>
This is a code example,                               ined, exactly as entered.
               note how f                               a
                   o                               t
                   r                               e
                   matting is r

<EXTEXT>
This shows the flexibility available within a code example in an example
sequence.
<ENDEXAMPLE_SEQUENCE>
```

This example may produce the following output:

An Interactive Example and a Code Example

This is introductory text for the sample examples.

1 \$ SET WORK/NOTIME

This command sets the /NOTIME qualifier to the SET WORK command.

2 This is a code example, ined, exactly as entered.
 note how f a
 o t
 r e
 matting is r

This shows the flexibility available within a code example in an example sequence.

8.2 Using the Reference Templates

The SOFTWARE doctype contains four *tag templates* that let you create software reference documentation. A tag template is a set of tags that are intended for some specialized documentation purpose, such as creating an outline, composing the front matter of a book, or documenting a series of software commands.

Every tag template has a beginning and an end. These template boundaries are set by a pair of tags:

- The *template-enabling tag* begins the template and sets up all the template-specific formats and tag definitions.
- The *template-ending tag* ends the template and disables the template-specific formats and definitions.

If template-specific tags are encountered outside of the template in which they are defined, these tags will be undefined and VAX DOCUMENT will issue a warning message. Note that you must end a template before you begin another template; templates cannot be nested inside one another.

Using the SOFTWARE Doctype

Tag templates are especially useful in creating and maintaining reference documentation. They make the coding of reference information easier in several ways:

- The structured nature of tag templates makes it easier to consistently code, format, and order reference information.
- Text coded into a tag template is more modular and structured than the text in a nontemplated SDML file, which makes it easier to modify and maintain reference information.
- Using a tag template for reference information allows the writer to “fill in the blanks,” and helps ensure that no essential information, such as restrictions or the lack of restrictions on a command, is omitted.
- Using a tag template for reference information lets you use default formats, headings, and tags. This helps guarantee that even when several writers work on different portions of the same book, all of these portions look alike.

For these reasons, VAX DOCUMENT provides the *reference templates* that are tag templates specialized for the description of software reference information.

Using the SOFTWARE Doctype Reference Templates

There are four reference templates available in the SOFTWARE doctype to create reference documentation for software commands, routines, statements, or VAX DOCUMENT tags:

- **Command template**
Describes commands and their various components. Examples of command descriptions are the descriptions of the DCL commands used in the VMS operating system.
- **Routine template**
Describes software routines and their various components. Examples of routines descriptions are the descriptions of the VMS run-time library routines.
- **Statement template**
Describes programming language constructs (such as statements and functions) and their various components. An example of a statement description is the CASE statement available in the VAX Pascal programming language.
- **Tag template**
Describes VAX DOCUMENT tags and their components. Examples of tag descriptions can be found in the reference chapters in this manual.

These templates are similar in design because all are intended to be used to create similarly formatted reference material. This similarity means that certain tags are common to all four templates; however, because each template is customized for the presentation of a particular kind of information, several tags are available only in specific templates. For example, all four templates have a description section that describes the current command, routine, statement, or tag. However, only the Routine template has a section for describing in detail the values returned by a routine.

Like all tag templates, the software reference templates are begun by a template-enabling tag (for example, the `<COMMAND_SECTION>` tag), and terminated by a similarly-named template-ending tag (for example, the `<ENDCOMMAND_SECTION>` tag). However, because the reference templates are designed to create reference-oriented documentation, they also must contain one or more *reference-element* tags.

A reference-element is a single element in a reference section; for example, the description of a single command in a command reference section, or the description of a single routine in a routine reference section. It is the collection of these single elements into a group that creates a reference section.

The default reference-element tag names match the templates they are used in. The `<COMMAND>` tag occurs in the Command template, the `<ROUTINE>` tag occurs in the Routine template, and so on. When these reference-element tags occur in a reference template, they signal the beginning of a new reference-element description.

By default, each new reference-element description begins on a new output page, with the name of the current reference-element output at the top of the page.

Reference-element tags are not terminated by matching ending tags as are most template tags. Instead, a reference-element description is terminated either by the next reference-element tag, by the end of the reference template, or by the end of the SDML file.

Note that of all the tags used in the various reference templates, only the template-enabling tags and the reference-element tags are required; all other tags used in the templates are optional.

The following list shows the template-enabling tag and the reference-element tag for each template.

Template Name	Template-Enabling Tag	Reference-Element Tag
Command	<code><COMMAND_SECTION></code>	<code><COMMAND></code>
Routine	<code><ROUTINE_SECTION></code>	<code><ROUTINE></code>
Statement	<code><STATEMENT_SECTION></code>	{ <code><STATEMENT></code> ¹ }
		{ <code><FUNCTION></code> }
Tag	<code><TAG_SECTION></code>	<code><SDML_TAG></code>

¹The use of braces indicates that the enclosed tag names are used interchangeably. VAX DOCUMENT provides two such names to improve the readability of your SDML file.

The Default Format of the Reference Templates

The software reference templates all have the same default output format. When a reference template is begun by specifying a template-enabling tag, the template has the following format:

- A new reference-element description is begun and placed at the beginning of a new output page.

Using the SOFTWARE Doctype

For example, each time a `<COMMAND>` tag is encountered in the command template, a new reference-element description is begun and placed at the beginning of the next output page.

- The name of the current reference-element is placed on each output page. The placement of this name depends on the doctype used. In most doctypes, this name is placed at the top of the page.

For example, if a command called SET TIME was being described and had been specified as `<COMMAND> (SET TIME)` in the SOFTWARE.REFERENCE doctype, then the single line running heading "SET TIME" would be placed at the top of the output page above the reference description.

- Each page is numbered using the last major page number prefix.

For example, if Chapter 2 was the immediately preceding chapter, then "2" would be the prefix associated with the page numbers in the reference section; likewise "A" would be the prefix if appendix A had preceded the reference section.

- Default headings are defined by the reference template for those tags that have such headings.

For example, the `<RESTRICTIONS>` tag has associated with it the default heading text "Restrictions" in the templates in which it is available.

Using the NONE Keyword Argument to Template Tags

Most of the reference template tags place a heading on the output page. All of these tags accept the keyword NONE as an argument. This keyword indicates that the heading should be placed on the output page followed by the text "None."

By using the NONE keyword, you ensure that a particular kind of information is explicitly declared as nonexistent or not applicable in a reference-element description. Use the NONE keyword rather than typing the word "None." under the heading because inconsistencies in formatting and spelling of the word may otherwise be introduced.

When you use the NONE keyword as an argument to a template tag, the tag that terminates that template tag must be omitted. For example, if you specify the NONE argument to the `<PARAMDEFLIST>` tag, the `<ENDPARAMDEFLIST>` tag must be omitted.

Of all the tags that output a heading in the reference templates, only the `<FORMAT>`, `<STATEMENT_FORMAT>`, and `<DESCRIPTION>` tags do not accept a keyword argument of NONE, because placing the text "None." beneath the headings output by any of these tags would be inappropriate. If there is no format or description sections, these tags should be omitted.

Using Samples of the Reference Templates

VAX DOCUMENT provides the following sample reference template SDML files in the directory DOC\$TEMPLATES:

Command Template: DOC\$TEMPLATES:COMMAND.SDML
 Routine Template: DOC\$TEMPLATES:ROUTINE.SDML
 Statement Template: DOC\$TEMPLATES:STATEMENT.SDML
 Tag Template: DOC\$TEMPLATES:TAG.SDML

You can also enter an editing session with LSE and expand the appropriate template tags into a reference template SDML file. See the *VAX DOCUMENT User Manual, Volume 1* for more information on using LSE with VAX DOCUMENT.

See the template samples at the end of this chapter for examples of the SDML code and output from each of the four reference templates.

8.2.1 Documenting Reference-Elements within Templates

When describing a reference-element (such as a command or function) in a template, it is often useful to provide both a brief description and a detailed explanation of the reference-element. The tags used for this purpose are the <OVERVIEW> and <DESCRIPTION> tags.

- <OVERVIEW>

Used to briefly describe the reference-element. Such overviews are typically a single-sentence summary of the reference-element.

- <DESCRIPTION>

Used to describe the reference-element in detail. This description may cover suggested use, special considerations, or how to use the reference-element with related commands, routines, statements, or tags. Such description sections can be as short as two or three sentences or may be as long as several pages.

Chapter 14 contains more detailed reference material on each of these tags.

The following SDML code fragment shows a sample use of the <OVERVIEW> and <DESCRIPTION> tags within the context of the Command template. The <COMMAND> tag and the global <FORMAT> and <PARAMDEFLIST> tags are included to make a more complete example. Note how the inclusion of the reference-element tag (<COMMAND>) forces a page break by default.

```
<COMMAND_SECTION>
<COMMAND>(SET TIME)
<OVERVIEW>
Resets the system time to the time specified as a parameter to this command.
<ENDOVERRIDE>
<FORMAT>
<FCMD>(SET TIME) <FPARMS>(time-specification)
<ENDFORMAT>
<PARAMDEFLIST>
<PARAMITEM>(time-specification)
<PARAMDEF> Specifies the time to which the system should be set.
<ENDPARAMDEFLIST>
<DESCRIPTION>
The SET TIME command lets you reset the system clock on your VMS system.
This command is generally used to make allowances for daylight savings time
and other such events.
<ENDDescription>
<ENDCOMMAND_SECTION>
```

This example may produce the following output:

SET TIME

Resets the system time to the time specified as a parameter to this command.

FORMAT **SET TIME** *time-specification*

PARAMETERS *time-specification*
Specifies the time to which the system should be set.

DESCRIPTION The SET TIME command lets you reset the system clock on your VMS system. This command is generally used to make allowances for daylight savings time and other such events.

8.3 Creating Your Own Reference Template Tags

VAX DOCUMENT provides the following tags for the creation of specialized reference template tags:

- `<SET_TEMPLATE_LIST>`
Creates a user-defined set of tags for creating your own headed list within the template.
- `<SET_TEMPLATE_PARA>`
Creates a user-defined set of tags for creating your own headed template paragraph section within the template.
- `<SET_TEMPLATE_TABLE>`
Creates a user-defined set of tags for creating your own headed table within the template.

Each of these tags defines a template tag and its terminator for use within a template section. You can specify the following arguments to these tags:

- Name by which a template tag is to be invoked
- Default heading for the tag when it is invoked
- Name of tags defined within the context of the template tag

For example, if you specify "MYLIST" as the first argument to the `<SET_TEMPLATE_LIST>` tag, you would define a list that is begun by the `<MYLIST>` tag and terminated by the `<ENDMYLIST>` tag. If you specify "My List" as the second argument to the `<SET_TEMPLATE_LIST>` tag, you would define a default heading for this list of "My List." And if you specify "MY_ITEM" as the third argument to the `<SET_TEMPLATE_LIST>` tag, you would define the tag `<MY_ITEM>` as a tag to be used within the context of the `<MYLIST>` tag.

Each of the tags defined using these tags accepts an alternate heading argument and the NONE keyword argument. Note that these user-defined template tags have the same behavior as standard template tags. If the NONE keyword argument is specified, the terminating tag must not be used.

8.3.1 Creating Your Own Template Lists

The `<SET_TEMPLATE_LIST>` tag lets you create your own user-defined set of tags for listing information. This tag requires four arguments and accepts an optional fifth argument. This tag has the following syntax.

```
<SET_TEMPLATE_LIST> (list-tag-name
                     \ default-list-heading
                     \ list-item-tag-name
                     \ list-type
                     [ \ heading-level ] )
```

Using the SOFTWARE Doctype

The following list summarizes the arguments allowed by this tag in the order in which they must be specified:

list-tag-name

Specifies the name of the tag that will begin the user-defined list. This name is also used for the terminating tag name. For example, if the *list-tag-name* argument is specified as "SAMPLE_LISTTAG," the beginning tag will be `<SAMPLE_LISTTAG>` and the terminating tag will be `<ENDSAMPLE_LISTTAG>`.

The user-defined tag created by this argument is similar to the global `<LIST>` tag.

default-list-heading

Specifies the default text heading to be output by the user-defined list. This heading can be overridden by an alternate heading specified as an argument to the `<list-tag-name>` tag.

list-item-tag-name

Specifies the name of the tag to be used to indicate individual items in the list being defined. For example, if the *list-item-tag-name* argument is specified as "SAMP_ITEM," the individual list item tag will be `<SAMP_ITEM>`.

The user-defined tag created by this argument is similar to the global `<LE>` tag.

list-type

Specifies the keyword that indicates the type of list on which the user-defined list is based. There are three available keywords:

SIMPLE
NUMBERED
UNNUMBERED

These keywords have the same effect as the keywords of the same name used with the global `<LIST>` tag.

heading-level

Specifies the heading level to be associated with the tag. The accepted arguments are as follows:

- 1 — Specifies a primary template heading level
- 2 — Specifies a secondary template heading level

If this argument is not specified, the value defaults to 2.

Note that the case of the text output by the template heading is controlled by the doctype under which the SDML file is processed.

The first line of the first example shows how the user-defined `<SHOPPING_LIST>` tag is defined using the `<SET_TEMPLATE_LIST>` tag. The following examples show various uses of the `<SHOPPING_LIST>` tag.

The first example shows the `<SHOPPING_LIST>` tag defined with a heading value of 1 and the second example shows the same tag defined with the default heading value of 2. Note how this change affects the printed output.

```
<COMMAND_SECTION>
.
<SET_TEMPLATE_LIST>(SHOPPING_LIST\Shopping List\SHOP_ITEM\NUMBERED\1)
<SHOPPING_LIST>
<SHOP_ITEM>Hair gel
<SHOP_ITEM>Beer nuts
<SHOP_ITEM>Frisbee
<ENDSHOPPING_LIST>
```

This example may produce the following output:

SHOPPING LIST	1 Hair gel
	2 Beer nuts
	3 Frisbee

The following example shows how an alternate heading may be specified for a user-defined list. It also shows how the NONE keyword argument may be specified. Note that both of these arguments are optional.

```
<COMMAND_SECTION>
.
<SET_TEMPLATE_LIST>(SHOPPING_LIST\Shopping List\SHOP_ITEM\NUMBERED)
<SHOPPING_LIST>(NONE)
<SHOPPING_LIST>(Bobs Wood Goods)
<SHOP_ITEM>Wooden Tables
<SHOP_ITEM>Wooden chairs
<ENDSHOPPING_LIST>
<SHOPPING_LIST>(Bloomingdales\NONE)
```

This example may produce the following output:

shopping list	<i>None.</i>
bobs wood goods	1 Wooden Tables 2 Wooden chairs
bloomingdales	<i>None.</i>

8.3.2 Creating Your Own Template Sections

The `<SET_TEMPLATE_PARA>` tag lets you create your own user-defined set of tags for making a template section. This tag requires two arguments and accepts an optional third argument. This tag has the following syntax.

```
<SET_TEMPLATE_PARA> (para-tag-name
                     |default-para-heading
                     [heading-level])
```

The following list summarizes the arguments allowed by this tag in the order in which they must be specified:

Using the SOFTWARE Doctype

para-tag-name

Specifies the name of the tag that will begin the user-defined paragraph section. This name is also used for the terminating tag name. For example, if the *para-tag-name* argument is specified as "SAMPLE_PARATAG," the beginning tag will be `<SAMPLE_PARATAG>` and the terminating tag will be `<ENDSAMPLE_PARATAG>`.

default-para-heading

Specifies the default text heading to be output by the user-defined paragraph section. This heading can be overridden by an alternate heading specified as an argument to the `<para-tag-name>` tag.

heading-level

Specifies the template heading level to be associated with the tag. The accepted arguments are as follows:

- 1 — Specifies a primary template heading level.
- 2 — Specifies a secondary template heading level.

If this argument is not specified, the value defaults to 2, and the secondary template heading is used.

Note that the case of the text output by the template heading is controlled by the doctype. For example, in some doctypes specifying the value 1 might result in the heading being output in all uppercase text, and specifying the value 2 might result in the heading being output in all lowercase text.

The following examples show how to create a template tag called `<SIDE_EFFECTS>` using the `<SET_TEMPLATE_PARA>` tag. The first example uses the `<SET_TEMPLATE_PARA>` tag to define the tag `<SIDE_EFFECTS>` as the beginning tag of a paragraph template that has a default heading of "Side Effects:." Note that this first definition uses the a heading level value of 1 for the `<SIDE_EFFECTS>` tag. Note how the second `<SIDE_EFFECTS>` tag example explicitly specifies a default heading level value of 2.

```
<COMMAND_SECTION>
<SET_TEMPLATE_PARA>(SIDE_EFFECTS\Side Effects:\1)
<SIDE_EFFECTS>
Modifying the arguments to the PLACE command changes the positioning of the
page number.
<ENDSIDE_EFFECTS>
```

This example may produce the following output:

SIDE EFFECTS:

Modifying the arguments to the PLACE command changes the positioning of the page number.

The following example illustrates how to specify the template tag `<SIDE_EFFECTS>` with an alternate heading of "desired effect=" and shows how the NONE keyword argument can be used.

Note how the alternate heading "desired effect=" overrides the default heading specified by the `<SET_TEMPLATE_PARA>` tag for the current tag. Note also how this definition of the `<SIDE_EFFECTS>` tag has a heading level of 2. Note how the second `<SIDE_EFFECTS>` tag uses the default heading.

```

<COMMAND_SECTION>
.
.
<SET_TEMPLATE_PARA>(SIDE_EFFECTS\side effects:\2)
<side_effects>(desired effect=)
Modifying the arguments to the PLACE command changes the positioning of the
page number.
<endside_effects>
<side_effects>(NONE)
<ENDCOMMAND_SECTION>

```

This example may produce the following output:

desired effect= Modifying the arguments to the PLACE command changes the positioning of the page number.

side effects: *None.*

8.3.3 Creating Your Own Template Tables

The `<SET_TEMPLATE_TABLE>` tag lets you create your own user-defined set of tags for making a table with optional headings within a template. Tables created using this tag can have either two or three columns. This tag requires five arguments and accepts the optional *table-column-headings* argument. This tag has the following syntax:

```

<SET_TEMPLATE_TABLE> (table-tag-name
                      \ default-table-heading
                      \ table-row-tag-name
                      \ column-count
                      \ column-widths
                      [ \ table-column-headings ])

```

The following list summarizes the arguments allowed by this tag in the order in which they must be specified:

table-tag-name

Specifies the user-defined name of the tag that begins the user-defined table.

default-table-heading

Specifies the default text heading to be output over the entire user-defined table. This heading can be overridden by an alternate heading specified as an argument to the `<table-tag-name>` tag.

table-row-tag-name

Specifies the name of the tag to be used to indicate individual table rows in the table being defined. For example, if the *table-row-tag-name* argument is specified as "SAMP_ROW," the individual table row tag will be `<SAMP_ROW>`.

The user-defined tag created by this argument is similar to the global `<TABLE_ROW>` tag.

Using the SOFTWARE Doctype

column-count

Specifies the number of columns in the user-defined table. The accepted arguments are as follows:

- 2 — Specifies that the table is to have two columns.
- 3 — Specifies that the table is to have three columns.

table-column-widths

Specifies the approximate widths of the table columns. The width of the last table column is determined by VAX DOCUMENT. So, if you specify a two-column list, you must specify only one column-width argument as shown in the following code example.

```
<SET_TEMPLATE_TABLE>(KEYVALS\Keyword Values\KEYVAL\2\10\Keyword\Value)
```

If you specify a three-column list, you must specify two column-width arguments as shown in the following code example.

```
<SET_TEMPLATE_TABLE>(KEYVAL_LIST\Keyword Ranges\KEYVAL\3\10\10\Keyword\High\Lower)
```

table-column-headings

Specifies optional default headings for each column in the user-defined table. If you specified a two-column list, you may specify up to two heading arguments. If you specified a three-column list, you may specify up to three heading arguments.

The following examples show a definition of the user-defined `<RECORDTABLE>` table tags. In this example, the `<RECORDTABLE>` tag is defined as a two-column list with column headings, and with a default table heading of "Best Songs."

The first use of these table tags in the following example sets the text supplied to the two `<45RPM>` tags in the table, and then terminates the table. The second use shows how the `NONE` keyword argument can be used.

```
<SET_TEMPLATE_TABLE>(RECORDTABLE\Best Songs\45RPM\2\12\Performer\Song Title)
<RECORDTABLE>
<45RPM>(Sinatra\Strangers in the Night)
<45RPM>(Moody Blues\Nights in White Satin)
<ENDRECORDTABLE>
<RECORDTABLE>(NONE)
```

This example may produce the following output:

best songs

Performer	Song Title
Sinatra	Strangers in the Night
Moody Blues	Nights in White Satin

best songs

None.

8.4 Modifying the Reference Templates

In most cases, you will not find it necessary to modify the reference templates. However, if you do wish to modify these formats, you can do so by using one of the following tags:

- `<SET_TEMPLATE_HEADING>` tag
Creates new default headings for tags used within the reference templates.
- Template-enabling tags
Modifies the format of the entire template. This includes creating running headings, creating page number prefixes, and setting whether the reference template (and any reference-elements within it) begins on a new page of output.
- `<SET_TEMPLATE_templatename>` tags
Modifies the format of the reference-element tags used in each template. This includes specifying the current reference-element name as a secondary running heading, specifying that the reference-element heading has additional information stacked beneath it, and specifying whether the reference-element tag should begin on a new page of output.

The modifications made by these tags only affect those tags that follow the modifying tag in the SDML file in the current reference template.

8.4.1 Modifying Default Headings Within a Template

Each reference template enables several tags that place default headings on the output page. In most cases you will not want to change these default headings because of the possibility of introducing typographical errors into the text of the heading. Changing the default headings can also cause your new headings to be incompatible with other headings in the current template or with headings in other reference sections.

However, if you do wish to modify these headings, you can do so by using the `<SET_TEMPLATE_HEADING>` tag, which lets you specify a new default heading for a template tag within one of the reference templates. This tag has the following syntax:

`<SET_TEMPLATE_HEADING>` (*element-keyword*\(*default-heading*)

The `<SET_TEMPLATE_HEADING>` tag accepts two arguments: the name of the template tag to receive the new default heading, and the text of the new default heading. The new default heading will then be used by all subsequent uses of the template tag named as an argument to the `<SET_TEMPLATE_HEADING>` tag in the current template. This heading overrides any previously defined default heading for that template tag in the current template.

A default heading created using the `<SET_TEMPLATE_HEADING>` tag will be in effect until that heading is reset within the current template by another `<SET_TEMPLATE_HEADING>` tag. Note also that the `<SET_TEMPLATE_HEADING>` tag has no effect on templates other than the one in which it is used.

Using the SOFTWARE Doctype

The following example shows how you would create a new default heading. In this example, the default heading for the <RSDEFLIST> tag is set to be "Conditions Signalled."

```
<ROUTINE_SECTION>
<SET_TEMPLATE_HEADING>(RSDEFLIST\Conditions Signalled)
<RSDEFLIST>
<RSITEM>(SS$_NORMAL\Service successfully completed.)
<RSITEM>(SS$_ACCVIO\Access violation.)
<ENDRSDEFLIST>
```

This example may produce the following output:

CONDITIONS SIGNALLED

```
SS$_NORMAL          Service successfully completed.
SS$_ACCVIO          Access violation.
```

Table 8-1 summarizes the default headings assigned to the standard template tags.

Table 8-1 Default Headings of Reference Template Tags

Template Tag	Default Heading
Command Template Tags	
<FORMAT>	Format
<PARAMDEFLIST>	Parameters
<QUALDEFLIST>	Qualifiers
<DESCRIPTION>	Description
<RESTRICTIONS>	Restrictions
<PROMPTS>	Prompts
Routine Template Tags	
<FORMAT>	Format
<ARGDEFLIST>	Arguments
<DESCRIPTION>	Description
<RSDEFLIST>	Return Values
Tag Template	
<FORMAT>	Format
<PARAMDEFLIST>	Arguments
<RELATED_TAGS>	Related Tags
<TERMINATING_TAG>	Required Terminator
<DESCRIPTION>	Description
Statement Template	
<STATEMENT_FORMAT>	Format
<DESCRIPTION>	Description

8.4.2 Using the Template-enabling Tags

The template-enabling tags are used to begin a reference template, and may optionally be used to alter the default format of that template. The following table lists the template-enabling tags for each template.

Template Name	Template-enabling Tag
Command Template	<COMMAND_SECTION>
Routine Template	<ROUTINE_SECTION>
Statement Template	<STATEMENT_SECTION>
Tag Template	<TAG_SECTION>

All the template-enabling tags have the same syntax and perform the same functions. The syntax for the Command template form of the template-enabling tags is as follows:

```
<COMMAND_SECTION>[[([running-title] [\number-prefix] [\NEWPAGE]]]
.
.
.
<ENDCOMMAND_SECTION>
```

Arguments to the template-enabling tags are optional. However, if arguments are used, they must be specified in the order shown in the previous syntax description.

If you decide to modify a template using the template-enabling tag, these modifications will be in effect only within that particular template. The three arguments accepted by the template-enabling tags are given in the following list.

running-title

Sets the second running heading for the page to be the text specified as the *running-title* argument. This argument is valid only when double running heads are being used in the template.

number-prefix

Sets the numbering prefix to be used to construct page numbers and formal figure, table, and example numbers. Such numbers might be DCL-12, STAT-5, or Table STAT-3.

NEWPAGE

Causes the initial text for the reference section to start on a new page. This argument also causes any template reference-element (such as the <COMMAND> tag), to begin on a new page.

When you use a reference template, you typically use it in one of the following contexts:

- In a part begun using the global <PART> tag within a large document. Generally, the part follows one or more chapters that are numbered using the chapter number as the prefix for page numbers and for formal figure, table, and example numbers.
- In a chapter begun using the global <CHAPTER> tag in a book that has chapter-oriented page numbers.

Using the SOFTWARE Doctype

- In an appendix begun using the global `<APPENDIX>` tag that contains reference information that is intended to stand alone, that is, can be pulled out of the book in which it appears and placed in a binder with other system commands or routines.

In any of these situations you can use multiple reference templates, so long as each is terminated before the next reference template begins.

Note: VAX DOCUMENT does not allow you to nest reference templates inside one another and will signal an error if this occurs.

8.4.3 Using the `<SET_TEMPLATE_templatename>` Tags

The `<SET_TEMPLATE_templatename>` tag is not a VAX DOCUMENT tag. It is a generic name given to a set of four tags that have the same syntax and perform the same functions. These tags differ only in their names and in the templates in which they are available:

Command Template	<code><SET_TEMPLATE_COMMAND></code>
Routine Template	<code><SET_TEMPLATE_ROUTINE></code>
Statement Template	<code><SET_TEMPLATE_STATEMENT></code>
Tag Template	<code><SET_TEMPLATE_TAG></code>

These tags have the following syntax (illustrated by the `<SET_TEMPLATE_COMMAND>` tag in this case):

```
<SET_TEMPLATE_COMMAND> (tag-name[\attribute]
                        [\attribute][\attribute])
```

These tags let you override the default formatting attributes for reference-elements used in each of the reference templates. These tags require a first argument that is the name of the reference-element tag in the template. You can specify the name of the default reference-element tag (such as the `<ROUTINE>` tag in the Routine template), or you can specify a new tag name (such as `<MY_ROUTINE>`).

If you specify a new reference-element tag name, the old tag name is no longer valid and should not be used. This new tag name replaces the previous reference-element tag. All subsequent uses of a reference-element tag should use the last established tag name.

You can then specify one or more of the following keywords that are accepted by these tags (note that these keyword arguments must be separated by backslashes):

NONENPAGE

Specifies that reference descriptions are not to start on new pages. By default, the tag defined by the first argument of the `<SET_TEMPLATE_templatename>` begins a command description on a new page.

DOUBLERUNNINGHEADS

Specifies that the command descriptions have two running titles at the top of every page. The top running title will be set by the `<COMMAND_SECTION>` tag or by heading of the most recent `<CHAPTER>` tag. By default, if a doctype design option does not call for running top titles, only the current command name is placed at the top of each page.

STACK

Specifies that when multiple arguments are specified for the tag defined by the first argument to the `<SET_TEMPLATE_templatename>` tag, the arguments should be stacked at the beginning of the page.

By default, when multiple arguments are specified, the second and third arguments are assumed to be optional descriptive information and are output on the same line as the command name.

The following code example shows a sample use of the `<SET_TEMPLATE_COMMAND>` tag. In this example, the tag `<XYZ_COMMAND>` is defined and used. The keyword argument `NONEWPAGE` is specified to the `<COMMAND_SECTION>` tag so that the first description will not start on a new page. The keyword argument `STACK` is specified so that the two arguments "FIND_FIRST" and "FF" are stacked with the first argument on top.

```
<COMMAND_SECTION>(XYZ Commands\\NEWPAGE)
<SET_TEMPLATE_COMMAND>(XYZ_COMMAND\\NONEWPAGE\\STACK)
<XYZ_COMMAND>(FIND_FIRST\\FF)
```

This example may produce the following output:

FIND_FIRST
FF

8.5 Using the Command Template

Table 8-2 summarizes the tags available in the Command template, the default headings associated with them, and how they should be used. Such a table can be found at the beginning of each of the template reference sections. Chapter 14 contains the reference information on the tags listed in this table.

Also included in this section are two subsections, which provide a sample use of the Command template. You may find these sample files useful in understanding how the Command template tags fit together into a whole.

- Section 8.5.1 contains an SDML file listing of a sample use of the Command template tags.
- Section 8.5.2 contains a sample output file created using the SDML code listed in Section 8.5.1

These samples describe the APPEND command. These two sections are intended only as sample files and should not be used as a source of reference for this command.

Using the SOFTWARE Doctype

Table 8–2 Tags Available in the Command Template in their Standard Order

Tag Name	Default Heading	Template Usage
<COMMAND_SECTION>	None	Begins a Command Section
<SET_TEMPLATE_COMMAND>	None	Alters the default format of a Command section
<SET_TEMPLATE_HEADING>	None	Alters the default headings in a reference section
<SET_TEMPLATE_PARA>	None	Creates user-defined tags for the creation of a specially-formatted paragraph in a reference section
<SET_TEMPLATE_LIST>	None	Creates user-defined tags for the creation of a specially-formatted list in a reference section
<SET_TEMPLATE_TABLE>	None	Creates user-defined tags for the creation of a specially-formatted table in a reference section
<COMMAND>	None	Begins each new element to be referenced in the template; this is the default reference-element tag in the Command reference template
<OVERVIEW>	None	Labels an overview of the reference-element
<FORMAT>	"Format"	Labels the format of the reference-element's syntax
<PARAMDEFLIST>	"Parameters"	Begins a definition list of the parameters or arguments associated with the reference-element
<RESTRICTIONS>	"Restrictions"	Begins a list of zero or more restrictions on the reference-element's use
<PROMPTS>	"Prompts"	Begins a list of the prompts associated with the reference-element
<DESCRIPTION>	"Description"	Labels a reference-element description section
<QUALDEFLIST>	"Qualifiers"	Begins a definition list of zero or more qualifiers associated with the reference-element
<EXAMPLE_SEQUENCE>	"Examples"	Begins a sequence of one or more examples
<SUBCOMMAND_SECTION>	None	Begins a section of subcommands within a Command section
<SUBCOMMAND_SECTION_HEAD>	None	Specifies the heading for introductory text that precedes a subcommand section
<SET_TEMPLATE_SUBCOMMAND>	None	Alters the name of the <SUBCOMMAND> tag, and optionally lets you specify that each use of that tag should not begin output on a new page
<SUBCOMMAND>	None	Begins a new subcommand element to be described within a subcommand section

8.5.1 Sample SDML File of the Command template

The following is an extended code example showing a VAX DOCUMENT SDML file that uses the Command template.

```

<COMMAND_SECTION>(Using the SOFTWARE Doctype\\NEWPAGE)
<SET_TEMPLATE_COMMAND>(DCL_COMMAND)

<DCL_COMMAND>(APPEND)

<OVERVIEW>
Adds the contents of one or more specified input files to the end of the
specified output file.
<ENDOVERRIDE>

<FORMAT>
<fCMD>(APPEND) <FPARMS>(input-file-spec[,<hellipsis>] output-file-spec)

<QUAL_LIST>(Command Qualifiers)
<QPAIR>( /BACKUP /CREATED)
<QPAIR>( /BEFORE [=time] /BEFORE= TODAY)
<ENDQUAL_LIST>

<QUAL_LIST>(Positional Qualifiers)
<QPAIR>( /ALLOCATION = n \See text.)
<QPAIR>( / [NO] CONTIGUOUS \None.)
<ENDQUAL_LIST>
<ENDFORMAT>

<RESTRICTIONS>(NONE)

<PROMPTS>
<PROMPT>(From: \input-file-spec[,<hellipsis>])
<PROMPT>(To: \output-file-spec)
<ENDPROMPTS>
<PARAMDEFLIST>
<PARAMITEM>(input-file-spec[,<hellipsis>])
<PARAMDEF>Specifies the names of one or more input files to be appended.
<P>
If you specify more than one input file, separate the specifications with
either commas (,) or plus signs (+).
Commas and plus signs are equivalent. All input files
are appended, in the order specified, to the end of the output file.
<P>
You can use wildcard characters in the file specification(s).

<PARAMITEM>(output-file-spec)
<PARAMDEF>
Specifies the name of the file to which the input files will be appended.
<P>
You must include at least one field in the output file specification. If you
do not specify a device and/or directory, the APPEND command uses the current
default device and directory. For other fields that you do not specify, the
APPEND command uses the corresponding field of the input file specification.
<P>
If you use the asterisk wildcard character in any field(s) of the
output file specification, the APPEND command uses the corresponding field of
the input file specification. If you are appending more than one
input file, APPEND uses the corresponding fields from the first input file.
<ENDPARAMDEFLIST>

<DESCRIPTION>
The APPEND command is similar in syntax and function to the COPY command.
Normally, the APPEND command adds the contents of one or more files
to the end of an existing file without incrementing the version number.
The /NEW_VERSION qualifier causes the APPEND command to create a new output
file if no file with that name exists.
<ENDDescription>

```

Using the SOFTWARE Doctype

```
<QUALDEFLIST>(Command Qualifiers)
<QUALITEM>( /BACKUP)
<QUALDEF>
Selects files according to the dates of their most recent backup.
This qualifier is relevant only
when used with the /BEFORE or /SINCE qualifier. Use of the
/BACKUP qualifier is incompatible with /CREATED, /EXPIRED, and /MODIFIED.
The default is /CREATED.
<QUALITEM>( /BEFORE[=time])
<QUALDEF>
Selects only those files that are dated before the specified time.
<P>
You can specify
either an absolute time or a combination of absolute and delta times.
See Section <reference>(time_sec) for complete information on
specifying time values.
You can also use the
keywords TODAY, TOMORROW, and YESTERDAY. If no time is specified,
TODAY is assumed.
<ENDQUALDEFLIST>
<QUALDEFLIST>(Positional Qualifiers)
<QUALITEM>( /ALLOCATION=n\ )
<QUALDEF>
Forces the initial allocation of the output file to the number of 512-byte
blocks specified as n.
<P>
This qualifier is valid in conjunction with the /NEW_VERSION qualifier.
The allocation size is
applied only if a new file is actually created. If a new file is created and
you do not specify /ALLOCATION, the initial allocation of the output file is
determined by the size of the input file(s).
<QUALITEM>( /CONTIGUOUS\ /NOCONTIGUOUS)
<QUALDEF>
Indicates whether the output file is contiguous, that is, whether the file
must occupy consecutive physical disk blocks.
<P>
By default, the APPEND command creates an output file in the same format as
the corresponding input file. If an input file is contiguous, the APPEND
command attempts to create a contiguous output file, does not report an
error if there is not enough space. If you append multiple input files of
different formats, the output file might or might not be contiguous. Use the
/CONTIGUOUS qualifier to ensure that the output file is contiguous.
<ENDQUALDEFLIST>

<EXAMPLE_SEQUENCE>
<EXI><S>($ ) <U>(APPEND TEST.DAT NEWTEST.DAT)
<EXTTEXT>
The APPEND command appends the contents of the file TEST.DAT from the default
disk and directory to the file NEWTEST.DAT also located on the default disk
and directory.
<EXI>(WIDE) <S>($ ) <U>(APPEND /NEW_VERSION /LOG *.TXT T.SUM)
<S>(%APPEND-I-CREATED, D1$:[MAL]T.SUM;1 created)
<S>(%APPEND-S-COPIED, D1$:[MAL]A.TXT;2 copied to D1$:[MAL]T.SUM;1 (1 block))
<S>(%APPEND-S-APPENDED, D1$:[MAL]B.TXT;3 appended to D1$:[MAL]T.SUM;1 (3 records))
<S>(%APPEND-S-APPENDED, D1$:[MAL]G.TXT;7 appended to D1$:[MAL]T.SUM;1 (51 records))
<S>(%APPEND-S-NEWFILES, 1 file created)
<EXTTEXT>
The APPEND command appends all files with file types of TXT to a file named
T.SUM. The /LOG qualifier requests a display of the specifications of each
input file appended. If the file T.SUM does not exist, the APPEND command
creates it, as the output shows. The number of blocks or records shown in the
output refers to the SDML file and not to the target file total.
<ENDEXAMPLE_SEQUENCE>
<ENDCOMMAND_SECTION>
```

8.5.2 Sample Output File of the Command Template

The following is the output from the extended code example in Section 8.5.1. Note that your own output may vary, depending on the SOFTWARE design under which you process the SDML file.

APPEND

Adds the contents of one or more specified input files to the end of the specified output file.

FORMAT **APPEND** *input-file-spec*[, . . .] *output-file-spec*

Command Qualifiers	Defaults
<i>/BACKUP</i> <i>/BEFORE</i> [= <i>time</i>]	<i>/CREATED</i> <i>/BEFORE</i> = <i>TODAY</i>

Positional Qualifiers	Defaults
<i>/ALLOCATION</i> = <i>n</i> <i>/[NO]CONTIGUOUS</i>	<i>See text.</i> <i>None.</i>

restrictions *None.*

prompts From: *input-file-spec*[, . . .]
 To: *output-file-spec*

PARAMETERS *input-file-spec*[, . . .]

Specifies the names of one or more input files to be appended.

If you specify more than one input file, separate the specifications with either commas (,) or plus signs (+). Commas and plus signs are equivalent. All input files are appended, in the order specified, to the end of the output file.

You can use wildcard characters in the file specification(s).

output-file-spec

Specifies the name of the file to which the input files will be appended.

You must include at least one field in the output file specification. If you do not specify a device and/or directory, the APPEND command uses the current default device and directory. For other fields that you do not specify, the APPEND command uses the corresponding field of the input file specification.

If you use the asterisk wildcard character in any field(s) of the output file specification, the APPEND command uses the corresponding field of the input file specification. If you are appending more than one input file, APPEND uses the corresponding fields from the first input file.

DESCRIPTION The APPEND command is similar in syntax and function to the COPY command. Normally, the APPEND command adds the contents of one or more files to the end of an existing file without incrementing the version number. The */NEW_VERSION* qualifier causes the APPEND command to create a new output file if no file with that name exists.

**COMMAND
QUALIFIERS*****/BACKUP***

Selects files according to the dates of their most recent backup. This qualifier is relevant only when used with the */BEFORE* or */SINCE* qualifier. Use of the */BACKUP* qualifier is incompatible with */CREATED*, */EXPIRED*, and */MODIFIED*. The default is */CREATED*.

/BEFORE[=time]

Selects only those files that are dated before the specified time.

You can specify either an absolute time or a combination of absolute and delta times. See Section 1.2 for complete information on specifying time values. You can also use the keywords *TODAY*, *TOMORROW*, and *YESTERDAY*. If no time is specified, *TODAY* is assumed.

**POSITIONAL
QUALIFIERS*****/ALLOCATION=n***

Forces the initial allocation of the output file to the number of 512-byte blocks specified as *n*.

This qualifier is valid in conjunction with the */NEW_VERSION* qualifier. The allocation size is applied only if a new file is actually created. If a new file is created and you do not specify */ALLOCATION*, the initial allocation of the output file is determined by the size of the input file(s).

/CONTIGUOUS***/NOCONTIGUOUS***

Indicates whether the output file is contiguous, that is, whether the file must occupy consecutive physical disk blocks.

By default, the *APPEND* command creates an output file in the same format as the corresponding input file. If an input file is contiguous, the *APPEND* command attempts to create a contiguous output file but does not report an error if there is not enough space. If you append multiple input files of different formats, the output file might or might not be contiguous. Use the */CONTIGUOUS* qualifier to ensure that the output file is contiguous.

EXAMPLES

1 \$ APPEND TEST.DAT NEWTEST.DAT

The *APPEND* command appends the contents of the file *TEST.DAT* from the default disk and directory to the file *NEWTTEST.DAT* also located on the default disk and directory.

2 \$ APPEND/NEW_VERSION/LOG *.TXT T.SUM
 %APPEND-I-CREATED, D1\$:[MAL]T.SUM;1 created
 %APPEND-S-COPIED, D1\$:[MAL]A.TXT;2 copied to D1\$:[MAL]T.SUM;1 (1 block)
 %APPEND-S-APPENDED, D1\$:[MAL]B.TXT;3 appended to D1\$:[MAL]T.SUM;1 (3 records)
 %APPEND-S-APPENDED, D1\$:[MAL]G.TXT;7 appended to D1\$:[MAL]T.SUM;1 (51 records)
 %APPEND-S-NEWFILES, 1 file created

The *APPEND* command appends all files with file types of *TXT* to a file named *T.SUM*. The */LOG* qualifier requests a display of the specifications of each input file appended. If the file *T.SUM* does not exist, the *APPEND* command creates it, as the output shows. The number of blocks or records shown in the output refers to the *SDML* file and not to the target file total.

8.6 Using the Routine Template

Table 8–3 summarizes the tags available in the Routine template, the default headings associated with them, and how they should be used. Such a table can be found at the beginning of each of the template reference sections. Chapter 14 contains the reference information on the tags listed in this table.

Also included in this section are two subsections, which provide a sample use of the Routine template. You may find these sample files useful in understanding how the Routine template tags fit together into a whole.

- Section 8.6.1 contains a SDML file listing of a sample use of the Routine template tags.
- Section 8.6.2 contains a sample output file created using the SDML code listed in Section 8.6.1

These samples describe the \$ENQ and MTH\$xSORT routines. These two subsections are intended only as sample files and should not be used as a source of reference for these routines.

Table 8–3 Tags Available in the Routine Template in their Standard Order

Tag Name	Default Heading	Template Usage
<ROUTINE_SECTION>	None	Begins a Routine section
<SET_TEMPLATE_ROUTINE>	None	Alters the default format of a Routine section
<SET_TEMPLATE_HEADING>	None	Alters the default headings in a reference section
<SET_TEMPLATE_PARA>	None	Creates user-defined tags for the creation of a specially-formatted paragraph in a reference section
<SET_TEMPLATE_LIST>	None	Creates user-defined tags for the creation of a specially-formatted list in a reference section
<SET_TEMPLATE_TABLE>	None	Creates user-defined tags for the creation of a specially-formatted table in a reference section
<ROUTINE>	None	Begins each new element to be referenced in the template; this is the default reference-element tag in the Routine reference template
<OVERVIEW>	None	Labels an overview of the reference-element
<FORMAT>	"Format"	Labels the format of the reference-element's syntax
<RETURNS>	"Returns"	Provides specific information about the attributes of the value returned by the routine
<RETTEXT>	None	Provides general information about the attributes of the value returned by the routine
<ARGDEFLIST>	"Arguments"	Begins a definition list of the arguments associated with the reference-element
<DESCRIPTION>	"Description"	Labels a reference-element description section
<RSDEFLIST>	"Return Values"	Begins a definition list of zero or more routine return status codes and their meanings
<EXAMPLE_SEQUENCE>	"Examples"	Begins a sequence of one or more examples

8.6.1 Sample SDML File of the Routine Template

The following is an extended code example showing a VAX DOCUMENT SDML file that uses the Routine template.

```

<routine_section>(Using the SOFTWARE Doctype\\NEWPAGE)
<set_template_routine>(VMS_ROUTINE\doublerunningheads)
<VMS_ROUTINE>($ENQ\Enqueue Resource)
<OVERVIEW>This is the brief description section. It contains one or
two sentences of description of what the routine does.
<ENDOVERVIEW>
<FORMAT>
<frtn>($ENQ) <fargs>([efn], lkmode, lksb, itmlst)
<ENDFORMAT>
<RETURNS>(cond_value\longword integer\read only\by value in R0)
<RETTEXT>At times, it may be necessary to include a sentence or two
here to further describe the nature of the information returned.
<ENDRETTEXT>
<ARGDEFLIST>
<ARGITEM>(efn\cond_value\longword integer\read only\by value)
<ARGDEF>Number of the event flag that is to be set when access is
granted to the specified resource. If not specified, the default is
event flag number 0.
<ARGITEM>(lkmode\cond_value\longword integer\read only\by descriptor\varying string
array descriptor)
<ARGDEF>Name of lock mode requested. May be one of the following:
<TABLE>
<TABLE_SETUP>(2\17)
<TABLE_HEADS>(Name of Lock Mode\Description)
<TABLE_ROW>(LCK$K_NLMODE\Null lock mode)
<TABLE_ROW>(LCK$K_CRMODE\Concurrent read mode)
<TABLE_ROW>(LCK$K_CWMODE\Concurrent write mode)
<ENDTABLE>
<ARGITEM>(lksb\cond_value\longword integer\write only\by value)
<ARGDEF>Address of the lock status block. The lock status block
receives the final completion status and lock I.D., and optionally
contains a lock value block.
<ARGITEM>(itmlst)
<ARGDEF>
Item list specifying the lock information that $GETLKI is to return. The
<variable>(itmlst) argument is the address of a list of item descriptors, each
of which describes an item of information. The list of item descriptors is
terminated by a longword of 0.
<line_art>
      31                15                0
      +-----+-----+-----+
      |   item code   |   buffer length   |
      +-----+-----+-----+
      |           buffer address           |
      +-----+-----+-----+
      |           return length address    |
      +-----+-----+-----+
<endline_art>
<endargdeflist>
<DESCRIPTION>
This section contains the full, detailed description of the routine.
It may contain tables and figures. There is no fixed size for this
description section.
<ENDDescription>
<RSDEFLIST>
<RSITEM>(SS$ _NORMAL\Indicates successful completion)
<RSITEM>(SS$ _ABORT\This description contains a full explanation of some of
the possible causes for the abortion)
<RSITEM>(SS$ _DEADLOCK\This description contains a full explanation of
some possible causes for the deadlock situation.)
<ENDRSDEFLIST>

```

Using the SOFTWARE Doctype

```
<EXAMPLE_SEQUENCE>
<examples_intro>
This section contains an example of the use of the routine.
This section can also contain figures and tables.
<ENDEXAMPLE_SEQUENCE>

<VMS_ROUTINE>(MTH$xSQRT)
<OVERVIEW>The square root procedure returns the square root of the
input parameter. The input parameter may have one of four data types:
F_Floating, D_Floating, G_Floating, and H_Floating.
<endoverview>
<FORMAT>
<ffunc>(MTH$SQRT\ (x))
<ffunc>(MTH$DSQRT\ (x))
<ffunc>(MTH$GSQRT\ (x))
<ffunc>(MTH$HSQRT\ (x))
<ENDFORMAT>
<RETURNS>(cond_value\F_Floating, D_Floating, or G_Floating
\write only\by value in R0)
<RETURNS>(headonly)
<RETTEXT>The square roots of F_Floating, D_Floating, and G_Floating
input parameters are returned by immediate value in R0 and R1. The
square root of an H_Floating parameter is returned by reference
in the output parameter <VARIABLE>(sqrt).
<ENDRETTEXT>
<ARGDEFLIST>(Argument)
<ARGITEM>(x\cond_value\F_Floating,D_Floating, G_Floating, or H_Floating
\read only\by reference)
<ARGDEF>The number for which the square root is desired.
<ARGITEM>(sqrt\cond_value\H_Floating\write only\by reference)
<ARGDEF>The square root of the H_Floating parameter.
<ENDARGDEFLIST>
<description>
This is a description of the MTH$xSQRT function.
<enddescription>
<endroutine_section>
```

8.6.2 Sample Output File of the Routine Template

The following is the output from the extended code example in Section 8.6.1. Note that your own output may vary, depending on the SOFTWARE design under which you process the SDML file.

\$ENQ

Enqueue Resource

This is the brief description section. It contains one or two sentences of description of what the routine does.

FORMAT **\$ENQ** *[efn], lkmode, lksb, itmlst*

RETURNS VMS Usage: **cond_value**
 type: **longword integer**
 access: **read only**
 mechanism: **by value in R0**

At times, it may be necessary to include a sentence or two here to further describe the nature of the information returned.

ARGUMENTS ***efn***
 VMS Usage: **cond_value**
 type: **longword integer**
 access: **read only**
 mechanism: **by value**

Number of the event flag that is to be set when access is granted to the specified resource. If not specified, the default is event flag number 0.

lkmode
 VMS Usage: **cond_value**
 type: **longword integer**
 access: **read only**
 mechanism: **by descriptor—varying string array descriptor**

Name of lock mode requested. May be one of the following:

Name of Lock Mode	Description
LCK\$_NLMODE	Null lock mode
LCK\$_CRMODE	Concurrent read mode
LCK\$_CWMODE	Concurrent write mode

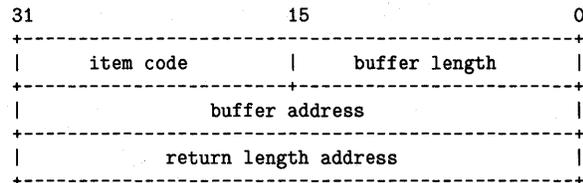
lksb
 VMS Usage: **cond_value**
 type: **longword integer**
 access: **write only**
 mechanism: **by value**

Address of the lock status block. The lock status block receives the final completion status and lock I.D., and optionally contains a lock value block.

Using the SOFTWARE Doctype

itmlst

Item list specifying the lock information that \$GETLKI is to return. The *itmlst* argument is the address of a list of item descriptors, each of which describes an item of information. The list of item descriptors is terminated by a longword of 0.



DESCRIPTION This section contains the full, detailed description of the routine. It may contain tables and figures. There is no fixed size for this description section.

RETURN VALUES

SS\$_NORMAL	Indicates successful completion
SS\$_ABORT	This description contains a full explanation of some of the possible causes for the abortion
SS\$_DEADLOCK	This description contains a full explanation of some possible causes for the deadlock situation.

EXAMPLES

This section contains an example of the use of the routine. This section can also contain figures and tables.

MTH\$xSQRT

The square root procedure returns the square root of the input parameter. The input parameter may have one of four data types: F_Floating, D_Floating, G_Floating, and H_Floating.

FORMAT

MTH\$SQRT(*x*)
MTH\$DSQRT(*x*)
MTH\$GSQRT(*x*)
MTH\$HSQRT(*x*)

RETURNS

VMS Usage: **cond_value**
type: **F_Floating, D_Floating, or G_Floating**
access: **write only**
mechanism: **by value in R0**

RETURNS

The square roots of F_Floating, D_Floating, and G_Floating input parameters are returned by immediate value in R0 and R1. The square root of an H_Floating parameter is returned by reference in the output parameter *sqrt*.

ARGUMENT

x
VMS Usage: **cond_value**
type: **F_Floating, D_Floating, G_Floating, or H_Floating**
access: **read only**
mechanism: **by reference**

The number for which the square root is desired.

sqrt

VMS Usage: **cond_value**
type: **H_Floating**
access: **write only**
mechanism: **by reference**

The square root of the H_Floating parameter.

DESCRIPTION This is a description of the MTH\$xSQRT function.

8.7 Using the Statement Template

Table 8-4 summarizes the tags available in the Statement template, the default headings associated with them, and how they should be used. Such a table can be found at the beginning of each of the template reference sections. Chapter 14 contains the reference information on the tags listed in this table.

Also included in this section are two subsections, which provide a sample use of the Statement template. You may find these sample files useful in understanding how the Statement template tags fit together into a whole.

- Section 8.7.1 contains a SDML file listing of a sample use of the Statement template tags.
- Section 8.7.2 contains a sample output file created using the SDML code listed in Section 8.7.1.

These samples describe the RECORD statement and the MID\$ function. This sample is intended only as an output sample and should not be used as a source of reference for these statements and functions.

Table 8-4 Tags Available in the Statement Template in their Standard Order

Tag Name	Default Heading	Template Usage
<STATEMENT_SECTION>	None	Begins a Command Section
<SET_TEMPLATE_STATEMENT>	None	Alters the default format of a Statement section
<SET_TEMPLATE_HEADING>	None	Alters the default headings in a reference section
<SET_TEMPLATE_PARA>	None	Creates user-defined tags for the creation of a specially-formatted paragraph in a reference section
<SET_TEMPLATE_LIST>	None	Creates user-defined tags for the creation of a specially-formatted list in a reference section
<SET_TEMPLATE_TABLE>	None	Creates user-defined tags for the creation of a specially-formatted table in a reference section
{ <STATEMENT> ¹ } { <FUNCTION> }	None	Begins each new element to be referenced in the template; this is the default reference-element tag in the Statement reference template
<OVERVIEW>	None	Labels an overview of the reference-element
<STATEMENT_FORMAT>	"Format"	Labels the format of the reference-element's syntax
<DESCRIPTION>	"Description"	Labels a reference-element description section
<EXAMPLE_SEQUENCE>	"Examples"	Begins a sequence of one or more examples

¹The use of braces indicates that the enclosed tag names are used interchangeably. VAX DOCUMENT provides two such names to improve the readability of your SDML file.

8.7.1 Sample SDML File of the Statement Template

The following is an extended code example showing a VAX DOCUMENT SDML file that uses the Statement template.

```

<statement_section>(Using the SOFTWARE Doctype\NEWPAGE)
<statement>(RECORD)
<overview>
The RECORD statement lets you name and define data structures in a
BASIC program and provides the BASIC interface to the VAX Common Data
Dictionary (CDD). You can use the defined RECORD name anywhere a
BASIC data-type keyword is valid.
<endoverview>
<statement_format>
<FCMD>(RECORD) <FPARMS>(rec-nam)
<STATEMENT_LINE>(rec-component)
<ellipsis>
<FCMD>(END RECORD) <FPARMS>([ rec-nam ])
<construct_list>(rec-component:)
<construct>(rec-component:)<list>(stacked\braces)
    <le>data-type rec-item [ , [ data-type ] rec-item ]
    <le>group-clause
    <le>variant-clause<endlist>
<construct>(rec-item:)<list>(stacked\braces)
    <le>unsubs-vbl [ = int-const ]
    <le>array ( int-const,...) [ = int-const ]
    <le><keyword>(FILL) [ ( int-const ) ] [ = int-const ]<endlist>
<construct>(group-clause:)
    <keyword>(GROUP) group-nam [ ( int-const,... ) ]
    <statement_line>(rec-component\indent)
    <ellipsis>
    <statement_line>(<keyword>(END GROUP) [ group-nam ])
<construct>(variant-clause:)
    <keyword>(VARIANT)
    <statement_line>(case-clause\indent)
    <ellipsis>
    <statement_line>(<keyword>(END VARIANT))
<construct>(case-clause:)
    <keyword>(CASE)
    <statement_line>([ rec-component ]\indent)
<endconstruct_list>
<endstatement_format>
<function>(MID$)
<overview>
The MID$ function extracts a specified substring from the middle of a
string, leaving the main string unchanged.
<endoverview>
<statement_format>
<fcmd>()<fparms>(str-vbl =<list>(stacked\braces)
    <le>MID
    <le>MID$ <endlist> <keyword>((str-exp, int-exp1, int-exp2)))
<endstatement_format>
<endstatement_section>

```

8.7.2 Sample Output File of the Statement Template

The following is the output from the extended code example in Section 8.7.1. Note that your own output may vary, depending on the SOFTWARE design under which you process the SDML file.

RECORD

The RECORD statement lets you name and define data structures in a BASIC program and provides the BASIC interface to the VAX Common Data Dictionary (CDD). You can use the defined RECORD name anywhere a BASIC data-type keyword is valid.

Format

```
RECORD  rec-nam
         rec-component
```

```
      .
      .
      .
```

```
END RECORD  [ rec-nam ]
```

```
rec-component:  { data-type rec-item [ , [ data-type ] rec-item ]
                  group-clause
                  variant-clause }
```

```
rec-item:      { unsubs-vbl [= int-const ]
                  array ( int-const, . . . ) [= int-const ]
                  FILL [ ( int-const ) ] [= int-const ] }
```

```
group-clause:  GROUP group-nam [ ( int-const, . . . ) ]
                 rec-component
```

```
      .
      .
      .
```

```
END GROUP [ group-nam ]
```

```
variant-clause: VARIANT
                  case-clause
```

```
      .
      .
      .
```

```
END VARIANT
```

```
case-clause:   CASE
                 [ rec-component ]
```

Using the SOFTWARE Doctype

MID\$

The MID\$ function extracts a specified substring from the middle of a string, leaving the main string unchanged.

Format

$$str-vbl = \left\{ \begin{array}{l} MID \\ MID\$ \end{array} \right\} (str-exp, int-exp1, int-exp2)$$

8.8 Using the Tag Template

Table 8–5 summarizes the tags available in the Tag template, the default headings associated with them, and how they should be used. Such a table can be found at the beginning of each of the template reference sections. Chapter 14 contains the reference information on the tags listed in this table.

Also included in this section are two subsections that provide sample uses of the Tag template. You may find these sample files useful in understanding how the Tag template tags fit together into a whole.

- Section 8.8.1 contains a SDML file listing of a sample use of the Tag template tags.
- Section 8.8.2 contains a sample output file created using the SDML code listed in Section 8.8.1

This sample describes the <SYNTAX> tag. This sample is intended only as an output sample and should not be used as a source of reference for this tag.

Table 8–5 Tags Available in the Tag Template in their Standard Order

Tag Name	Default Heading	Template Usage
<TAG_SECTION>	None	Begins a Tag Section
<SET_TEMPLATE_TAG>	None	Alters the default format of a Tag section
<SET_TEMPLATE_HEADING>	None	Alters the default headings in a reference section
<SET_TEMPLATE_PARA>	None	Creates user-defined tags for the creation of a specially-formatted paragraph in a reference section
<SET_TEMPLATE_LIST>	None	Creates user-defined tags for the creation of a specially-formatted list in a reference section
<SET_TEMPLATE_TABLE>	None	Creates user-defined tags for the creation of a specially-formatted table in a reference section
<SDML_TAG>	None	Begins each new element to be referenced in the template; this is the default reference-element tag in the Tag reference template
<OVERVIEW>	None	Labels an overview of the reference-element
<FORMAT>	"Format"	Labels the format of the reference-element's syntax
<PARAMDEFLIST>	"Arguments"	Begins a definition list of the arguments associated with the reference-element
<RELATED_TAGS>	"Related Tags"	Begins a list of zero or more tags related to the tag being described
<TERMINATING_TAG>	"Required Terminator"	Labels the tag that terminates the tag being described
<RESTRICTIONS>	"Restrictions"	Begins a list of zero or more restrictions on the reference-element's use
<DESCRIPTION>	"Description"	Labels a reference-element description section
<EXAMPLE_SEQUENCE>	"Examples"	Begins a sequence of one or more examples

Using the SOFTWARE Doctype

8.8.1 Sample SDML File of the Tag Template

The following is an extended code example showing a VAX DOCUMENT SDML file that uses the Tag template.

```
<TAG_SECTION>(Using the SOFTWARE Doctype\\NEWPAGE)
<SDML_TAG>(SYNTAX)
<OVERVIEW>
Allows you to use special characters to describe language
syntaxes.
<ENDOVERVIEW>

<FORMAT>
<FTAG>(SYNTAX\\<LIST>(STACKED\\braces)
      <LE>heading-text [<ARG_SEP>WIDE]
      <LE>WIDE<ENDLIST>\\OPTIONAL)

<ENDFORMAT>
<PARAMDEFLIST>
  <PARAMITEM>(heading-text)
  <PARAMDEF>Specifies a heading. The doctype controls the font used to
display the heading. By default, this tag has no heading. You may wish to create
a heading using the <TAG>(SYNTAX_DEFAULT_HEAD) tag.
    <PARAMITEM>(WIDE)
    <PARAMDEF>Specifies that the syntax statement may exceed the normal right
margin of the text. If you are using doctype designs that indent
the text body, a wide syntax example will extend into the left margin.
  <ENDPARAMDEFLIST>

<RELATED_TAGS>
<RELATED_TAG>(display)
<RELATED_TAG>(syntax_default_head)
<RELATED_ITEM>The global <TAG>(CODE_EXAMPLE) tag
<RELATED_ITEM>The global <TAG>(FORMAT) tag
<ENDRELATED_TAGS>

<RESTRICTIONS>
You cannot use tab characters,
index tags (such as the <TAG>(x) and <TAG>(Y) tags,
or text element tags (such as <TAG>(p), <TAG>(list), or <TAG>(note))
within this type of example.
<ENDRESTRICTIONS>

<TERMINATING_TAG>(ENDSYNTAX)

<DESCRIPTION>
The <TAG>(SYNTAX) tag allows you to accurately describe language
syntax. Languages can include programming languages, command
languages, application defined languages, and so forth. This tag also
separates the syntax example from the remaining text, retains blank
spaces and open lines, and labels the example (if you specified one)
using a doctype-specific font different from the current text font.
<ENDDescription>

<EXAMPLE_SEQUENCE>(EXAMPLE)
<EXC><LITERAL><P>The COPY command has the following syntax:
<SYNTAX>
      COPY input-file output-file
<ENDSYNTAX>

<ENDLITERAL>
<EXTEXT>
This example may produce the following output:
<P>
The COPY command has the following syntax:
<SYNTAX>
      COPY input-file output-file
<ENDSYNTAX>
<ENDEXAMPLE_SEQUENCE>
```

8.8.2 Sample Output File of the Tag Template

The following is the output from the extended code example in Section 8.8.1. Note that your own output may vary, depending on the SOFTWARE design under which you process the SDML file.

Using the SOFTWARE Doctype

<SYNTAX>

Allows you to use special characters to describe language syntaxes.

FORMAT

<SYNTAX> [({ *heading-text* [\ *WIDE*] })]

ARGUMENTS

heading-text

Specifies a heading. The doctype controls the font used to display the heading. By default, this tag has no heading. You may wish to create a heading using the `<SYNTAX_DEFAULT_HEAD>` tag.

WIDE

Specifies that the syntax statement may exceed the normal right margin of the text. If you are using doctype designs that indent the text body, a wide syntax example will extend into the left margin.

related tags

- `<DISPLAY>`
- `<SYNTAX_DEFAULT_HEAD>`
- The global `<CODE_EXAMPLE>` tag
- The global `<FORMAT>` tag

restrictions

You cannot use tab characters, index tags (such as the `<X>` and `<Y>` tags, or text element tags (such as `<P>`, `<LIST>`, or `<NOTE>`) within this type of example.

required terminator

`<ENDSYNTAX>`

DESCRIPTION

The `<SYNTAX>` tag allows you to accurately describe language syntax. Languages can include programming languages, command languages, application defined languages, and so forth. This tag also separates the syntax example from the remaining text, retains blank spaces and open lines, and labels the example (if you specified one) using a doctype-specific font different from the current text font.

EXAMPLE

<P>The COPY command has the following syntax:

<SYNTAX>

COPY input-file output-file

<ENDSYNTAX>

This example may produce the following output:

The COPY command has the following syntax:

COPY input-file output-file

REMOVE

10 LETTER Doctype Tag Reference

This chapter provides reference information on the tags specific to the LETTER doctype. These tags allow you to create memos and letters. In general, tag names prefixed with MEMO_ are used to create memos; the other tags are used to create letters or to create headings in either memos or letters. You can use these tags in any sequence to create your own letter and memo formats. Chapter 3 provides the tutorial information on the use of these tags.

The LETTER doctype supports all of the VAX DOCUMENT global tags with the following exceptions:

- <APPENDIX>
- <CHAPTER>
- <FRONT_MATTER>
- <HEAD1> through <HEAD6>
- <PART_PAGE>

Although the LETTER doctype does not support the global numbered heading tags (<HEADn>), it does support the global unnumbered heads (<SUBHEAD1> and <SUBHEAD2>).

See the *VAX DOCUMENT User Manual, Volume 1* for more information on the global tags.

LETTER Doctype Tag Reference

<CC>

<CC>

Lists the name of someone who is to receive a copy of a memo or letter.

FORMAT <CC> (*receiver-name*)

ARGUMENTS *receiver-name*
Specifies the name of someone who should receive a copy of the memo or letter.

related tags • <CCLIST>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <CC> tag lists a single name of someone who is to receive a copy of a memo or letter. This tag can be used by itself or within the context of the <CCLIST> tag.

If the <CC> tag is used alone, it places the heading "cc:" on the left margin and places the *receiver-name* argument on the same line as that heading.

If the <CC> tag is used within the context of the <CCLIST> tag, it places the text of the *receiver-name* argument in the same location as when it is used without the <CCLIST> tag, but omits the heading.

EXAMPLES The following example shows the beginning of a letter using the <CCLIST> tag with the <CC> tag.

```
1 <MEMO_FROM>(Bob Smith\Harvard University)
  <MEMO_TO>(Carol Jones)
  <CCLIST>
  <CC>(Mr. A. Square)
  <CC>(Ms. B. Box)
  <ENDCCLIST>
  <SUBJECT>(Ted Fields and Alice Johnson)
  <P>
  This is some text to show you where the text begins.
```

The following example shows the beginning of a letter using only the <CC> tag.

LETTER Doctype Tag Reference

<CC>

2

<MEMO_FROM>(Bob Smith\Harvard University)

<MEMO_TO>(Carol Jones)

<CC>(Mr. A. Square)

<SUBJECT>(Ted Fields and Alice Johnson)

<P>

This is some text to show you where the text begins.

LETTER Doctype Tag Reference

<CCLIST>

<CCLIST>

Begins a list of persons to whom you want to send a copy of a memo or letter.

FORMAT <CCLIST>

ARGUMENTS *None.*

related tags • <CC>
 • <DISTLIST>

restrictions *None.*

required terminator <ENDCCLIST>

DESCRIPTION The <CCLIST> tag begins a listing of one or more persons' names who are to receive a copy of a memo or letter. The <CCLIST> tag places the heading "cc:" on the left margin. The names are specified using the <CC> tag.

The names are formatted such that the argument to the first <CC> tag is placed on the same line as the heading, and the text arguments associated with any following <CC> tags are placed immediately beneath the argument to the first <CC> tag.

EXAMPLE The following example shows the beginning of a letter using the <CCLIST> tag with the <CC> tag.

```
<MEMO_FROM>(Bob\Harvard University)
<MEMO_TO>(Carol)
<CCLIST>
<CC>(Mr. A. Square)
<CC>(Ms. B. Box)
<ENDCCLIST>
<SUBJECT>(Ted and Alice)
<P>
This is some text to show you where the text begins.
.
.
.
<CLOSING>(Best Wishes,\Bob\Chairman, QZZA, Inc.)
```

<CHEAD>

Creates a centered heading.

FORMAT <CHEAD> (*heading-text* [\ *symbol-name*])

ARGUMENTS *heading-text*
Specifies the text of the centered heading.

symbol-name
Specifies a symbol to identify the heading associated with the <CHEAD> tag. Use this argument if you are planning to reference this heading by its title elsewhere in the document. This argument is provided so that you can reference unnumbered headings.

The *symbol-name* argument can be any text string of 31 characters or less without spaces. This text must contain only the letters A through Z, the numbers 0 through 9 and the underscore (_) character. The *symbol-name* may not begin with an underscore character.

related tags

- <HEAD>
- <SECTION>
- The global <CENTER> tag
- The global <SUBHEAD1> tag

restrictions *None.*

required terminator *None.*

DESCRIPTION The <CHEAD> tag creates a centered unnumbered heading. The capitalization of the letters in this heading is retained in the output as typed. If you need to use a primary heading, use the <HEAD> or <SECTION> tags.

LETTER Doctype Tag Reference

<CHEAD>

EXAMPLE

The following example shows how to use the <CHEAD> tag and the <HEAD> tags to create unnumbered headings.

```
<HEAD>(How to Write a Letter)
```

```
<P>
```

```
The first thing that you should do in writing a letter is to get  
a clean piece of paper and a well-sharpened pencil.
```

```
<CHEAD>(Getting the Piece of Paper)
```

```
<P>
```

```
You can buy clean paper in most major supermarkets, department stores,  
and hardware stores. You should try to get ruled paper so that  
your letter will be neat and easy to read.
```


LETTER Doctype Tag Reference

<DISTLIST>

<DISTLIST>

Begins a list of persons to whom you want to distribute a memo or letter.

FORMAT <DISTLIST>

ARGUMENTS *None.*

related tags • <CCLIST>

restrictions *None.*

required terminator <ENDDISTLIST>

DESCRIPTION The <DISTLIST> tag begins a list of persons to whom you want to distribute a memo or letter. The <DISTLIST> tag places the heading "Distribution:" on the left margin. The names of the people on the distribution list appear beneath the heading, formatted exactly as you entered them between the <DISTLIST> and <ENDDISTLIST> tags.

The <DISTLIST> tag retains all spacing and capitalization exactly as entered. This lets you place asterisks before certain names, indent certain names, and so on.

EXAMPLE The following example shows the end of a letter using the <DISTLIST> tag. The format of the text within the context of the <DISTLIST> tag will be retained exactly as entered, just as the global <CODE_EXAMPLE> tag retains formatting exactly as entered.

```
<DISTLIST>
*Bob
Carol  *Ted  Alice
Pete   Jon  *Art
* - Indicates primary reviewer
<ENDDISTLIST>
```


LETTER Doctype Tag Reference

<FROM_ADDRESS>

EXAMPLE

The following example shows the beginning of a letter that uses the <FROM_ADDRESS> tag.

```
<FROM_ADDRESS>(Bob Smith\Harvard University\Cambridge, MA)
<TO_ADDRESS>(Carol Jones\World Wide Wickets Co.\Seattle, WA)
<SUBJECT>(Ted and Alice)
<SALUTATION>(Hi Carol,)
<P>
This is the text of the letter
.
.
.
<CLOSING>(Best Wishes,\Bob Smith\Chairman, QZZA, Inc.)
```

<HEAD>

Creates a main heading on the left of the page.

FORMAT <HEAD> (*heading-text* [\ *symbol-name*])

ARGUMENTS *heading-text*
Specifies the text of the heading

symbol-name

Specifies the symbol to identify the heading associated with the <HEAD> tag. Use this argument if you are planning to reference this heading by its title elsewhere in the document. This argument is provided so that you can reference unnumbered headings.

The *symbol-name* argument can be any text string of 31 characters or less without spaces. This text must contain only the letters A through Z, the numbers 0 through 9 and the underscore (–) character. The *symbol-name* may not begin with an underscore character.

related tags

- <CHEAD>
- <SECTION>
- The global <SUBHEAD1> tag

restrictions *None.*

required terminator *None.*

DESCRIPTION The <HEAD> tag creates an unnumbered heading on the left side of the page. Other tags in the LETTER doctype that produce unnumbered headings are the doctype-specific <SECTION> and <CHEAD> tags, and the global <SUBHEAD1> and <SUBHEAD2> tags.

See the appropriate reference descriptions in this chapter for more information on the LETTER doctype tags, or see the *VAX DOCUMENT User Manual, Volume 1* for information on any of the global tags.

LETTER Doctype Tag Reference

<HEAD>

EXAMPLE

The following example shows how to use the <HEAD> tag and the <CHEAD> tags to create unnumbered headings.

```
<HEAD>(How to Write a Letter)
```

```
<P>
```

```
The first thing that you should do in writing a letter is to get  
a clean piece of paper and a well-sharpened pencil.
```

```
<CHEAD>(Getting the Piece of Paper)
```

```
<P>
```

```
You can buy clean paper in most major supermarkets, department stores,  
and hardware stores. You should try to get ruled paper so that  
your letter will be neat and easy to read.
```

<MEMO_DATE>

Creates a line in a memo or letter that displays the date after the heading "Date:."

FORMAT <MEMO_DATE> (*date-argument*)

ARGUMENTS *date-argument*

Specifies the date of the memo or letter. This argument can be the global <DATE> tag (which returns the date the file was processed on), or a date that you specify.

related tags

- <FROM_ADDRESS>
- <MEMO_LINE>
- <MEMO_TO>
- The global <DATE> tag

restrictions

None.

required terminator

None.

DESCRIPTION The <MEMO_DATE> tag places the heading "Date:" on the left margin and places the *date-argument* argument immediately after it on the same line.

If you want the date to be the date on which you processed the file, use the global <DATE> tag as the argument to the <MEMO_DATE> tag.

If you want a date that does not vary each time you process the file, or a date that follows a different format than the format output by the <DATE> tag, enter that date explicitly as a text argument to the <MEMO_DATE> tag.

EXAMPLES

The following example shows the beginning of a memo using the <MEMO_DATE> tag with the global <DATE> tag as an argument.

```
1 <MEMO_HEADER>
  <MEMO_FROM>(Bob\Dept. of English)
  <MEMO_TO>(Carol\Dept. of Archeology)
  <MEMO_LINE>(Req No.\ARC-132)
  <MEMO_DATE>(<DATE>)
  <SUBJECT>(Awards for Ted and Alice)
  <P>
  This is the text of the memo.
```

The following example shows the beginning of a memo using the <MEMO_DATE> tag with a text string as an argument.

LETTER Doctype Tag Reference

<MEMO_DATE>

2 <MEMO_HEADER>
<MEMO_FROM>(Bob Smith\Dept. of English)
<MEMO_LINE>(Phone:\9-5151)
<MEMO_TO>(Carol Jones\Dept. of Archeology)
<MEMO_LINE>(Req No.\ARC-132)
<MEMO_DATE>(January 1, 1987, 10:00 pm)
<SUBJECT>(Awards for Ted and Alice)
<P>
This is the text of the memo.

LETTER Doctype Tag Reference

<MEMO_FROM>

EXAMPLE

The following example shows a typical beginning of a memo using the <MEMO_FROM> tag.

```
<MEMO_HEADER>
<MEMO_FROM>(Bob Smith\Dept. of English)
<MEMO_TO>(Carol Jones\Dept. of Archeology)
<MEMO_LINE>(Req No.\ARC-132)
<MEMO_DATE>(<DATE>)
<SUBJECT>(Ted and Alice)
<P>
This is the text of the memo.
.
.
<CLOSING>(Yours Truly,\Bob Smith\CEO, Harvard English Dept)
```

<MEMO_HEADER>

Centers the heading "Interoffice Memorandum" in bold letters on the page.

FORMAT <MEMO_HEADER>

ARGUMENTS *None.*

related tags

- <MEMO_TO>
- <MEMO_FROM>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <MEMO_HEADER> tag automatically centers the heading "Interoffice Memorandum" in bold letters on the page. This tag accepts no arguments.

EXAMPLE The following example shows a typical beginning of a memo using the <MEMO_HEADER> tag.

```
<MEMO_HEADER>
<MEMO_FROM>(Bob Smith\Dept. of English)
<MEMO_TO>(Carol Jones\Dept. of Archeology)
<MEMO_DATE>(<DATE>)
<SUBJECT>(Ted and Alice)
<P>
This is the text of the memo.
```


LETTER Doctype Tag Reference

<MEMO_LINE>

EXAMPLE

The following example shows how the <MEMO_LINE> tag can be used to create an additional line of information with a heading. In this example, the heading "Corp:" is created with the text following it being "Drofnats Ltd." Note that the *heading-text* argument does not exceed seven characters; note also that even though a colon is desired in the heading it is not specified as part of the *heading-text* argument.

<MEMO_HEADER>

<MEMO_FROM>(J. Simpson\Accounting Consultant)

<MEMO_LINE>(Corp\Drofnats Ltd.)

<MEMO_TO>(Mr. Smith\ACME Corporate Accounting)

<MEMO_DATE>(March 17, 1986)

<CC>(Departmental Distribution)

<SUBJECT>(Conference Report)

<P>This conference was hosted by Numbers Inc. in Seattle, Washington

LETTER Doctype Tag Reference

<MEMO_TO>

EXAMPLE

The following example shows a typical beginning of a memo using the <MEMO_TO> tag.

```
<MEMO_HEADER>  
<MEMO_FROM>(Bob Smith\Dept. of English)  
<MEMO_TO>(Carol Jones\Dept. of Archeology)  
<SUBJECT>(Ted and Alice)  
<P>  
This is the text of the memo.
```

LETTER Doctype Tag Reference

<SALUTATION>

<SALUTATION>

Specifies the salutation for the letter.

FORMAT <SALUTATION> (*greeting-text*)

ARGUMENTS *greeting-text*
Specifies the text of the salutation, including any punctuation.

related tags

- <FROM_ADDRESS>
- <TO_ADDRESS>
- <MEMO_FROM>
- <MEMO_TO>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <SALUTATION> tag lets you select the opening greeting of your letter or memo, such as "Dear Sirs:." This text begins at the left margin.

Any punctuation that is part of the salutation (such as a comma, colon, or semicolon) must be specified as part of the *greeting-text* argument.

EXAMPLE The following example shows a use of the <SALUTATION> tag in the beginning of a letter. Note that you must provide any needed punctuation, such as the comma after "Hi Carol" in this example.

```
<FROM_ADDRESS>(Bob Smith\Harvard University\ Cambridge, MA)
<TO_ADDRESS>(Carol Jones\World Wide Wickets Co.\Seattle, WA)
<SUBJECT>(Ted and Alice)
<SALUTATION>(Hi Carol,)
<P>
This is the text of the letter.
```

<SUBJECT>

Specifies the subject of a memo or letter and places this information with a heading of "Subject:" at the left margin.

FORMAT **<SUBJECT>** (*subject-text*)

ARGUMENTS ***subject-text***
Specifies the text that describes the subject of a memo or letter.

related tags

- <FROM_ADDRESS>
- <MEMO_FROM>
- <MEMO_TO>
- <MEMO_LINE>

restrictions *None.*

required terminator *None.*

DESCRIPTION This tag is used to create a subject line with a heading. The <SUBJECT> tag places the heading "Subject:" at the left margin and places the text from the *subject-text* argument on that same line.

EXAMPLE The following example shows a use of the <SUBJECT> tag in the beginning of a letter. Note that this tag can also be used in a memo.

```
<FROM_ADDRESS>(Bob Smith\Harvard University\Cambridge, MA)
<TO_ADDRESS>(Carol Jones\World Wide Wickets Co.\Seattle, WA)
<SUBJECT>(Ted and Alice)
<SALUTATION>(Hi Carol,)
<P>
This is the text of the letter.
```


11 MILSPEC Doctype Tag Reference

This chapter contains the reference information on all the tags available within the MILSPEC doctype. The MILSPEC doctype is a full implementation of the United States Military Specification Standard MIL-STD-490A. Chapter 5 contains a tutorial on how to use these tags.

The MILSPEC doctype accepts the full range of VAX DOCUMENT global tags, with the exception of the <PART> and <PART_PAGE> tags. See the *VAX DOCUMENT User Manual, Volume 1*, for more information on global tags.

The MILSPEC doctype also provides additional doctype-specific tags that let you create the front matter of your document in accordance with the MIL-STD-490A standard. These tags are described alphabetically in this chapter.

MILSPEC Doctype Tag Reference

<SET_APPENDIX_NUMBER>

<SET_APPENDIX_NUMBER>

Overrides the default appendix Roman numeral assigned to an appendix by VAX DOCUMENT.

FORMAT <SET_APPENDIX_NUMBER> (*Roman-number-integer*)

ARGUMENTS *Roman-number-integer*

Specifies an integer that evaluates to the Roman numeral for the appendix. This argument must be an integer greater than zero.

related tags

- The global <APPENDIX> tag
- The global <SET_APPENDIX_LETTER> tag
- The global <SET_CHAPTER_NUMBER> tag

restrictions

This tag should not be used in a file that uses a book build profile or that uses the /PROFILE qualifier during processing. If either of these conditions occur, a warning message is issued and the <SET_APPENDIX_NUMBER> tag is ignored.

required terminator

None.

DESCRIPTION

Use the <SET_APPENDIX_NUMBER> tag to override the default appendix Roman numeral created by VAX DOCUMENT.

The <SET_APPENDIX_NUMBER> tag resets the current appendix Roman numeral to the integer you specify as the *Roman-number-integer*. If you specified 4 as the Roman numeral integer, VAX DOCUMENT would set the appendix number to "IV."

Place the <SET_APPENDIX_NUMBER> tag in your SDML file before the <APPENDIX> tags you want it to affect because the <SET_APPENDIX_NUMBER> tag affects only the <APPENDIX> tags that follow it.

The new appendix number you specify resets the numbering for all following appendixes. For example, if you set the appendix number to 2, the next appendix will be numbered "II," the appendix following that appendix will be numbered "III," and so on.

The <SET_APPENDIX_NUMBER> can be used multiple times in an SDML file.

MILSPEC Doctype Tag Reference

<SET_APPENDIX_NUMBER>

EXAMPLE

In the following example the appendix is explicitly set to IV using the <SET_APPENDIX_NUMBER> tag. This will cause any subsequent appendixes to be numbered beginning with the Roman numeral IV unless another <SET_APPENDIX_NUMBER> tag is used.

```
<SET_APPENDIX_NUMBER>(4)
<APPENDIX>(Run-time Functions\functions_ap)
<p>
The following functions...
```

MILSPEC Doctype Tag Reference

<SIGNATURE_LINE>

<SIGNATURE_LINE>

Creates up to two rules on a line and places a name below each rule; each rule is used as a signatory line for the person listed below it.

FORMAT <SIGNATURE_LINE> ([name-1] \ [name-2])

ARGUMENTS *name-n*
Specifies the name or title of the person who is to sign on the previous line. These names and lines are output in two columns, if one of these arguments is not specified, neither the name nor the signature line is output in that column.

related tags

- <SIGNATURE_LIST>
 - <SPECIFICATION_INFO>
 - <SPEC_TITLE>
 - <SUBTITLE>
 - The global <FRONT_MATTER> tag
 - The global <TITLE_PAGE> tag
-

restrictions

Valid only within the context of the <SIGNATURE_LIST> tag.

required terminator

None.

DESCRIPTION Use the <SIGNATURE_LINE> tag to create up to two rules with the name of a person listed below each rule. This tag creates signature lines within a signature list with the name of the person placed below each signature line. The signature list has a two-column format; if you omit a name in either column, no name (or rule) is placed in that column.

MILSPEC Doctype Tag Reference

<SIGNATURE_LINE>

EXAMPLE

The following example shows a signature list with two names in the first row of signatures, a single name in the left column of the second row, and a name in the right column of the third row.

```
<TITLE_PAGE>
```

```
.
```

```
.
```

```
<SIGNATURE_LIST>
```

```
<SIGNATURE_LINE>(Mr. A. Baloo\Ms. C. Dee)
```

```
<SIGNATURE_LINE>(John Doe\ )
```

```
<SIGNATURE_LINE>( \Mrs. Smith)
```

```
<SIGNATURE_LIST>
```


MILSPEC Doctype Tag Reference

<SIGNATURE_LIST>

EXAMPLE

The following example shows a sample use of the <SIGNATURE_LIST> tag. Note how it is used within the context of the global <TITLE_PAGE> tag.

```
<FRONT_MATTER>
<TITLE_PAGE>
.
<SUBTITLE>(Submitted Under\Contract A00000--11--A--2222)
<SIGNATURE_LIST>(Authenticated by:\Approved by:)
<SIGNATURE_LINE>(Procuring Activity\Program Manager)
<SIGNATURE_LINE>(Date\Technical Director)
<SIGNATURE_LINE>(\Consultant)
<ENDSIGNATURE_LIST>
.
<ENDTITLE_PAGE>
```


MILSPEC Doctype Tag Reference

<SPECIFICATION_INFO>

restrictions

Valid only within the context of the global <TITLE_PAGE> tag.

required terminator

None.

DESCRIPTION

The <SPECIFICATION_INFO> tag creates a listing of information about the specification on the title page and creates a two-line running heading for the rest of the document.

The listing of information is formatted on the upper right of the title page, each line specifies the following.

- Line one lists the *specification-number* argument.
- Line two lists the *code-id-number* argument.
- Line three lists the *specification-date* argument.
- Line four is optional and lists the *additional-info* argument.

The *specification-number* and *specification-date* arguments are additionally used throughout the document as the top and bottom lines (respectively) of a two-line running heading.

EXAMPLE

The following example shows a sample use of the <SPECIFICATION_INFO> tag.

```
<FRONT_MATTER>
<TITLE_PAGE>
<SPECIFICATION_INFO>(12345B\a142-b4\July 4, 1776\Part I of Three Parts)
<SPEC_TITLE>(Prime Item Development Specification\
For the \Approved Title)\of the \Supported\Device)
.
<ENDTITLE_PAGE>
```


12 OVERHEADS Doctype Tag Reference

This chapter provides the reference information on all the tags available within the OVERHEADS doctype. Chapter 6 provides information on how to use these tags.

The OVERHEADS doctype supports all of the VAX DOCUMENT global tags with the following exceptions:

- <APPENDIX>
- <CHAPTER>
- <CONTENTS_FILE>
- <FRONT_MATTER>
- <GLOSSARY>
- <HEAD1> through <HEAD6>
- <INDEX_FILE>
- <PART>
- <PART_PAGE>
- <PROFILE>

See the *VAX DOCUMENT User Manual, Volume 1* for more information on the global tags.

<AUTO_NUMBER>

Specifies that slides are to be numbered automatically, and that the slide number is to be placed at the bottom of every slide. Optionally, specifies text to be placed in front of the slide number.

FORMAT <AUTO_NUMBER> [(text)]

ARGUMENTS *text*
Specifies text to be placed in front of the slide number at the foot of every overhead slide. If this argument is not specified, only the slide number is printed.

related tags • <RUNNING_FEET>
 • <SLIDE>

restrictions *None.*

required terminator *None.*

DESCRIPTION By default, no page or slide number is placed on the bottom of overhead slides. You can override this default behavior in the following ways:

- Use the <AUTO_NUMBER> tag to request numbering and to optionally specify text to go along with the numbers.
- Use the <RUNNING_FEET> tag to specify text that is to be placed on the bottom of every overhead slide.
- Use an argument to the <SLIDE> tag. The text of the tag's argument is placed on the bottom of the current slide.

EXAMPLES In the following example the <AUTO_NUMBER> tag specifies that the slides are to be numbered. The current number will be placed in the bottom right corner of every slide.

1 <AUTO_NUMBER>

In the following example the <AUTO_NUMBER> tag specifies that the slides are to be numbered and that the numbers are to be preceded with the word "Earnings" and placed in the bottom right corner of every slide.

2 <AUTO_NUMBER>(Earnings)

OVERHEADS Doctype Tag Reference

<RUNNING_FEET>

<RUNNING_FEET>

Specifies text to be placed at the bottom of the next slide and all subsequent slides.

FORMAT <RUNNING_FEET> (*running-footer-text*)

ARGUMENTS *running-footer-text*

Specifies the text you want to be placed on your slides on the bottom left of the page.

related tags

- <AUTO_NUMBER>
- <SLIDE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <RUNNING_FEET> tag creates a running footer for a series of slides. This footer is output at the bottom left of the slide. If you want the tag to place running feet on all of your slides, you should place this tag in your SDML file before the first occurrence of a <SLIDE> tag. If you wish to disable these running feet, you should place the <RUNNING_FEET> tag in your SDML file with a null argument.

By default, no page or slide number is placed on the bottom of overhead slides. You can override this default behavior in the following ways:

- Use the <AUTO_NUMBER> tag to request numbering and to optionally specify text to go along with the numbers.
- Use the <RUNNING_FEET> tag to specify text that is to be placed on the bottom of every overhead slide.
- Use an argument to the <SLIDE> tag. The text of the tag's argument is placed on the bottom of the current slide.

If <RUNNING_FEET> is specified in conjunction with <AUTO_NUMBER>, the slide number is placed on the right and the text on the left.

OVERHEADS Doctype Tag Reference

<RUNNING_FEET>

EXAMPLES

In the following example, the <RUNNING_FEET> tag specifies that all slides are to carry the text "May 1986 Presentation" at the bottom right corner.

1 <RUNNING_FEET>(May 1986 Presentation)
<SLIDE>
<TITLE>(Base Level 13)

In the following example, the <AUTO_NUMBER> tag is used to create automatic numbering of the slides at the bottom right of the page, and the <RUNNING_FEET> tag is used to place a running footer on the bottom left of the page.

2 <RUNNING_FEET>(May 1987)
<AUTO_NUMBER>(Slide)

OVERHEADS Doctype Tag Reference

<RUNNING_TITLE>

<RUNNING_TITLE>

Creates a one- or two-line running heading at the top of each slide.

FORMAT

<RUNNING_TITLE> ({ *OFF*
title-1 [\ *title-2*]
[\ *FIRST_PAGE*] })

ARGUMENTS

title-1

Specifies the text of a running title. If a two-line title is specified, this title is output on the upper title line.

title-2

Specifies the optional bottom line of a running title that has two lines.

FIRST_PAGE

Specifies that the running title is to begin output on the first slide. If this keyword is not specified, the running title begins on the slide after the current slide.

OFF

Specifies that any existing running titles created using the <RUNNING_TITLE> tag should be disabled for the slide on which this tag occurs and on any subsequent slides.

related tags

- <RUNNING_FEET>
- <SLIDE>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <RUNNING_TITLE> tag to create a one- or two-line running title for a series of slides. If you want the tag to place a running title on all of your slides, you should place this tag in your SDML file before the first occurrence of a <SLIDE> tag. If you use this tag within the context of the first <SLIDE> tag and you want the running title to begin on that slide, use the *FIRST_PAGE* argument.

Use the *OFF* argument to disable any existing running titles created using the <RUNNING_TITLE> tag. These titles will then be disabled for the page on which this tag occurs and on any subsequent pages.

OVERHEADS Doctype Tag Reference

<RUNNING_TITLE>

Use the <RUNNING_FEET> tag to create a heading that appears at the bottom of the page. See the reference description of the <RUNNING_FEET> tag for more information on that tag.

EXAMPLES

The following examples show how to set running titles at the top of the page for a series of related slides.

In the first example each occurrence of <RUNNING_TITLE> creates a running title for the next slide or series of slides.

1 <RUNNING_TITLE>(Introduction to SDML)
<SLIDE>
<TOPIC>(What is SDML?)
<SLIDE>
<TOPIC>(What is a Tag?)

<RUNNING_TITLE>(Overview of the Tags)
<SLIDE>
<TOPIC>(The Basic Tags)
.
.
.

The following example shows how you can disable a running title by using the OFF argument to the <RUNNING_TITLE> tag.

2 <COMMENT>(turn off running titles for the next slide)
<RUNNING_TITLE>(OFF)
<SLIDE>
<TOPIC>(An Example of Output)

OVERHEADS Doctype Tag Reference

<SLIDE>

<SLIDE>

Begins a new overhead slide.

FORMAT <SLIDE> (*footer-text*)

ARGUMENTS *footer-text*
Specifies text to be placed at the bottom of the slide.

related tags

- <RUNNING_TITLE>
- <AUTO_NUMBER>
- <RUNNING_FEET>
- <INTRO_TITLE>
- <INTRO_SUBTITLE>
- <AUTHOR_INFO>
- <TITLE>
- <SUBTITLE>
- <TOPIC>
- <TEXT_SIZE>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <SLIDE> tag begins a new overhead slide and generates a new page of output. An overhead slide can consist of the following:

- Major titles and subtitles
Use the <INTRO_TITLE> and <INTRO_SUBTITLE> tags to create title page slides.
- Titles and subtitles followed by topics, lists, tables or text
Use the <TITLE> and <SUBTITLE> tags to place headings at the top of each new slide. Use the global <LIST> (numbered) and <LIST> (unnumbered) tags for numbered and unnumbered lists.

You can use any of the global tags to specify text elements on any slide page, except those associated with the creation of front matter, appendixes, glossaries, indexes, or part pages.

EXAMPLES

The following example shows a sequence of tags that begin a new slide and specify two main topics.

1 <SLIDE>
<TITLE>(System Components)
<TOPIC>(Parser)
<TOPIC>(Interpreter)

In the following example a slide is specified with a main heading and a subheading. The text "Slide 1" will be placed at the bottom of this slide.

2 <SLIDE>(Slide 1)
<INTRO_TITLE>(A New \ Production System)
<INTRO_SUBTITLE>(Introduction and Overview)

The following example shows a slide that uses a numbered list to number steps.

3 <SLIDE>
<TITLE>(WORK FLOW)
<LIST>(NUMBERED)
<LE>Create the File
<LE>Run the Checker
<ENDLIST>

<TEXT_SIZE>

Changes the size of type used within the context of topics, tables, and lists on a single slide.

FORMAT

<TEXT_SIZE> ({ *SMALL*
REGULAR
TABLE })

ARGUMENTS

SMALL
Reduces the standard point size of type used in topics and lists.

REGULAR
Increases the size of type in tables to that of standard text.

TABLE
Reduces text to the smallest font size available in the OVERHEADS doctype, which is the standard font size used inside of tables.

related tags

- <SLIDE>

restrictions

None.

required terminator

<ENDTEXT_SIZE>

DESCRIPTION

Use the <TEXT_SIZE> tag to change the size of type in a single overhead slide. This tag alters the type face used by text in the following contexts:

- Text specified within a list using the global <LIST> tag
- Text specified as an argument to the <TOPIC> tag
- Text specified within the context of the global <TABLE> tag

The <TEXT_SIZE> tag alters the default size of type only on a single slide. The following slide will have the default type sizes. To alter the type size for more than one slide, use the <TEXT_SIZE> tag on each of the slides.

OVERHEADS Doctype Tag Reference

<TEXT_SIZE>

EXAMPLE

The following example illustrates each of the various formats available using the <TEXT_SIZE> tag.

```
<SLIDE>
<TOPIC>(ANIMALS THAT MAKE GOOD PETS)
<list>(UNNUMBERED)
<LE>Rabbits---Small, furry, generally best as pets for children
growing up in a non-urban setting.
<p><TEXT_SIZE>(small)
Rabbits are typically not good pets in urban settings because of their
extreme sensitivity to noise and because of their love of the outdoors.
<ENDTEXT_SIZE>
<LE>Dogs---Come in assorted shapes and sizes: a general purpose pet.
<p><TEXT_SIZE>(TABLE)
Dogs are man's (and woman's) best friend.
<ENDTEXT_SIZE>
<LE>Cats---Cats are ideal pets for apartment dwellers.
<p>
<TEXT_SIZE>(regular)
Cats should not be pets in households that already have tropical fish for pets.
<ENDTEXT_SIZE>
<ENDLIST>
```

<TITLE>

Specifies a title for a new slide.

FORMAT <TITLE> (*title-line-1*[\ *title-line-2*] . . . [\ *title-line-4*])

ARGUMENTS *title-line-n*
Specifies up to four lines of text for a main slide title.

related tags • <SUBTITLE>

restrictions *None.*

required terminator *None.*

DESCRIPTION Use the <TITLE> tag to create a main title for an overhead slide that has up to four separate lines. Each of the title lines is centered on the slide. Use the <SUBTITLE> tag to create a subordinate title for an overhead slide.

EXAMPLES The following example shows a sequence of tags that begin a new slide and specifies a single-line title.

1 <SLIDE>
 <TITLE>(System Components)
 <TOPIC>(Parser)
 <TOPIC>(Interpreter)

This example illustrates a three-line title. Each line is centered.

2 <TITLE>(THE \DEVELOPMENT \CYCLE)

OVERHEADS Doctype Tag Reference

<TOPIC>

<TOPIC>

Specifies a line of topic text for a slide.

FORMAT <TOPIC> (*topic-text*)

ARGUMENTS *topic-text*
Specifies text for the topic.

related tags • <SLIDE>
 • <TEXT_SIZE>

restrictions *None.*

required terminator *None.*

DESCRIPTION Use the <TOPIC> tag to specify a topic line for a slide. The text of the <TOPIC> tag begins at the left margin and has a smaller type font than the font used by the <TITLE> tag. You can alter the size of the topic text by using the <TEXT_SIZE> tag.

EXAMPLE The following example illustrates a slide with a title, a topic sentence and an unnumbered list.

```
<SLIDE>
<TITLE>(THE \DEVELOPMENT \CYCLE)
<TOPIC>(WHO)
<LIST>(UNNUMBERED)
<LE>WRITERS
<LE>EDITORS
<LE>COMPOSITORS AND ARTISTS
<ENDLIST>
```

13 REPORT Doctype Tag Reference

This chapter contains the reference information on the doctype-specific tags available within the REPORT doctype. Chapter 7 provides information on how to use these tags.

The REPORT doctype accepts the full range of VAX DOCUMENT global tags. See the *VAX DOCUMENT User Manual, Volume 1* for more information on the global tags. The REPORT doctype also provides tags that let you do the following:

- Create headings or modify the default attributes of the REPORT doctype
- Create signature lines and list author information within the context of the global `<FRONT_MATTER>` tag
- Create formal outlines within the context of the `<OUTLINE>` tag

These tags are described alphabetically in this chapter.

EXAMPLE

The following example shows how you can use the <AUTHOR> tag within the front matter of a document. Note how the optional second argument to the <AUTHOR> tag is used to list the author's title.

```
<FRONT_MATTER>
<TITLE_PAGE>
<TITLE>(The NYUC Simulator Reference Manual)
<ORDER_NUMBER>(AA-Z0000-TE)
<ABSTRACT>
This manual describes the NYUC Simulator.
This program simulates a conversation between three people
by analyzing the syntactic and semantic components of three
related statements, and then synthesizing statements and responses
based upon these original statements.
<ENDABSTRACT>
<REVISION_INFO>(This revision is personally signed.)
<AUTHOR>(Mr. Jones\Research Head, STG Inc.)
<SIGNATURES>
<BYLINE>(Nat Jones\Author)
<DATE>(July 11, 1985)
<PRINT_DATE>(June 1987)
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
```

REPORT Doctype Tag Reference

<BYLINE>

<BYLINE>

Places a name and other optional information below a ruled line in a signature list.

FORMAT <BYLINE> (*name* [\ *additional-info*])

ARGUMENTS *name*
Specifies the name of the signatory. This name is placed under the beginning of the signature line on the left side of the page.

additional-info
Specifies any additional information about the signatory. This information is placed on the same line as the *name* argument with an em dash (—) between the two arguments.

related tags

- <AUTHOR>
- <SIGNATURES>
- The global <FRONT_MATTER> tag

restrictions Valid only in the context of the global <FRONT_MATTER> tag after the <SIGNATURES> tag.

required terminator *None.*

DESCRIPTION Use the <BYLINE> tag to create an approval line in the front matter of a document and to place the name of a signatory beneath that line. You can place additional information about the signer by using the *additional-info* argument. Additional information is formatted to the right of the name of the signer, on the same line, separated by an em dash (—).

You can use as many <BYLINE> tags as you want to create approval lines within the front matter of a document, as long as all of these tags follow the <SIGNATURES> tag. You can use the <SIGNATURES> tag to begin all the approval lines on a separate page of the front matter. See the <SIGNATURES> tag in this chapter for more information.

EXAMPLE

The following example shows three occurrences of the <BYLINE> tag. The first two occurrences list the positions of the signers using the optional *additional-info* argument. The third occurrence of the <BYLINE> tag omits the optional argument. Note that all three tags follow the <SIGNATURES> tag.

```
<FRONT_MATTER>
<TITLE_PAGE>
<TITLE>(The NYUC Simulator Reference Manual)
<REVISION_INFO>(This revision is personally signed.)
<AUTHOR>(Mr. Jones\Research Head, STG Inc.)
<SIGNATURES>
<BYLINE>(Nat Jones\Author)
<BYLINE>(Cecil Mills\Co-author)
<BYLINE>(Matt Smith)
<DATE>(July 11, 1985)
<PRINT_DATE>(June 1987)
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
```

REPORT Doctype Tag Reference

<CHEAD>

<CHEAD>

Creates a centered heading.

FORMAT <CHEAD> (*heading-text* [\ *symbol-name*])

ARGUMENTS *heading-text*

Specifies the text of the centered heading.

symbol-name

Specifies a symbol to identify the heading associated with the <CHEAD> tag. You can use this argument if you are planning to reference this heading by its title elsewhere in the document. This argument is provided so that you can reference unnumbered headings.

The *symbol-name* argument can be any text string of 31 characters or less without spaces. This text must contain only the letters A through Z, the numbers 0 through 9 and the underscore (_) character. The *symbol-name* may not begin with an underscore character.

related tags

- <HEAD>
 - <SECTION>
 - The global <CENTER> tag
 - The global <HEAD1> tag
 - The global <REFERENCE> tag
-

restrictions

None.

required terminator

None.

DESCRIPTION

The <CHEAD> tag places a secondary, unnumbered heading on the center of the output page. The capitalization of the heading is kept as you enter it. If you need to use a primary unnumbered heading, use the <HEAD> or the <SECTION> tags.

EXAMPLE

The following example shows how to use the <CHEAD> and <HEAD> tags to create unnumbered headings. Note that the <CHEAD> tag uses the optional *symbol-name* argument so that it can be referred to in the previous section using the global <REFERENCE> tag.

```
<HEAD>(How to Write a Letter)
<P>
```

```
The first thing that you should do in writing a letter is to get
a clean piece of paper and a well-sharpened pencil.
<REFERENCE>(paper_getting) describes how to procure a piece of paper.
```

```
<CHEAD>(Getting the Piece of Paper\paper_getting)
<P>
```

```
You can buy clean paper in most major supermarkets, department stores,
and hardware stores. You should try to get ruled paper so that
your letter will be neat and easy to read.
```

REPORT Doctype Tag Reference

<COLUMN>

<COLUMN>

Specifies that a new column of output should begin in a two-column doctype.

FORMAT <COLUMN>

ARGUMENTS *None.*

related tags • The global <FINAL_CLEANUP> tag

restrictions Valid only in a two-column doctype such as REPORT.TWOCOL or ARTICLE.

required terminator *None.*

DESCRIPTION Use the <COLUMN> tag to cause the text immediately following it to be started in a new column. If this tag occurs in the left text column, the text immediately following it begins in the right text column. If this tag occurs in the right text column, the text immediately following it begins in the left column of the next page.

Use the <COLUMN> tag when you always want to begin a new column at that point in your text. You can use the COLUMN_BREAK argument to the global <FINAL_CLEANUP> tag to also specify a column break; however, this should be used only during the final processing of the two-column document.

See Section 2.2 for more information on improving the formatting of a two-column doctype such as REPORT.TWOCOL.

EXAMPLE The following example shows how to use the <COLUMN> tag to begin a new text column. In this example, the writer wants the two descriptions to appear side by side, one in each column.

REPORT Doctype Tag Reference

<COLUMN>

```
<HEAD>(Woodwind Instruments)
<P>Woodwind instruments have the following
attributes:
<LIST>(UNNUMBERED)
<LE>They are often made of wood, hence their name.
<LE>Musicians create sound using these instruments by causing a reed
to vibrate.
.
.
<ENDLIST>
<COLUMN>
<HEAD>(Brass Instruments)
<P>Brass instruments have the following
attributes:
<LIST>(UNNUMBERED)
<LE>They are often made of brass, hence their name.
<LE>Musicians create sound using these instruments by
vibrating (buzzing) their lips into a steel mouthpiece.
.
.
<ENDLIST>
```

REPORT Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

<DOCUMENT_ATTRIBUTES>

Enables doctype-specific tags that override the default design format of the REPORT doctype.

FORMAT	<DOCUMENT_ATTRIBUTES>
---------------	-----------------------

ARGUMENTS	<i>None.</i>
------------------	--------------

related tags	<i>None.</i>
---------------------	--------------

restrictions	<i>None.</i>
---------------------	--------------

required terminator	<ENDDOCUMENT_ATTRIBUTES>
----------------------------	--------------------------

DESCRIPTION	The <DOCUMENT_ATTRIBUTES> tag can be used in three doctypes:
--------------------	--

- ARTICLE
- REPORT
- SOFTWARE

The <DOCUMENT_ATTRIBUTES> tag enables a group of tags in each of these doctypes that allow you to modify the default format of that doctype. These tags are recognized only within the context of the <DOCUMENT_ATTRIBUTES> tag. If other VAX DOCUMENT tags occur in this context, they are ignored, as if they had occurred within the context of a <COMMENT> tag.

Typically, you use the <DOCUMENT_ATTRIBUTES> tag at the beginning of an input file (or in a file specified using the DOCUMENT /INCLUDE qualifier) to alter the default format of a doctype for the processing of that entire file.

Table 13-1 summarizes the formatting tags enabled by the <DOCUMENT_ATTRIBUTES> tag in each of the three supported doctypes.

REPORT Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

Table 13–1 Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> tag

Formatting Tags	Description
Tags Enabled in the ARTICLE Doctype	
<SET_HEADINGS> (UNNUMBERED)	<p>The <SET_HEADINGS> tag specifies whether numbered or unnumbered headings are produced by the heading-level tags (<HEAD1> , <HEAD2> , and so on). By default, headings are not numbered in a document processed using the ARTICLE doctype.</p> <p>Use the <SET_HEADINGS> (NUMBERED) tag to specify that your headings should be numbered.</p>
<SET_HEADINGS> (NUMBERED)	
Tags Enabled in the REPORT Doctype	
<SET_PAGE_NUMBERING> (BY_CHAPTER)	<p>The <SET_PAGE_NUMBERING> tag specifies how pages are to be numbered in a document processed using the REPORT doctype. Pages are numbered sequentially by default.</p> <p>If the BY_CHAPTER argument is used, the pages are numbered by chapter. For example, the second page in Chapter 3 would be numbered 3-2. If the BY_CHAPTER argument is used in a document that contains no chapters, the page numbering is sequential.</p> <p>The SEQUENTIAL argument specifies that the pages should be numbered sequentially, as they are by default.</p>
<SET_PAGE_NUMBERING> (SEQUENTIAL)	
<SET_FORMAL_ELEMENT_NUMBERING> (BY_CHAPTER)	<p>The <SET_FORMAL_ELEMENT_NUMBERING> tag specifies how formal tables, figures, and examples are to be numbered in a document processed using the REPORT doctype. By default, formal tables, figures, and examples are numbered sequentially.</p> <p>The BY_CHAPTER argument indicates that formal elements are to be numbered using the chapter prefix; for example, the first formal table in Chapter 4 would be numbered as Table 4-1.</p> <p>The SEQUENTIAL argument indicates that formal elements are to be numbered sequentially throughout the document; for example, the eighth formal table in the document, regardless of what chapter it occurs in, would be numbered as Table 8.</p> <p>If the BY_CHAPTER argument is used in a document that contains no chapters, VAX DOCUMENT uses the default sequential formal element numbering.</p>
<SET_FORMAL_ELEMENT_NUMBERING> (SEQUENTIAL)	

REPORT Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

Table 13–1 (Cont.) Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> tag

Formatting Tags	Description
Tags Enabled in the SOFTWARE Doctype	
<SET_RUNNING_TITLES> (BY_HEADONE)	<p>The <SET_RUNNING_TITLES> tag specifies that the running title at the top of each page is composed of two lines: the first line is the title of the chapter, and the second line is the heading text of the current <HEAD1> tag. Note that the argument BY_HEADONE is required.</p> <p>By default, the chapter title is used as a single-line running title.</p>

EXAMPLE

The following example is of a file that is to be processed under the REPORT doctype. This example shows how the <SET_PAGE_NUMBERING> and <SET_FORMAL_ELEMENT_NUMBERING> tags can be used to create page and formal element numbering that is chapter-oriented rather than sequential. Note how the BY_CHAPTER argument is used by both tags to specify that numbering should be by chapter rather than sequential.

```
<DOCUMENT_ATTRIBUTES>
<SET_PAGE_NUMBERING> (BY_CHAPTER)
<SET_FORMAL_ELEMENT_NUMBERING> (BY_CHAPTER)
<ENDDOCUMENT_ATTRIBUTES>
```

<HEAD>

Places a main heading on the left of the page.

FORMAT **<HEAD>** (*heading-text* [\ *symbol-name*])

ARGUMENTS ***heading-text***
Specifies the text of the heading.

symbol-name

Specifies the symbol to identify the heading associated with the <HEAD> tag. Use this argument if you are planning to reference this heading by its title elsewhere in the document. This argument is provided so that you can reference unnumbered headings.

The *symbol-name* argument can be any text string of 31 characters or less without spaces. This text must contain only the letters A through Z, the numbers 0 through 9 and the underscore (_) character. The *symbol-name* may not begin with an underscore character.

related tags

- <CHEAD>
 - <SECTION>
 - The global <HEAD1> tag
 - The global <REFERENCE> tag
-

restrictions *None.*

required terminator *None.*

DESCRIPTION The <HEAD> tag is one of three unnumbered heading tags available only in the REPORT doctype. This tag places an unnumbered heading on the left-hand side of the output page.

The other REPORT doctype-specific tags that produce unnumbered headings are the <SECTION> and the <CHEAD> tags. See the appropriate reference descriptions in this chapter for more information on these tags.

You may also want to use the global <SUBHEAD1> and <SUBHEAD2> tags to produce unnumbered headings in the REPORT doctype. See the *VAX DOCUMENT User Manual, Volume 1* for more information on these tags.

REPORT Doctype Tag Reference

<HEAD>

EXAMPLE

The following example shows how to use the <HEAD> and <CHEAD> tags to create unnumbered headings. Note that the <HEAD> tag uses the optional *symbol-name* argument so that it can be referred to in the previous section using the <REFERENCE> tag.

```
<CHEAD>(How to Write a Letter)
```

```
<P>
```

```
The first thing that you should do in writing a letter is to get  
a clean piece of paper and a well-sharpened pencil.
```

```
<REFERENCE>(paper_getting) describes how to procure a piece of paper.
```

```
<HEAD>(Getting the Piece of Paper\paper_getting)
```

```
<P>
```

```
You can buy clean paper in most major supermarkets, department stores,  
and hardware stores. You should try to get ruled paper so that  
your letter will be neat and easy to read.
```

<LEVEL>

Specifies an outline entry and the organizational level of that outline entry.

FORMAT **<LEVEL>** (*level-number* \ *entry-text*)

ARGUMENTS ***level-number***

Specifies the organizational level of the entry. This argument can be any whole number from 1 to 6.

entry-text

Specifies the text for a particular level.

related tags

- <OUTLINE>
 - <SHOW_LEVELS>
-

restrictions

Valid only in the context of the <OUTLINE> tag.

required terminator

None.

DESCRIPTION

The <LEVEL> tag is used to create numbered entries in an outline. Top-level entries (those specified as <LEVEL> (1)) are marked using uppercase Roman numerals. At the lowest level, level 6, the entries are marked with lowercase letters enclosed in parentheses. The top level is formatted at the current left margin; each lower level is indented from the level above it.

EXAMPLE

The following example illustrates an outline created using the <LEVEL> tag in the context of the <OUTLINE> tag. Note how the <LEVEL> tags are indented in the SDML file to make the file easier to read and more maintainable.

```
<OUTLINE>(<EMPHASIS>(Maxillary Taxonomy)\An Enumeration of the
Maxillae\from a Dentition Perspective)
<LEVEL>(1\Historical introduction)
<LEVEL>(1\Dentition in various groups of vertebrates)
  <LEVEL>(2\Reptilia)
    <LEVEL>(3\Histology and development of reptile teeth)
      <LEVEL>(4\Survey of forms)
```

EXAMPLE

The following example illustrates an outline created using the <OUTLINE> tag and the tags it enables. Note how the <LEVEL> tags are indented in the SDML file to make the file easier to read and more maintainable. This indentation in the SDML file has no effect on the output, which is indented automatically according to the level specified in the <LEVEL> tags.

```
<OUTLINE><EMPHASIS>(Maxillary Taxonomy)\An Enumeration of the
Maxillae\from a Dentition Perspective)
<LEVEL>(1\Historical introduction)
<LEVEL>(1\Dentition in various groups of vertebrates)
  <LEVEL>(2\Reptilia)
    <LEVEL>(3\Histology and development of reptile teeth)
    <LEVEL>(4\Survey of forms)
  <LEVEL>(2\Mammalia)
    <LEVEL>(3\Histology and development of mammalian teeth)
    <LEVEL>(3\Survey of forms)
      <LEVEL>(4\Primates)
        <LEVEL>(5\Lemuroidea)
        <LEVEL>(5\Anthropoidea)
          <LEVEL>(6\Platyrrhini)
          <LEVEL>(6\Catarrhini)
        <LEVEL>(4\Carnivora)
          <LEVEL>(5\Creodonta)
          <LEVEL>(5\Fissipedia)
            <LEVEL>(6\Aeluroidea)
            <LEVEL>(6\Arctoidea)
          <LEVEL>(5\Pinnipedia)
        <LEVEL>(4\Etc<hellipsis>)
<ENDOUTLINE>
```

REPORT Doctype Tag Reference

<RUNNING_FEET>

<RUNNING_FEET>

Creates a single line heading at the bottom of each page.

FORMAT <RUNNING_FEET> (*title-text*)

ARGUMENTS *title-text*

Specifies the text to be used as a running heading at the foot of the page.

related tags

- <RUNNING_TITLE>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <RUNNING_FEET> tag to place a heading at the bottom of every page. This heading is called a "footer" because it appears at the foot of the page. When the same footer is used for several pages, the footers are collectively called "running feet."

This tag accepts one argument, which is the text that should appear at the bottom of the page. This text is output exactly as entered, including spacing and capitalization.

Use the <RUNNING_TITLE> tag to create a heading at the top of the page. See the <RUNNING_TITLE> tag in this chapter for more information.

Note that the headings established by the <RUNNING_FEET> and <RUNNING_TITLE> tags may be overridden by a subsequent use of the <CHAPTER> tag.

EXAMPLE

The following example shows how to use the <RUNNING_FEET> tag to place the heading "Getting the Piece of Paper" at the bottom of each page. The running footer will be output exactly as entered.

```
<RUNNING_FEET>(Getting the Piece of Paper)
```

```
<CHHEAD>(Getting the Piece of Paper)
```

```
<p>
```

```
You can buy clean paper in most major supermarkets, department stores,  
and hardware stores. You should try to get ruled paper so that  
your letter will be neat and easy to read.
```

<RUNNING_TITLE>

Creates a one- or two-line running heading at the top of each page.

FORMAT

<RUNNING_TITLE> ({ *OFF*
title-1 [\ *title-2*]
[\ *FIRST_PAGE*] })

ARGUMENTS

title-1

Specifies the text of a running title. If a two-line title is specified, this title is output on the upper title line.

title-2

Specifies the bottom line of a running title that has two lines.

FIRST_PAGE

Specifies that the running title is to be placed on the first output page. If this keyword is not specified, the running title is placed on the page after the current page.

OFF

Specifies that any existing running titles created using the <RUNNING_TITLE> tag should be disabled for the page on which this tag occurs and on any subsequent pages.

related tags

- <RUNNING_FEET>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <RUNNING_TITLE> tag to specify a one- or two-line title at the top of the page. Use the *FIRST_PAGE* argument to the <RUNNING_TITLE> tag to begin the title lines on the first page of output, rather than on the page after the current page as is the default.

Use the *OFF* argument to disable any existing running titles created using the <RUNNING_TITLE> tag. These titles will then be disabled for the page on which this tag occurs and on any subsequent pages.

Use the <RUNNING_FEET> tag to create a heading that appears at the bottom of the page. See the <RUNNING_FEET> tag in this chapter for more information.

REPORT Doctype Tag Reference

<RUNNING_TITLE>

The headings established by the <RUNNING_TITLE> and <RUNNING_FEET> tags may be overridden by a subsequent use of the <CHAPTER> tag.

EXAMPLES

The following examples show how to use the <RUNNING_TITLE> tag to create titles and how to disable them.

The following example shows how to use the <RUNNING_TITLE> tag to create the two-line running title "An E. B. Bartz Course:" "Writing Quality Correspondence." Note that because the FIRST_PAGE argument is used, the two-line running title will appear at the top of the first page.

1 <RUNNING_TITLE>(An E. B. Bartz Course:\Writing Quality Correspondence\FIRST_PAGE)
<HEAD>(How to Write a Letter)
<P>
The first thing that you should do in writing a letter is to get
a clean piece of paper and a well-sharpened pencil.

The following example shows how you can disable a running title by using the OFF argument to the <RUNNING_TITLE> tag.

2 <COMMENT>(turn off running titles for the following example page)
<RUNNING_TITLE>(OFF)
<HEAD>(An Example of a Letter)
.
.
.

<SECTION>

Begins a new page and creates a major heading on the left side of that page.

FORMAT <SECTION> (*heading-text* [\ *symbol-name*])

ARGUMENTS *heading-text*
Specifies the text of the section heading.

symbol-name
Specifies a symbol to identify the heading created by the <SECTION> tag. Use this argument if you are planning to reference this heading by its title elsewhere in the document. This argument is provided so that you can reference unnumbered headings.

The *symbol-name* argument can be any text string of 31 characters or less without spaces. This text must contain only the letters A through Z, the numbers 0 through 9 and the underscore (–) character. The *symbol-name* may not begin with an underscore character.

related tags

- <CHEAD>
- <HEAD>
- The global <CHAPTER> tag
- The global <HEAD1> tag
- The global <REFERENCE> tag

restrictions *None.*

required terminator *None.*

DESCRIPTION The <SECTION> tag is one of the three REPORT doctype-specific tags that create unnumbered headings. The <SECTION> tag begins a new output page and creates a major heading on the left-hand side of that page. The other tags in the REPORT doctype that produce unnumbered headings are the <HEAD> and the <CHEAD> tags; these tags do not begin a new page of output.

REPORT Doctype Tag Reference

<SECTION>

EXAMPLE

The following example shows how to use the <SECTION> tag to begin a new page and place an unnumbered heading on that page. Note that this sample omits the *symbol-name* argument to the <SECTION> tag, because the writer will not be referencing this section.

```
<SECTION>(Writing Personal Correspondence)
```

```
<P>
```

```
Writing personal correspondence is more fun and less formal than  
writing business correspondence, but much of the same rules apply.
```

```
<HEAD>(How to Write a Letter)
```

```
<P>
```

```
The first thing that you should do in writing a letter is to get  
a clean piece of paper and a well-sharpened pencil.
```

<SHOW_LEVELS>

Emphasizes text within an outline.

FORMAT

<SHOW_LEVELS> ({ *BOLD*
ITALIC
OFF })

ARGUMENTS

BOLD

Specifies that the *entry-text* arguments given to all subsequent <LEVEL> tags should be output in a bold typeface.

ITALIC

Specifies that the *entry-text* arguments given to all subsequent <LEVEL> tags should be output in an italic typeface.

OFF

Specifies that the *entry-text* arguments given to all subsequent <LEVEL> tags should be output in the standard typeface; this is the default.

related tags

- <OUTLINE>
- <SHOW_LEVELS>

restrictions

Valid only in the context of the <OUTLINE> tag.

required terminator

None.

DESCRIPTION

The <SHOW_LEVELS> tag lets you emphasize the text associated with one or more <LEVEL> tags. Such text is output in a bold typeface if the BOLD argument is used, or in an italic typeface if the ITALIC argument is used.

The OFF argument disables the bolding or italicizing of the text used as an argument to the <LEVEL> tag. If the <SHOW_LEVELS> tag is not used, or if it is specified with the OFF argument, the standard typeface is used within the outline.

See the <LEVEL> tag description in this chapter for more information on the <LEVEL> tag.

REPORT Doctype Tag Reference

<SHOW_LEVELS>

EXAMPLE

The following example shows an outline that uses the <SHOW_LEVELS> tag.

The entry text is italicized using the <SHOW_LEVELS> (ITALIC) tag beginning with the second-level entry "Mammalia" through the fourth-level entry "Primates." The italicized text is then turned off using the OFF argument to the <SHOW_LEVELS> tag.

```
<OUTLINE>(<EMPHASIS>(Maxillary Taxonomy)\An Enumeration of the
Maxillae\from a Dentition Perspective)
<LEVEL>(1\Historical introduction)
<LEVEL>(1\Dentition in various groups of vertebrates)
<LEVEL>(2\Reptilia)
<LEVEL>(3\Histology and development of reptile teeth)
<LEVEL>(4\Survey of forms)
    <COMMENT>( **Italicize the information covered on this weeks test** )
    <SHOW_LEVELS>(ITALIC)
<LEVEL>(2\Mammalia)
<LEVEL>(3\Histology and development of mammalian teeth)
<LEVEL>(3\Survey of forms)
<LEVEL>(4\Primates)
    <COMMENT>( **turn off italicization** )
    <SHOW_LEVELS>(OFF)
<LEVEL>(5\Lemuroidea)
<LEVEL>(5\Anthropoidea)
<LEVEL>(6\Platyrrhini)
<LEVEL>(6\Catarrhini)
<LEVEL>(4\Carnivora)
<LEVEL>(5\Creodonta)
<LEVEL>(5\Fissipedia)
<LEVEL>(6\Aeluroidea)
<LEVEL>(6\Arctoidea)
<LEVEL>(5\Pinnipedia)
<LEVEL>(4\Etc<hellipsis>)
<ENDOUTLINE>
```

<SIGNATURES>

Begins a list of signatures which are to appear in the front matter of a document.

FORMAT <SIGNATURES> [(NEWPAGE)]

ARGUMENTS **NEWPAGE**

Specifies that the signature list is to begin on a new page.

related tags

- <AUTHOR>
- <BYLINE>
- The global <FRONT_MATTER> tag

restrictions

Valid only when following the global <FRONT_MATTER> tag in the REPORT doctype.

required terminator

None.

DESCRIPTION

The <SIGNATURES> tag begins a list of persons who are to sign a document. Each person's name is listed by using the <BYLINE> tag following the <SIGNATURES> tag. The <BYLINE> tag places the name of the person, and additional information about that person (such as their title or affiliation), below a line on which the person is to sign.

See the reference description of the <BYLINE> tag for more information on that tag.

EXAMPLE

The following example shows the <SIGNATURES> tag beginning a list of signature lines. Note how the <BYLINE> tag is used to create each signature line.

```
<FRONT_MATTER>
<TITLE_PAGE>
<TITLE>(The NYUC Simulator Reference Manual)
<REVISION_INFO>(This revision is personally signed.)
<AUTHOR>(Mr. Jones\Research Head, STG Inc.)
<SIGNATURES>
<BYLINE>(Nat Jones\Author)
<BYLINE>(Cecil Mills\Co-author)
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
```

14 SOFTWARE Doctype Tag Reference

This chapter provides reference information on the SOFTWARE doctype tags and templates. Chapter 8 provides the tutorial information on these tags and templates.

This reference chapter divides the SOFTWARE doctype tags into the following groups:

- Tags that can be used anywhere in the SOFTWARE doctype, including inside or outside of the reference templates.
- Tags available in any of the reference templates.
- Tags available only in the Command template.
- Tags available only in the Routine template.
- Tags available only in the Statement template.
- Tags available only in the Tag template.

The program examples in this chapter were processed using a local doctype based on one of the six SOFTWARE doctypes. Your output of these examples may vary depending on the doctype you processed the example under, and whether any doctype modifications have been made to your local installation of VAX DOCUMENT. Each output example is introduced by a form of the sentence, "This example may produce the following output," to remind you that the output examples may vary.

14.1 The SOFTWARE Doctype Tags

The tags described in this section are available for use anywhere in a SOFTWARE doctype.

SOFTWARE Doctype Tag Reference

<ARGDEFLIST>

<ARGDEFLIST>

Begins a definition list of arguments.

FORMAT

<ARGDEFLIST> [({ *alternate-heading*
NOHEAD
NONE })]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading. The default heading provided by VAX DOCUMENT for the <ARGDEFLIST> tag can vary. See the description section for more information on default argument definition list headings.

NOHEAD

Suppresses the output of the default heading for the <ARGDEFLIST> tag.

NONE

Causes the text "None." to be written to indicate that no arguments are available. Note that when using the NONE keyword the <ENDARGDEFLIST> tag should not be used.

related tags

- <ARGITEM>
- <ARGTEXT>
- <ARGDEF>
- <PARAMDEFLIST>
- <QUALDEFLIST>
- <SET_TEMPLATE_HEADING>
- The global <DEFINITION_LIST> tag

required terminator

<ENDARGDEFLIST>

Required unless the NONE keyword is specified as an argument to the <ARGDEFLIST> tag.

restrictions

None.

DESCRIPTION

The <ARGDEFLIST> tag creates a definition list of arguments; it is similar in format and syntax to the global <DEFINITION_LIST> tag. See the *VAX DOCUMENT User Manual, Volume 1* for more information on the <DEFINITION_LIST> tag.

SOFTWARE Doctype Tag Reference

<ARGDEFLIST>

The <ARGDEFLIST> tag enables two tags to create an argument definition list. The <ARGITEM> tag labels the list item being defined, and the <ARGDEF> tag begins the definition of the list item.

When the <ARGDEFLIST> tag is used within the Routine template, the arguments accepted by the <ARGITEM> tag are changed, and the <ARGTEXT> tag is enabled. The change in the arguments accepted by the <ARGITEM> tag and the addition of the <ARGTEXT> tag give you a more structured environment in which to create Routine template argument definition lists. See the descriptions of these tags in this chapter for more information.

When the <ARGDEFLIST> tag is used within the templates, a default heading is provided. This heading can be modified for that single argument definition list using the *alternate-heading* argument. A heading specified in this way overrides any existing default heading.

Use the <SET_TEMPLATE_HEADING> tag to create your own default headings within a reference template. Using this tag modifies the default headings for all subsequent <ARGDEFLIST> tags used in that reference template. See the reference description of the <SET_TEMPLATE_HEADING> tag for more information on that tag.

When the <ARGDEFLIST> tag is used outside of a template, no default heading is defined for it. You can create your own heading for a single argument definition list by specifying that heading as the *alternate-heading* argument.

The default headings for the <ARGDEFLIST> are listed by their context in the following informal table.

Context	Default Heading
Command Template	"Arguments"
Routine Template	"Arguments "
Tag Template	"Arguments"
Statement Template	No default heading
Outside of a Template	No default heading

EXAMPLES

The following examples show various uses of the <ARGDEFLIST> tag. The first example shows an argument definition list used outside of the context of the Routine template. Note the syntax used for the <ARGITEM> tag. The second example shows two argument definition lists used within the routine template.

```
1 <ARGDEFLIST>(Arguments)
  <ARGITEM>(data-1\data-2)
  <ARGDEF>Specifies the arguments through which data is passed to the routine.
  <ENDARGDEFLIST>
```

This example may produce the following output.

SOFTWARE Doctype Tag Reference

<ARGDEFLIST>

ARGUMENTS *data-1* *data-2*

Specifies the arguments through which data is passed to the routine.

The following example shows two argument definition lists used within the Routine template. The first list is coded using the Routine template-specific <ARGITEM> tag syntax. Note the headings produced by the <ARGITEM> tag in the output of this example.

The second argument definition list illustrates a use of the <ARGTEXT> tag. Typically, the <ARGTEXT> tag is used instead of the <ARGITEM> or <ARGDEF> tags.

```
2 <ROUTINE_SECTION>
.
<ARGDEFLIST>
<ARGITEM>(x\floating_point\F_Floating,D_Floating, G_Floating, or H_Floating
\read only\by reference\may also be passed by value)
<ARGDEF>The number for which the square root is desired.
<ENDARGDEFLIST>
<ARGDEFLIST>
<ARGTEXT>
The arguments to the SYS$NONE routine are identical to those used
by the SYS$NULL routine. See the description of the SYS$NULL routine for
more information on these arguments.
<ENDARGTEXT>
<ENDARGDEFLIST>
.
<ENDROUTINE_SECTION>
```

This example may produce the following output.

ARGUMENTS **X**

VMS Usage: **floating_point**
type: **F_Floating,D_Floating, G_Floating, or H_Floating**
access: **read only**
mechanism: **by reference—may also be passed by value**

The number for which the square root is desired.

ARGUMENTS

The arguments to the SYS\$NONE routine are identical to those used by the SYS\$NULL routine. See the description of the SYS\$NULL routine for more information on these arguments.

<ARGDEF>

Begins the text that defines an item in an argument definition list.

FORMAT **<ARGDEF>****ARGUMENTS** *None.***related tags**

- <ARGDEFLIST>
- <ARGITEM>

restrictions

May only be used in the context of the <ARGDEFLIST> tag.

required terminator

None.

DESCRIPTION The <ARGDEF> tag is used within an argument definition list to label the beginning of the text that defines a list item. This text describes the item listed by the previous <ARGITEM> tag. The text begun by the <ARGDEF> tag is terminated by the next <ARGITEM> or <ENDARGDEFLIST> tag.**EXAMPLE**

The following example shows an argument definition list used outside of the routine template.

```
<ARGDEFLIST>(Arguments)
<ARGITEM>(data-1\data-2)
<ARGDEF>Specifies the arguments through which data is passed to the routine.
<ENDARGDEFLIST>
```

This example may produce the following output.

ARGUMENTS ***data-1***
data-2

Specifies the arguments through which data is passed to the routine.

SOFTWARE Doctype Tag Reference

<ARGITEM>

<ARGITEM>

Labels one to seven routine argument items to be defined in an argument definition list outside of the Routine template, or a single routine argument and its attributes within the Routine template.

FORMAT <ARGITEM> (*item-1* [\ *item-2* . . . \ *item-7*])

<ARGITEM> (*arg-name*[\ *usage-information*
 \ *data-type* \ *access*
 \ *mechanism*[\ *mechanism-info*]])

ARGUMENTS *item-n*

Specifies the item in the argument list to be defined. This tag accepts a minimum of one *item-n* argument and a maximum of seven. When more than one *item-n* argument is specified, each subsequent *item-n* argument after the initial argument is formatted under the first argument.

arg-name

Specifies the descriptive name assigned to the argument for reference purposes.

usage-information

Specifies a keyword indicating the category of data to which the argument's value belongs. These keywords are system dependent, and are specified by agreed-upon conventions.

data-type

Specifies the data type of the argument, for example, longword, byte, G_ floating, and so on.

access

Specifies the access applied to the argument; for example, read-only, write-only, and so on.

mechanism

Specifies the mechanism by which the argument is passed; for example, by descriptor, by reference, or by value.

mechanism-info

Specifies additional information that may be appended to the *mechanism* argument output.

related tags

- <ARGDEF>
- <ARGTEXT>
- <ARGDEFLIST>

SOFTWARE Doctype Tag Reference

<ARGITEM>

restrictions

The first syntax listed in the format section is valid within an argument definition list outside of the Routine template. The second syntax listed in the format section is valid only within an argument definition list inside of the Routine template.

required terminator

<ARGDEF>

DESCRIPTION

The <ARGITEM> tag labels items in an argument definition list within the context of the <ARGDEFLIST> tag; however, it accepts one of two sets of arguments, depending on whether or not the <ARGITEM> tag is used within the Routine reference template.

If the <ARGITEM> tag is used outside of the Routine template, it accepts the *item-n* argument, which may be repeated up to seven times, as specified in the first syntax listed in the Format section. This form of the <ARGITEM> tag allows you to label one to seven routine argument items inside of an argument definition list.

If the <ARGITEM> tag is used within the Routine template, it uses the arguments listed in the second syntax listed in the Format section. This form of the <ARGITEM> tag allows you to label a single routine argument and its attributes.

EXAMPLE

See the example in the <ARGDEFLIST> tag description.

SOFTWARE Doctype Tag Reference

<ARGUMENT>

<ARGUMENT>

Emphasizes an argument name within text.

FORMAT <ARGUMENT> (*argument-name*)

ARGUMENTS *argument-name*
Specifies an argument name to be emphasized.

related tags

- <VARIABLE>
- <KEYWORD>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <ARGUMENT> tag causes the *argument-name* to appear in an altered font (such as boldface or italics). This tag does not alter the case of its argument.

EXAMPLE The following example shows a sample use of the <ARGUMENT> tag.

<P>
The <ARGUMENT>(newadr) argument to the \$ADJSTK function provides the address of a longword to receive the updated value.

This example may produce the following output.

The **newadr** argument to the \$ADJSTK function provides the address of a longword to receive the updated value.

SOFTWARE Doctype Tag Reference

<AUTHOR>

EXAMPLE

The following example shows how you can use the <AUTHOR> tag within the front matter of a document. Note how the optional second argument to the <AUTHOR> tag is used to list the position of the author.

```
<FRONT_MATTER>
<TITLE_PAGE>
<TITLE>(The NYUC Simulator Reference Manual)
<ORDER_NUMBER>(AA-Z0000-TE)
<ABSTRACT>
This manual describes the NYUC Simulator.
This program simulates a conversation between three people
by analyzing the syntactic and semantic components of three
related statements, and then synthesizing statements and responses
based upon these original statements.
<ENDABSTRACT>
<REVISION_INFO>(This revision is personally signed.)
<AUTHOR>(Mr. Jones\Research Head, STG Inc.)
<SIGNATURES>
<BYLINE>(Nat Jones\Author)
<DATE>(July 11, 1985)
<PRINT_DATE>(June 1987)
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
```

<BYLINE>

Places a name and other optional information below a ruled line in a signature list processed using the SOFTWARE.SPECIFICATION doctype.

FORMAT **<BYLINE>** (*name* [\ *additional-info*])

ARGUMENTS ***name***
 Specifies the name of the signatory. This name is placed under the beginning of the signature line on the left side of the page.

additional-info

Specifies any additional information about the signatory. This information is placed on the same line as the *name* argument with an em dash (—) between the two arguments.

related tags

-
- <AUTHOR>
 - <SIGNATURES>
 - The global <FRONT_MATTER> tag

restrictions

Available only in the context of the <SIGNATURES> tag.

required terminator

None.

DESCRIPTION The <BYLINE> tag lets you create an approval line in the front matter of a document and place the name of the signatory beneath that line. You can optionally place additional information about the signer by using the *additional-info* argument. Additional information is formatted to the right of the name of the signer, on the same line, separated by an em dash (—).

You can use as many <BYLINE> tags as you want to create approval lines within the front matter of a document, as long as all of these tags follow the <SIGNATURES> tag. You can use the <SIGNATURES> tag to begin all the approval lines on a separate page of the front matter. See the reference description of the <SIGNATURES> tag in this section for more information on that tag.

SOFTWARE Doctype Tag Reference

<BYLINE>

EXAMPLE

The following example shows three occurrences of the <BYLINE> tag. The first two occurrences list the positions of the signers using the optional *additional-info* argument; the third occurrence omits the optional argument. Note that all three tags follow the <SIGNATURES> tag.

```
<FRONT_MATTER>
<TITLE_PAGE>
<TITLE>(The NYUC Simulator Reference Manual)
<REVISION_INFO>(This revision is personally signed.)
<AUTHOR>(Mr. Jones\Research Head, STG Inc.)
<SIGNATURES>
<BYLINE>(Nat Jones\Author)
<BYLINE>(Cecil Mills\Co-author)
<BYLINE>(Matt Smith)
<DATE>(July 11, 1985)
<PRINT_DATE>(June 1987)
<ENDTITLE_PAGE>
<ENDFRONT_MATTER>
```

<CPOS>

Marks the cursor position in a screen display.

FORMAT **<CPOS>** (*placement-character*)

ARGUMENTS ***placement-character***

Specifies the character occupying the position of the cursor. This character can be a blank space.

related tags

- <DISPLAY>
- <KEY>
- <KEY_SEQUENCE>

restrictions

None.

required terminator

None.

DESCRIPTION The <CPOS> tag places a special character within a screen display to indicate the position of the cursor.

EXAMPLE

The following example shows how to use the <CPOS> tag to indicate a cursor's position. In this case the cursor is placed on the letter X in the directory specification [TRXTFILES].

```
<P>
To correct the file specification, move the
cursor to the incorrect letter as shown in the following example:
<DISPLAY>
  $ COPY ABC.TXT [TR<CPOS>(X)TFILES]
<ENDDISPLAY>
```

This example may produce the following output.

To correct the file specification, move the cursor to the incorrect letter as shown in the following example:

```
  $ COPY ABC.TXT [TRXTFILES]
```

SOFTWARE Doctype Tag Reference

<DELETE_KEY>

<DELETE_KEY>

Creates a special character resembling a DELETE key on a keyboard.

FORMAT <DELETE_KEY>

ARGUMENTS *None.*

related tags • <GRAPHIC>
 • <KEY>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <DELETE_KEY> tag creates a special character that represents the delete key used on many types of keyboards.

EXAMPLE The following example shows how you can use the <DELETE_KEY> tag to create the delete key symbol.

<P>
You press the DELETE key (<DELETE_KEY>) to delete the text.

This example may produce the following output.

You press the DELETE key (<X>) to delete the text.

<DISPLAY>

Simulates the appearance and position of characters on a terminal screen.

FORMAT **<DISPLAY>** (*display-text*)

<DISPLAY> [(*KEEP*
WIDE[\ *MAXIMUM*])]

ARGUMENTS ***display-text***

Specifies a display fragment you want to insert into your text. If this argument is not specified, the **<ENDDISPLAY>** tag is required.

KEEP

Specifies that the display example should be placed on the next page rather than breaking it between the current page and the next page. If the display example is longer than a single page, it will be broken between the current page and the next page.

WIDE

Indicates a wide display that may exceed the normal right margin of the text.

MAXIMUM

Indicates that the example may require additional adjustment to fit within the bounds of the text page. May be used only with the **WIDE** keyword. Using this argument may result in the display text being output in a smaller type face.

related tags

- **<SYNTAX>**
 - The global **<INTERACTIVE>** tag
 - The global **<VALID_BREAK>** tag
-

restrictions

You cannot use tab characters, index tags (such as **<X>** and **<Y>**), text element tags (such as **<P>**, **<LIST>**, or **<NOTE>**), or the **<MATH>** tag within any example.

required terminator

<ENDDISPLAY>

Required if the display example is not passed as the *display-text* argument.

SOFTWARE Doctype Tag Reference

<DISPLAY>

DESCRIPTION

Use the <DISPLAY> tag to create display examples either as fragments in text or as extended examples. Such display examples are distinguished typographically from the surrounding text upon output.

If you want to place a display fragment in the text of your document, specify the *display-text* argument as the only argument to the <DISPLAY> tag and do not specify the <ENDDISPLAY> tag. This format is shown first in the format section of the <DISPLAY> tag.

If you have a long display example that you want to appear distinctly separate from the text of your document, place the text for the example between the <DISPLAY> and <ENDDISPLAY> tags. This format is shown second in the format section of the <DISPLAY> tag.

Display examples specified in this way retain all spaces and line breaks as entered. This allows you to position the text in the example so that it appears similar to a terminal screen display. When you specify display examples in this form, you can use the KEEP, WIDE, and MAXIMUM keyword arguments to control the formatting of your example.

If you have an especially long display example, you may want to use the <VALID_BREAK> tag in your example to indicate where a page may break.

EXAMPLES

The following examples show how to use the <DISPLAY> tag.

The first example shows a display example that uses the <DISPLAY> and <ENDDISPLAY> tags. Note the use of the KEEP keyword to ensure that the display example is not broken across the page.

1 <P>The operating system indicates success with the following message:
<DISPLAY>(KEEP)
 Welcome to VAX/VMS Version 4.4
 Username:
<ENDDISPLAY>

This example may produce the following output.

The operating system indicates success with the following message:

```
    Welcome to VAX/VMS Version 4.4  
    Username:
```

The following example shows the <DISPLAY> tag used to create a display fragment in text. Note that the <ENDDISPLAY> tag is omitted and that no keywords to the <DISPLAY> tag can be specified.

2 <P>The message <DISPLAY>(Welcome to VAX/VMS Version 4.4) is issued by default when a user presses RETURN.

This example may produce the following output.

The message *Welcome to VAX/VMS Version 4.4* is issued by default when a user presses RETURN.

<DOCUMENT_ATTRIBUTES>

Enables doctype-specific tags that override the default design format of the SOFTWARE doctype.

FORMAT <DOCUMENT_ATTRIBUTES>

ARGUMENTS *None.*

related tags *None.*

restrictions *None.*

required terminator <ENDDOCUMENT_ATTRIBUTES>

DESCRIPTION The <DOCUMENT_ATTRIBUTES> tag can be used in three doctypes:

- ARTICLE
- REPORT
- SOFTWARE

The <DOCUMENT_ATTRIBUTES> tag enables a group of tags in each of these doctypes that allow you to modify the default format of that doctype. These tags are recognized only within the context of the <DOCUMENT_ATTRIBUTES> tag. If other VAX DOCUMENT tags occur in this context, they are ignored just as if they had occurred within the context of a <COMMENT> tag.

Typically, you use the <DOCUMENT_ATTRIBUTES> tag at the beginning of an input file (or in a file specified using the DOCUMENT /INCLUDE qualifier) to alter the default format of a doctype for the processing of that entire file.

Table 14-1 summarizes the formatting tags enabled by the <DOCUMENT_ATTRIBUTES> tag in each of the three supported doctypes.

SOFTWARE Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

Table 14–1 Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> tag

Formatting Tags	Description
Tags Enabled in the ARTICLE Doctype	
<SET_HEADINGS> (UNNUMBERED) <SET_HEADINGS> (NUMBERED)	<p>The <SET_HEADINGS> tag specifies whether numbered or unnumbered headings are produced by the heading-level tags (<HEAD1> , <HEAD2> , and so on). By default, headings are not numbered in a document processed using the ARTICLE doctype.</p> <p>Use the <SET_HEADINGS> (NUMBERED) tag to specify that your headings should be numbered.</p>
Tags Enabled in the REPORT Doctype	
<SET_PAGE_NUMBERING> (BY_CHAPTER) <SET_PAGE_NUMBERING> (SEQUENTIAL)	<p>The <SET_PAGE_NUMBERING> tag specifies how pages are to be numbered in a document processed using the REPORT doctype. Pages are numbered sequentially by default.</p> <p>If the BY_CHAPTER argument is used, the pages are numbered by chapter. For example, the second page in Chapter 3 would be numbered 3-2. If the BY_CHAPTER argument is used in a document that contains no chapters, the page numbering is sequential.</p> <p>The SEQUENTIAL argument specifies that the pages should be numbered sequentially, as they are by default.</p>
<SET_FORMAL_ELEMENT_NUMBERING> (BY_CHAPTER) <SET_FORMAL_ELEMENT_NUMBERING> (SEQUENTIAL)	<p>The <SET_FORMAL_ELEMENT_NUMBERING> tag specifies how formal tables, figures, and examples are to be numbered in a document processed using the REPORT doctype. By default, formal tables, figures, and examples are numbered sequentially.</p> <p>The BY_CHAPTER argument indicates that formal elements are to be numbered using the chapter prefix; for example, the first formal table in Chapter 4 would be numbered as Table 4–1.</p> <p>The SEQUENTIAL argument indicates that formal elements are to be numbered sequentially throughout the document; for example, the eighth formal table in the document, regardless of what chapter it occurs in, would be numbered as Table 8.</p> <p>If the BY_CHAPTER argument is used in a document that contains no chapters, VAX DOCUMENT uses the default sequential formal element numbering.</p>

SOFTWARE Doctype Tag Reference

<DOCUMENT_ATTRIBUTES>

Table 14–1 (Cont.) Doctype-specific Tags Enabled by the <DOCUMENT_ATTRIBUTES> tag

Formatting Tags	Description
Tags Enabled in the SOFTWARE Doctype	
<SET_RUNNING_TITLES> (BY_HEADONE)	<p>The <SET_RUNNING_TITLES> tag specifies that the running title at the top of each page is composed of two lines: the first line is the title of the chapter, and the second line is the heading text of the current <HEAD1> tag. Note that the argument BY_HEADONE is required.</p> <p>By default, the chapter title is used as a single-line running title.</p>

EXAMPLE

The following example of a file processed under the SOFTWARE doctype shows how a two-line running title is specified using the <SET_RUNNING_TITLES> tag. This title will have the title of the current chapter as the first line, and the heading text of the current <HEAD1> tag as the second line.

```
<DOCUMENT_ATTRIBUTES>
<SET_RUNNING_TITLES>(BY_HEADONE)
<ENDDOCUMENT_ATTRIBUTES>
```

SOFTWARE Doctype Tag Reference

<EXAMPLE_SEQUENCE>

<EXAMPLE_SEQUENCE>

Begins a numbered sequence of informal examples.

FORMAT

<EXAMPLE_SEQUENCE> *[[heading-info]*
[\ NONUMBER]]

ARGUMENTS

heading-info

Consists of one of the following arguments:

- **EXAMPLE**—Causes the singular heading “Example” to be output. Also suppresses numbering of the example. Use this argument when you supply only one example.
- **Heading-text**—Causes the specified text to be used as a heading rather than the default heading “Examples.”
- **NOHEAD**—Suppresses the output of a heading for the section.

NONUMBER

Suppresses numbering of the examples. When **EXAMPLE** is supplied as the first argument, the **NONUMBER** argument is unnecessary.

related tags

- <EXAMPLES_INTRO>
- <EXC>
- <EXI>
- <EXTTEXT>

restrictions

None.

required terminator

<ENDEXAMPLE_SEQUENCE>

DESCRIPTION

The <EXAMPLE_SEQUENCE> tag begins a sequence of informal numbered examples. This tag (and the tags it enables to create the examples) can be used inside or outside of the reference templates.

Even though the examples are numbered, they are generally not referred to by these numbers. Instead, explanatory text directly follows each example. The example sections in this chapter provide examples of output from the <EXAMPLE_SEQUENCE> tag.

By contrast, the global <EXAMPLE> tag begins a formal example that is usually located within a chapter or section that is not formatted by a template. A formal example receives a number and caption, is listed in the table of contents, and can be referred to in a cross-reference.

SOFTWARE Doctype Tag Reference

<EXAMPLE_SEQUENCE>

Following the <EXAMPLE_SEQUENCE> tag, each example begins with either the <EXC> or <EXI> tags. Use the <EXC> tag to present code examples; use the <EXI> tag to present interactive examples. Use the <EXTTEXT> tag to terminate an example begun using the <EXI> or <EXC> tags and to begin the text that explains the example.

For some doctypes the <EXC> and <EXI> tags can use the full page width, which causes the example to begin on the far left of the page. The <EXTTEXT> tag, however, causes the text to be left justified at the normal text margin.

EXAMPLE

The following example shows two short interactive examples, a code example, and their explanatory text. The <S> and <U> tags are used to label the system and user portions of the interaction.

```
<example_sequence>(Debugging Examples)
<exi><s>(DBG> )<u>(SCROLL/LEFT)
<exttext>This command scrolls the screen left by eight spaces or columns.
<exi><s>(DBG> )<u>(SCROLL/UP:4)
<exttext>This command scrolls 4 lines up through the display.
<exc>PSL:  CMP TP FPD IS CURMOD PRVMOD IPL DV FU IV TN Z V C
          0  0  0  0 USER  USER  0  0  0  1  0  0  0  0
                                     !Display formatted the PSL.
                                     !All bits are cleared.
<exttext>
This shows you the contents of the PSL after issuing the EXAMINE command.
<endexample_sequence>
```

This example may produce the following output:

DEBUGGING EXAMPLES

1 DBG> SCROLL/LEFT

This command scrolls the screen left by eight spaces or columns.

2 DBG> SCROLL/UP:4

This command scrolls 4 lines up through the display.

```
3   PSL:  CMP TP FPD IS CURMOD PRVMOD IPL DV FU IV TN Z V C
          0  0  0  0 USER  USER  0  0  0  1  0  0  0  0
                                     !Display formatted the PSL.
                                     !All bits are cleared.
```

This shows you the contents of the PSL after issuing the EXAMINE command.

SOFTWARE Doctype Tag Reference

<EXAMPLES_INTRO>

<EXAMPLES_INTRO>

Provides introductory text before an example.

FORMAT <EXAMPLES_INTRO>

ARGUMENTS *None.*

related tags

- <EXAMPLE_SEQUENCE>
- <EXC>
- <EXI>
- <EXTEXT>

restrictions Available only in the context of the <EXAMPLE_SEQUENCE> tag.

required terminator *None.*

DESCRIPTION Use the <EXAMPLES_INTRO> tag to provide an introduction to numbered examples.

EXAMPLE The following example shows introductory text to two interactive examples and their explanatory text.

```
<EXAMPLE_SEQUENCE>(Debugging Examples)
<EXAMPLES_INTRO>
The following examples show various uses of the DBG Utility:
<EXI><S>(DBG)<U>(SCROLL/LEFT)
<EXTEXT>
This command scrolls the screen left by eight spaces or columns.
<EXI><S>(DBG)<U>(SCROLL/UP:4)
<EXTEXT>
This command scrolls 4 lines up through the display.
<ENDEXAMPLE_SEQUENCE>
```

This example may produce the following output:

DEBUGGING EXAMPLES

The following examples show various uses of the DBG Utility:

1 `DBG>SCROLL/LEFT`

This command scrolls the screen left by eight spaces or columns.

2 `DBG>SCROLL/UP:4`

This command scrolls 4 lines up through the display.

SOFTWARE Doctype Tag Reference

<EXC>

<EXC>

Begins a code example within a series of numbered informal examples.

FORMAT <EXC>

ARGUMENTS *None.*

related tags

- <EXAMPLE_SEQUENCE>
- <EXAMPLES_INTRO>
- <EXI>
- <EXTEXT>
- The global <CODE_EXAMPLE> tag

restrictions

- Valid only in the context of the <EXAMPLE_SEQUENCE> tag.
- The first line of the code example must be on the same source file line as the <EXC> tag.
- Indexing tags such as <X> and <Y> tags are invalid within the context of the code example.

required terminator

<EXTEXT>

DESCRIPTION The <EXC> tag begins a code example within a series of numbered examples. The text following the <EXC> tag is treated as the text of the code example until that example is terminated by the <EXTEXT> tag.

The <EXC> and <EXTEXT> tags can be used much like the global <CODE_EXAMPLE> and <ENDCODE_EXAMPLE> tags, with the exception that when using the <EXC> tag to begin a code example, the first line of the code example text must be on the same source file line as the <EXC> tag. This ensures that the code example is formatted correctly on the page.

EXAMPLE

See the example in the discussion of the <EXAMPLE_SEQUENCE> tag.

<EXI>

Begins an interactive example within a series of numbered informal examples.

FORMAT <EXI>

ARGUMENTS *None.*

- related tags**
- <EXAMPLE_SEQUENCE>
 - <EXC>
 - <EXTEXT>
 - The global <INTERACTIVE> tag
 - The global <S> tag
 - The global <U> tag

-
- restrictions**
- Valid only in the context of the <EXAMPLE_SEQUENCE> tag.
 - The first line of the interactive example must be on the same source file line as the <EXI> tag.

required terminator <EXTEXT>

DESCRIPTION The <EXI> tag begins an interactive example within a series of numbered examples. The text following the <EXI> tag is taken as the text of the interactive example until that example is terminated by the <EXTEXT> tag. Use the global <U> and <S> tags within the context of the <EXI> tag in the same way they are used within the context of the global <INTERACTIVE> tag.

The <EXI> and <EXTEXT> tags can be used just like the <INTERACTIVE> and <ENDINTERACTIVE> tags, with the exception that when using the <EXI> tag to begin an interactive example, the first line of the interactive example text must be on the same source file line as the <EXI> tag. This ensures that the interactive example is formatted correctly on the page.

SOFTWARE Doctype Tag Reference

<EXI>

EXAMPLE

The following example shows a use of the <EXI> tag within the context of the <EXAMPLE_SEQUENCE> tag to begin an interactive example. To produce a space between the DCL prompt and the user-entered command, you should include the space within the argument to the <S> tag.

```
<EXAMPLE_SEQUENCE>(DCL Examples)
<EXI><S>($ )<U>(SHOW TIME)
<S>(16-APR-1984 15:18:44)
<EXTTEXT>This example shows how to use the DCL command SHOW TIME to
request a display of the date and time.
<ENDEXAMPLE_SEQUENCE>
```

This example may produce the following output.

DCL EXAMPLES

```
1 $ SHOW TIME
16-APR-1984 15:18:44
```

This example shows how to use the DCL command SHOW TIME to request a display of the date and time.

<EXTEXT>

Terminates an example and begins an explanation in a sequence of numbered examples.

FORMAT **<EXTEXT>**

ARGUMENTS *None.***related tags**

- <EXAMPLE_SEQUENCE>
- <EXAMPLES_INTRO>
- <EXI>
- <EXC>

restrictions

Can be used only in the context of an <EXAMPLE_SEQUENCE> tag.

required terminator

None.

DESCRIPTION

The <EXTEXT> tag terminates a code example or an interactive example, and begins an explanation of the previous example. Within an <EXAMPLE_SEQUENCE> tag section, each numbered example begins with either the <EXC> or <EXI> tag, depending on whether you want a code example or an interactive example. Both kinds of examples are terminated by the <EXTEXT> tag, which labels the beginning of the text that explains the previous example.

Use the <EXAMPLES_INTRO> tag to place explanatory text before the first example in the example sequence. See the description of the <EXAMPLES_INTRO> tag for more information.

EXAMPLE

See the example in the discussions of the <EXAMPLE_SEQUENCE> and <EXI> tags.

SOFTWARE Doctype Tag Reference

<GRAPHIC>

<GRAPHIC>

Displays terminal graphics characters.

FORMAT <GRAPHIC> (*char-1* \ *char-2*)

ARGUMENTS *char-1*
Specifies the character to be used as the top portion of the graphics character.

char-2
Specifies the character to be used as the bottom portion of the graphics character.

related tags • <KEY>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <GRAPHIC> tag creates a single graphics terminal character (such as the linefeed or formfeed characters) by combining the two characters you specify as arguments. The second character you specify appears lower and adjacent to the first character.

EXAMPLE The following example shows how you can use the <GRAPHIC> tag to create the linefeed and formfeed characters that can appear on a computer terminal.

<P>
The <GRAPHIC>(F\F) and the <GRAPHIC>(L\F)
are two characters that the terminal displays.

This example may produce the following output.

The F_F and the L_F are two characters that the terminal displays.

<KEY>

Depicts a key from a keyboard or keypad graphically.

FORMAT

<KEY> (*key-label-1*[\ *key-label-2*] [\ *BOX*] \ *TEXT*)

ARGUMENTS***key-label-1***

Labels the key.

key-label-2

Stacks a second key label under *key-label-1*.

BOX

Draws a box around the key labels for devices that support this feature. *BOX* is the default key format.

TEXT

Encloses the key labels in angle brackets. This format is useful when you specify keys within a body of text.

related tags

- <GRAPHIC>
- <KEY_NAME>
- <KEY_PLUS>
- <KEY_SEQUENCE>

restrictions

You can only specify the argument *key-label-2* when using the <KEY> tag within a key sequence example. For more information, refer to the <KEY_SEQUENCE> tag in this section.

You can use the *BOX* keyword argument only for devices that support graphics (such as laser printers).

required terminator

None.

DESCRIPTION

The <KEY> tag labels the keys of a keyboard or keypad. If you are using the <KEY> in conjunction with the <KEY_SEQUENCE> tag, you can specify a second label that this tag stacks under the first.

The optional keyword arguments *BOX* and *TEXT* determine how the key labels appear in your document. If you specify *BOX* (the default), this tag draws a box around the labels. If you specify *TEXT*, this tag encloses the labels in angle brackets.

SOFTWARE Doctype Tag Reference

<KEY>

EXAMPLE

In the following example, note that in the context of the <KEY_SEQUENCE> tag, the <KEY> tag accepts two *key-label-n* arguments. Outside of the context of the <KEY_SEQUENCE> tag, it accepts only a single *key-label-n* argument.

Note also that the first <KEY> tag is specified with no keyword argument. This tag uses the default keyword argument BOX. The second <KEY> tag includes the BOX keyword argument to specify BOX formatting. The third <KEY> tag includes the TEXT keyword argument.

```
<P>
You would use the following sequence of keys:
<KEY_SEQUENCE>
<KEY>(Next\Screen) <KEY_PLUS> <KEY>(PF3\BOX)
<ENDKEY_SEQUENCE>
<P>
```

These keys are not associated with the <KEY>(WHITE\TEXT) keys.

This example may produce the following output.

You would use the following sequence of keys:

Next Screen

 +

PF3

These keys are not associated with the <WHITE> keys.

<KEY_NAME>

Emphasizes the name of a key within text.

FORMAT <KEY_NAME> (*key-name*)

ARGUMENTS *key-name*
Specifies the name of the key. This often corresponds to the name on a key label.

-
- related tags**
- <CPOS>
 - <KEY>
 - <KEY_SEQUENCE>
 - <GRAPHIC>

restrictions *None.*

required terminator *None.*

DESCRIPTION The <KEY_NAME> tag alters the appearance of a key name within text using uppercase letters, boldface, or italics.

EXAMPLE In the following example the <KEY-NAME> tag alters the appearance of the GOLD key.

<P>
In EDT, the <KEY_NAME>(GOLD) key changes the function of the other keys on the keypad.

This example may produce the following output:

In EDT, the GOLD key changes the function of the other keys on the keypad.

SOFTWARE Doctype Tag Reference

<KEYPAD>

<KEYPAD>

Specifies an individual keypad diagram and optionally supplies a title for that diagram.

FORMAT

<KEYPAD> [({ *alternate-heading*[\ *DISPLAY*] })]

ARGUMENTS

heading-text

Specifies the text of a heading for a keypad diagram. By default, this tag produces no heading.

DISPLAY

Allows you to specify individual key labels as arguments to the tags <KEYPAD_ROW> and <KEYPAD_ENDROW> .

related tags

- <KEYPAD_ENDROW>
 - <KEYPAD_ROW>
 - <KEYPAD_SECTION>
-

restrictions

Valid only in the context of the <KEYPAD_SECTION> tag.

required terminator

<ENDKEYPAD>

DESCRIPTION

The <KEYPAD> tag allows you to specify an optional heading for a keypad diagram. If you use the *DISPLAY* keyword argument, the <KEYPAD> tag allows you to specify individual key labels as arguments to the tags <KEYPAD_ROW> and <KEYPAD_ENDROW> . For more information, refer to the descriptions of the <KEYPAD_ROW> and <KEYPAD_ENDROW> tags in this chapter.

EXAMPLES

The following examples show two keypads. The first example does not include the *DISPLAY* keyword argument. The second example includes the *DISPLAY* keyword argument. To see other examples using the <KEYPAD> tag, refer to the description of the <KEYPAD_SECTION> tag in this section.

1

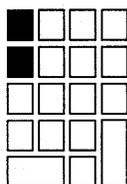
```

<KEYPAD_SECTION>
<KEYPAD>(Using the Command Keypad Function)
<KEYPAD_ROW>(CLOSED\\)
<KEYPAD_ROW>(CLOSED\\)
<KEYPAD_ROW>(\\)
<KEYPAD_ROW>(\\)
<KEYPAD_ENDROW>(\\)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>

```

This example may produce the following output:

Using the Command Keypad Function



2

```

<KEYPAD_SECTION>
<KEYPAD>(The Editing Keypad With Key Labels\DISPLAY)
<KEYPAD_ROW>(PF1\PF2\PF3\PF4)
<KEYPAD_ROW>(7\8\9\-)
<KEYPAD_ROW>(4\5\6\.)
<KEYPAD_ROW>(1\2\3\ )
<KEYPAD_ENDROW>(0\.\ENTER)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>

```

This example may produce the following output:

The Editing Keypad With Key Labels

PF1	PF2	PF3	PF4
7	8	9	-
4	5	6	.
1	2	3	ENTER
0	.		

3

```

<KEYPAD_SECTION>
<KEYPAD>(The Editing Keypad With Key Labels and Blacked-in Keys\DISPLAY)
<KEYPAD_ROW>(CLOSED\PF2\PF3\PF4)
<KEYPAD_ROW>(7\8\9\-)
<KEYPAD_ROW>(CLOSED\5\6\.)
<KEYPAD_ROW>(1\2\3\ )
<KEYPAD_ENDROW>(0\.\ENTER)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>

```

SOFTWARE Doctype Tag Reference

<KEYPAD>

This example may produce the following output:

The Editing Keypad With Key Labels and Blacked-in Keys

	PF2	PF3	PF4
7	8	9	-
	5	6	.
1	2	3	ENTER
0	.		

EXAMPLE

See the example in the discussion of the `<KEYPAD_SECTION>` tag.

SOFTWARE Doctype Tag Reference

<KEYPAD_SECTION>

<KEYPAD_SECTION>

Begins a series of one or more keypad diagrams.

FORMAT

<KEYPAD_SECTION>

related tags

- <KEYPAD>
- <KEYPAD_ENDROW>
- <KEYPAD_ROW>

restrictions

You can only draft files containing keypad diagrams for devices that support graphics (such as laser printers).

required terminator

<ENDKEYPAD_SECTION>

DESCRIPTION

The <KEYPAD_SECTION> tag enables the <KEYPAD>, <KEYPAD_ENDROW>, and <KEYPAD_ROW> tags to graphically depict one or more terminal editing keypads. Each of the keypads (or partial keypads) illustrated in a keypad section must be begun by the <KEYPAD> tag and terminated by the <ENDKEYPAD> tag.

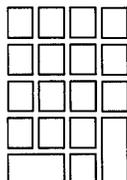
EXAMPLES

The following examples show several uses of the <KEYPAD_SECTION> tag. In the first example, notice how you use the <KEYPAD_ENDROW> to create the large sized keys found on many calculator keypads.

```
1 <KEYPAD_SECTION>
  <KEYPAD>(The VT200 Series Editing Keypad)
  <KEYPAD_ROW>(\\)
  <KEYPAD_ROW>(\\)
  <KEYPAD_ROW>(\\)
  <KEYPAD_ROW>(\\)
  <KEYPAD_ROW>(\\)
  <KEYPAD_ENDROW>(\\)
  <ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output.

The VT200 Series Editing Keypad



SOFTWARE Doctype Tag Reference

<KEYPAD_SECTION>

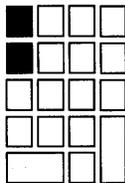
The following example shows two keypads created within a single keypad section. The first keypad shows a command keypad with two keys filled in. The second keypad is an irregularly shaped keypad.

```
2 <KEYPAD_SECTION>
<KEYPAD>(Using the Command Keypad Function)
<KEYPAD_ROW>(CLOSED\\)
<KEYPAD_ROW>(CLOSED\\)
<KEYPAD_ROW>(\\)
<KEYPAD_ROW>(\\)
<KEYPAD_ENDROW>(\\)
<ENDKEYPAD>

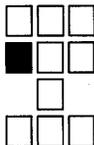
<KEYPAD>(Using the SELECT Key)
<KEYPAD_ROW>(\\NONE)
<KEYPAD_ROW>(CLOSED\\NONE)
<KEYPAD_ROW>(NONE\\NONE\\NONE)
<KEYPAD_ROW>(\\NONE)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

This example may produce the following output.

Using the Command Keypad Function



Using the SELECT Key



The following example shows how to create a keypad with the key names placed on each keypad key. Note how the DISPLAY keyword argument is used with the <KEYPAD> tag.

```
3 <KEYPAD_SECTION>
<KEYPAD>(The Editing Keypad With Key Labels\DISPLAY)
<KEYPAD_ROW>(PF1\PF2\PF3\PF4)
<KEYPAD_ROW>(7\8\9\-)
<KEYPAD_ROW>(4\5\6\,)
<KEYPAD_ROW>(1\2\3\ENTER)
<KEYPAD_ENDROW>(0\.\ENTER)
<ENDKEYPAD>
<ENDKEYPAD_SECTION>
```

SOFTWARE Doctype Tag Reference

<KEYPAD_SECTION>

This example may produce the following output.

The Editing Keypad With Key Labels

PF1	PF2	PF3	PF4
7	8	9	-
4	5	6	,
1	2	3	ENTER
0	.		

<KEY_PLUS>

Creates a plus sign between keys in a key sequence example.

FORMAT <KEY_PLUS>

ARGUMENTS *None.*

related tags

- <KEY>
- <KEY_SEQUENCE>
- <KEY_TYPE>

restrictions

This tag can only be used in the context of the <KEY_SEQUENCE> tag.

required terminator

None.

DESCRIPTION The <KEY_PLUS> tag creates a plus sign between keys in a key sequence example.

EXAMPLE

The following example shows how you can use the <KEY_PLUS> tag to create a plus sign between the BREAK and F10 keys in the context of the <KEY_SEQUENCE> tag.

```
<KEY_SEQUENCE>  
<KEY>(BREAK) <KEY_PLUS> <KEY>(F10) = Exit from Function.  
<ENDKEY_SEQUENCE>
```

This example may produce the following output.

BREAK + **F10** = Exit from Function.

SOFTWARE Doctype Tag Reference

<KEY_SEQUENCE>

<KEY_SEQUENCE>

Begins a section containing one or more key representations.

FORMAT <KEY_SEQUENCE>

ARGUMENTS *None.*

related tags

- <CPOS>
- <GRAPHIC>
- <KEY>
- <KEY_PLUS>
- <KEY_TYPE>

restrictions

Not available within the context of the example-producing tags (<CODE_EXAMPLE> , <DISPLAY> , <SYNTAX> and so on) or the <MATH> tag.

required terminator

<ENDKEY_SEQUENCE>

DESCRIPTION

The <KEY_SEQUENCE> tag labels an informal example of keys and sequences of keys, and enables two tags that make it easier for you to combine keys and text in your document. The two enabled tags are <KEY_PLUS> and <KEY_TYPE> . For more information concerning these tags, refer to the program example in this section, or refer to the <KEY_PLUS> and <KEY_TYPE> tag descriptions in this section.

EXAMPLES

The following examples show how you can create informal examples of keys and key sequences in your documentation. The first example shows how the <KEY_PLUS> and the tag can be used.

The second example shows how other tags can be used in a <KEY_SEQUENCE> tag section and then gives a sample of that output.

1 <KEY_SEQUENCE>
<key>(HELP) = <key>(PF1) <KEY_PLUS> <KEY>(PF2)
<ENDKEY_SEQUENCE>

This example may produce the following output.

HELP = **PF1** + **PF2**

SOFTWARE Doctype Tag Reference

<KEY_SEQUENCE>

2

```
<KEY_SEQUENCE>
The <CPOS>(q)uick brown fox
jumped over the lazy dog.
<KEY>(ADV)<KEY>(WORD)
The quick <CPOS>(b)rown fox
jumped over the lazy dog.
<KEY>(WORD)
The quick brown <CPOS>(f)ox
jumped over the lazy dog.
<KEY>(DEL W)
The quick brown <CPOS>( )
jumped over the lazy dog.
<ENDKEY_SEQUENCE>
```

This example may produce the following output.

The quick brown fox
jumped over the lazy dog.

ADV WORD

The quick brown fox
jumped over the lazy dog.

WORD

The quick brown fox
jumped over the lazy dog.

DEL W

The quick brown
jumped over the lazy dog.

<MESSAGE_SECTION>

Begins a section of error message descriptions.

FORMAT <MESSAGE_SECTION>

ARGUMENTS *None.*

related tags

- <MESSAGE_TYPE>
- <MSG>
- <MSG>
- <MSG_TEXT>

restrictions *None.*

required terminator <ENDMESSAGE_SECTION>

DESCRIPTION The <MESSAGE_SECTION> tag begins a message description section and enables the message tags listed as “related tags” in this tag description.

Messages generally come in one of three forms:

- A message text string only; for example, “System Resources Unavailable.”
- A message text string preceded by a text string identification code; for example, “%DIRECT-W-NOFILES, no files found.”
- A message text string preceded by a numeric string identification code; for example, “%1288374 file lookup failed.”

You select the tags you need to use in the message description section based upon the following criteria:

- The format your messages most closely resemble
- The number of lines each message occupies on the terminal display

Specify the format for your message section by using the <MESSAGE_TYPE> tag. The <MESSAGE_TYPE> tag accepts one of three keywords:

- NOIDENT—Specifies that the message contains no message identification string and that only message text will be specified.
- TEXTIDENT—Specifies that the message contains a text message identification string as well as message text.

SOFTWARE Doctype Tag Reference

<MESSAGE_SECTION>

- NUMIDENT—Specifies that the message contains a numeric message identification string as well as message text.

Select one of two message labeling tags (<MSG> or <MSGS>), based upon the number of messages or message text lines you need to describe in a single tag.

- <MSG> Describes a single message with one or two lines of message text.
- <MSGS> Describes either a single message with up to nine lines of message text or up to four message identification string/message text pairs.

Use the <MSG_TEXT> tag to describe the error messages. Each use of the <MSG_TEXT> tag creates a separate description section that you can label with a single word heading (such as "Facility" or "Severity"). If you do not specify a heading for this tag, it uses the heading, "Explanation:." You can use the <MSG_TEXT> as many times as you find necessary within the error message description section.

To complete your error message section, repeat using the <MSG> (or <MSGS>) and <MSG_TEXT> tags until you describe all of your messages. End the error message section by using the <ENDMESSAGE_SECTION> tag.

EXAMPLES

The following examples illustrate the two syntaxes available with the <MSG> tag and how they are specified. The second example also shows a sample use of the <MSGS> tag so you can compare the output of the <MSG> and <MSGS> tags.

In the first example, the NOIDENT keyword argument is used with the <MESSAGE_TYPE> tag. Therefore, the <MSG> tag only accepts two arguments.

In the second example, the TEXTIDENT keyword is used with the <MESSAGE_TYPE> tag. Therefore, the <MSG> tags in this section accept one message identifier and up to two message text arguments. Note also that the <MSGS> tag used in this example accepts two pairs of message identifier/message text arguments.

```
1 <MESSAGE_SECTION>
  <MESSAGE_TYPE>(NOIDENT)
  <MSG>(error initiating system\initialization file not found)
  <MSG_TEXT>
  The system could not begin operation because it could not find
  the system initialization file.
  <MSG_TEXT>(User Action)
  Check for the existence of the system initialization file.
  If it exists, check that it is in your current default directory.
  <ENDMESSAGE_SECTION>
```

SOFTWARE Doctype Tag Reference

<MESSAGE_SECTION>

This example may produce the following output:

```
error initiating system
initialization file not found
```

Explanation: The system could not begin operation because it could not find the system initialization file.

User Action: Check for the existence of the system initialization file. If it exists, check that it is in your current default directory.

```
2 <MESSAGE_SECTION>
  <MESSAGE_TYPE>(TEXTIDENT)
  <MSG>(BCK-F-BADLNK\Incorrect directory back link\Directory not found)
  <MSG_TEXT>(Facility) VERIFY, Verify Utility
  <MSG_TEXT>(Severity) Fatal
  <MSG_TEXT>
  The Verify Utility could not process your command. Please check the
  syntax of your statement.
  <MSG_TEXT>(User Action)
  Check the syntax of the command, especially the directory specification,
  and reenter the command.
  <MSG>(UAF-E-NAOFIL\unable to open file SYSUAF.DAT\~RMS-E-FNF\file not found)
  <MSG_TEXT>
  The AUTHORIZE Utility could not locate the file SYSUAF.DAT.
  <MSG_TEXT>(User Action)
  Check that your process is currently set to the system default directory,
  SYS$SYSTEM, and then reissue the command.
  <ENDMESSAGE_SECTION>
```

This example may produce the following output.

```
BCK-F-BADLNK, Incorrect directory back link
Directory not found
```

Facility: VERIFY, Verify Utility

Severity: Fatal

Explanation: The Verify Utility could not process your command. Please check the syntax of your statement.

User Action: Check the syntax of the command, especially the directory specification, and reenter the command.

```
UAF-E-NAOFIL, unable to open file SYSUAF.DAT
~RMS-E-FNF, file not found
```

Explanation: The AUTHORIZE Utility could not locate the file SYSUAF.DAT.

User Action: Check that your process is currently set to the system default directory, SYS\$SYSTEM, and then reissue the command.

SOFTWARE Doctype Tag Reference

<MESSAGE_TYPE>

<MESSAGE_TYPE>

Establishes the format for error messages in the context of the <MESSAGE_SECTION> tag.

FORMAT <MESSAGE_TYPE> (*type-keyword*)

ARGUMENTS *type-keyword*

Identifies the error message format and the number of arguments to pass to the <MSG> and <MSGS> tags. These keyword arguments can be any of the following formats:

- NOIDENT — Indicates that only message text will be used as arguments to the <MSG> or <MSGS> tag. This is the default.
 - TEXTIDENT — Indicates that the message has two parts, a text identification string and a line of message text. These two arguments are then required by the <MSG> or <MSGS> tags.
 - NUMIDENT — Indicates that the message has two parts, a numeric identification string and a line of message text. These two arguments are then required by the <MSG> or <MSGS> tags.
-

related tags

- <MESSAGE_SECTION>
 - <MSG>
 - <MSGS>
 - <MSG_TEXT>
-

restrictions

This tag can only be used in the context of a <MESSAGE_SECTION> tag.

required terminator

None.

DESCRIPTION

The <MESSAGE_TYPE> tag establishes the error message format for all subsequent error message descriptions. The format determines the number of arguments passed to the <MSG> and <MSGS> tags.

For a complete description of the message section tags, refer to the description of <MESSAGE_SECTION> in this section.

EXAMPLE

See the example in the discussion of the <MESSAGE_SECTION> tag.

<MSG>

Formats the text of a message within a series of error message descriptions.

FORMAT **<MSG>** (*message-text-1*[\ *message-text-2*])

<MSG> (*message-id* \ *message-text-1*
[\ *message-text-2*])

ARGUMENTS ***message-id***

Specifies a unique message identification string. This argument can be either a text string or a numeric string. You can only specify this keyword argument if you have specified either the NUMIDENT or the TEXTIDENT keyword argument to the <MESSAGE_TYPE> tag.

message-text-1

Is the text of the message.

message-text-2

Is the optional second line of the text of the message.

related tags

- <MESSAGE_SECTION>
- <MESSAGE_TYPE>
- <MSG_TEXT>
- <MSG>

restrictions

This tag can only be used in the context of a <MESSAGE_SECTION> tag.

required terminator

None.

DESCRIPTION

Use the <MSG> tag to specify the parts of a single error message. If you are using the NOIDENT keyword argument to the <MESSAGE_TYPE> tag, or if you do not specify the <MESSAGE_TYPE> tag, you must use the first syntax listed.

If you are using either the TEXTIDENT or NUMIDENT keywords as arguments to the <MESSAGE_TYPE> tag, you must use the second syntax listed.

The *message-id* argument corresponds to the %FAC-S-IDENT portion of a system or application message in the VMS operating system programming environment.

SOFTWARE Doctype Tag Reference

<MSG>

If you are using the NUMIDENT format, the *message-id* argument must be a numeric message identification string of not more than six picas (approximately ten characters) in length. If you are using the TEXTIDENT format, the *message-id* argument has a comma (,) placed between it and the following *message-text* argument.

For a complete description of the message section tags, refer to the description of <MESSAGE_SECTION> in this section.

EXAMPLE

See the example in the discussion of the <MESSAGE_SECTION> tag.

SOFTWARE Doctype Tag Reference

<MSG>

The *message-id* argument corresponds to the %FAC-S-IDENT portion of a system or application message in the VMS operating system programming environment.

If you are using the NUMIDENT format, the *message-id* argument must be a numeric message identification string of not more than six picas (approximately ten characters). If you are using the TEXTIDENT format, the *message-id* argument has a comma (,) placed between it and the following *message-text* argument.

For a complete description of the message section tags, refer to the description of <MESSAGE_SECTION> in this section.

EXAMPLE

See the example in the discussion of the <MESSAGE_SECTION> tag.

<MSG_TEXT>

Labels text that describes a message in a <MESSAGE_SECTION> tag section.

FORMAT <MSG_TEXT> [(*alternate-heading*)]

ARGUMENTS *alternate-heading*
Specifies the label for the text of the message description. This text automatically has a colon (:) appended to the end of it. If you do not specify this argument, the default heading is "Explanation:."

restrictions Valid only within the context of the <MESSAGE_SECTION> tag.

required terminator <ENDMESSAGE_SECTION>
The text labeled by the <MSG_TEXT> tag is terminated either by the next <MSG> or <MSGS> tag, or by the <ENDMESSAGE_SECTION> tag.

DESCRIPTION The <MSG_TEXT> tag labels a message description within an error message section. The text of this description begins after the <MSG_TEXT> tag and continues until the next message section tag is encountered.

If you want to use this tag to label various portions of the message explanation, you can do so by specifying this tag several times using various alternate headings.

For example, first you could have a brief explanation of the message under the default heading "Explanation:." Then you could specify the <MSG_TEXT> tag again, with the alternate heading of "User Action." This use of the <MSG_TEXT> tag labels the tasks the user should perform to correct the condition that caused the error message. Note that when specifying alternate headings, a colon (:) is supplied at the end of the heading.

For a complete description of all the message section tags, refer to the description of the <MESSAGE_SECTION> tag in this section.

EXAMPLE

See the example in the discussion of the <MESSAGE_SECTION> tag.

SOFTWARE Doctype Tag Reference

<PARAMDEF>

<PARAMDEF>

Begins the text that defines an item in a parameter definition list.

FORMAT <PARAMDEF>

ARGUMENTS *None.*

related tags • <PARAMDEFLIST>
 • <PARAMITEM>

restrictions Valid only in the context of the <PARAMDEFLIST> tag.

**required
terminator** *None.*

DESCRIPTION The <PARAMDEF> tag is used within a parameter definition list to label the beginning of the text that defines a list item. This text describes the item listed by the previous <PARAMITEM> tag. The text begun by the <PARAMDEF> tag is terminated by the next <PARAMITEM> or <ENDPARAMDEFLIST> tag.

EXAMPLE

See the example in the discussion of the <PARAMDEFLIST> tag.

<PARAMDEFLIST>

Begins a definition list of parameters or arguments.

FORMAT

<PARAMDEFLIST> [({ *alternate-heading*
NOHEAD
NONE })]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading. The default heading for the <PARAMDEFLIST> tag can vary. See the description section for more information on default parameter definition list headings.

NOHEAD

Suppresses the output of the default heading for the <PARAMDEFLIST> tag.

NONE

Causes the text "None." to be output beneath the heading for the parameter definition list to indicate that no parameters are available. Note that when using the NONE keyword the <ENDPARAMDEFLIST> tag should not be used.

related tags

- <ARGDEFLIST>
- <QUALDEFLIST>
- <PARAMDEF>
- <PARAMITEM>
- <SET_TEMPLATE_HEADING>
- The global <DEFINITION_LIST> tag

required terminator

<ENDPARAMDEFLIST>

Required unless the NONE keyword was specified as an argument to the <PARAMDEFLIST> tag.

restrictions

None.

DESCRIPTION

The <PARAMDEFLIST> tag creates a definition list of parameters or arguments. This tag is available both inside and outside the context of the reference templates.

SOFTWARE Doctype Tag Reference

<PARAMDEFLIST>

The <PARAMDEFLIST> tag is similar in format and syntax to the global <DEFINITION_LIST> tag. See *The VAX DOCUMENT User Manual, Volume 1* for more information on the <DEFINITION_LIST> tag. The <PARAMDEFLIST> tag enables two tags to create a parameter definition list. The <PARAMITEM> tag labels the list item being defined, and the <PARAMDEF> tag begins the definition of the list item.

A default heading is provided when the <PARAMDEFLIST> tag is used within a reference template; no default heading is provided when the <PARAMDEFLIST> tag is used outside a reference template.

You can create your own heading for an individual parameter definition list by specifying that heading as the *alternate-heading* argument. A heading specified in this way overrides any existing default headings.

Use the <SET_TEMPLATE_HEADING> tag to alter the default headings used by all subsequent <PARAMDEFLIST> tags. See the reference description of the <SET_TEMPLATE_HEADING> tag for more information.

The default headings for the <PARAMDEFLIST> are listed by their context in the following informal table.

Context	Default Heading
Command Template	Parameters
Routine Template	Arguments
Tag Template	Arguments
Statement Template	No default heading
Outside of a template	No default heading

EXAMPLES

The following examples show variations on the use of the <PARAMDEFLIST> tag.

This chapter uses the Tag template for its reference descriptions. Within this template, the heading for a parameter definition list is defined as "Arguments." You can see a sample of this heading just after the format section at the start of this tag description.

The following example uses the NOHEAD argument to the <PARAMDEFLIST> tag to suppress the output of a heading within a template. If this example were coded outside of a template, there would be no default heading for the parameter definition list, and so there would be no need to use the NOHEAD argument to suppress a heading.

1 The system maintains logical names and their associated equivalence strings in two types of tables:

```
<PARAMDEFLIST>(NOHEAD)
<PARAMITEM>(process-private)
<PARAMDEF>These tables contain logical names that are available only
to your process.
<PARAMITEM>(shareable)
<PARAMDEF>These tables contain logical names that are available to other
processes on the system.
<ENDPARAMDEFLIST>
```

SOFTWARE Doctype Tag Reference

<PARAMDEFLIST>

This example may produce the following output:

The system maintains logical names and their associated equivalence strings in two types of tables:

process-private

These tables contain logical names that are available only to your process.

shareable

These tables contain logical names that are available to other processes on the system.

The following example shows how to use the global <ALIGN_AFTER> tag for additional formatting flexibility in the parameter definition list. Note that this <PARAMDEFLIST> tag is used within the Command template and so has the default heading "Parameters."

```
2 <COMMAND_SECTION>
.
.
.
<PARAMDEFLIST>
<PARAMITEM>(STATUS:arg\
<ALIGN_AFTER>(STATUS:)COMMAND\
<ALIGN_AFTER>(STATUS:)TASK)
<PARAMDEF>Specifies whether status information is to be returned
from the RUN command.
<ENDPARAMDEFLIST>
.
.
<ENDCOMMAND_SECTION>
```

This example may produce the following output.

PARAMETERS ***STATUS:arg***
COMMAND
TASK

Specifies whether status information is to be returned from the RUN command.

SOFTWARE Doctype Tag Reference

<PARAMITEM>

<PARAMITEM>

Labels one to seven items to be defined in a parameter definition list.

FORMAT

```
<PARAMITEM> (item-1 [ \ item-2 ]  
                    [ \ item-3 ]  
                    [ \ item-4 ]  
                    [ \ item-5 ]  
                    [ \ item-6 ]  
                    [ \ item-7 ] )
```

ARGUMENTS *item-n*

Specifies the item in the parameter list to be defined. This tag accepts a minimum of one *item-n* argument and a maximum of seven *item-n* arguments. When more than one argument is specified, each subsequent *item-n* argument after the initial argument is formatted flush left under the first argument.

related tags

- <PARAMDEFLIST>
- <PARAMDEF>

restrictions

Valid only in the context of the <PARAMDEFLIST> tag.

required terminator

<PARAMDEF>

DESCRIPTION

Use the <PARAMITEM> tag to specify each of the items to be defined in a parameter list. One to seven arguments can be specified as items requiring a single definition in the parameter definition list. If more than one *item-n* argument is specified, the *item-n* arguments are stacked from top to bottom in the order in which they were specified.

If you need to format the *item-n* arguments differently than the default flush left formatting, you can use the global <ALIGN_AFTER> tag. A sample use of this tag is illustrated in the following example. See *The VAX DOCUMENT User Manual, Volume 1* for more information on the <ALIGN_AFTER> tag.

EXAMPLE

The following example shows how to use the global <ALIGN_AFTER> tag within the context of a parameter definition list for special formatting purposes. Note how it is used outside of the <PARAMITEM> tag.

```
<PARAMDEFLIST>(NOHEAD)
<PARAMITEM>(STATUS:arg\
<ALIGN_AFTER>(STATUS:)COMMAND\
<ALIGN_AFTER>(STATUS:)TASK)
<PARAMDEF>Specifies whether status information is to be returned
from the RUN command.
<ENDPARAMDEFLIST>
```

This example produces the following.

STATUS:arg
COMMAND
TASK

Specifies whether status information is to be returned from the RUN command.

SOFTWARE Doctype Tag Reference

<QPAIR>

<QPAIR>

Labels a qualifier pair within a qualifier format list.

FORMAT **<QPAIR>** (*qualifier-name* \ *default-qualifier-name*)

ARGUMENTS ***qualifier-name***
The command qualifier to be listed. A common convention indicates the negative form of the qualifier by placing brackets around the negative prefix, as in “[NO]CHECK.”

default-qualifier-name

The default value of the qualifier.

related tags

- <QUAL_LIST>
 - <QUAL_LIST_HEADS>
-

restrictions

Valid only within the context of a <QUAL_LIST> tag.

required terminator

None.

EXAMPLE

See the example in the discussion of the <QUAL_LIST> tag.

<QUAL_LIST>

Begins a qualifier summary list.

FORMAT <QUAL_LIST> [(*arg-1* \ *arg-2* \ *WIDE*)]

ARGUMENTS *arg-1*

The first argument can be one of the following:

- *alternate-heading*—Causes this text to be used instead of the default heading in column one, “Command Qualifiers.” This default heading can be modified or suppressed by using the <QUAL_LIST_DEFAULT_HEADS> tag.
- NONE—Indicates that there are no qualifiers or defaults and causes the text “None.” to appear under the default headings for columns one and two (“Command Qualifiers” and “Defaults”).
- SPECIAL—Labels an unusual case and causes VAX DOCUMENT to expect a value in the second argument that will set the width of the first qualifier column.

arg-2

The second argument can be one of the following:

- WIDE—Causes the margins to be adjusted to accommodate a wide list.
- *alternate-heading*—Causes this text to be used in column two in place of the default heading “Defaults.” This default heading can be modified or suppressed by using the <QUAL_LIST_DEFAULT_HEADS> tag.
- *column-width*—Provides VAX DOCUMENT with the size in picas of the desired width of the first column (there are six picas to an inch). This value must be a nonzero positive integer and can be used only when the first argument is SPECIAL.

WIDE

Causes the margins to be adjusted to accommodate a wide list. This third argument is used only in the two following cases:

- When the first two arguments have been used to specify headings for columns one and two.
- When the first two arguments have been used to specify a special list and the width of the first column.

related tags

- <QUAL_LIST_HEADS>
- <QUAL_LIST_DEFAULT_HEADS>
- <QPAIR>

SOFTWARE Doctype Tag Reference

<QUAL_LIST>

restrictions

If you use the SPECIAL argument, you must use the <QUAL_LIST_HEADS> tag to specify headings for the qualifier list.

required terminator

<ENDQUAL_LIST>

DESCRIPTION

A qualifier summary list provides a short table listing the qualifiers that are applicable for a system command. It is an optional part of the command template, but it can be used in any context in a SOFTWARE doctype document. In the context of the command template, it provides a brief listing of qualifiers for quick look-up.

The arguments you specify to the <QUAL_LIST> tag provide you with formatting flexibility, and the choice of overriding the default headings that are output.

- When you specify the tag with no arguments, the column widths are set using the doctype design's default settings, and the default headings "Command Qualifiers" and "Defaults" are output.
- When you specify text in the arguments to the <QUAL_LIST> tag, the default headings are replaced:

```
<QUAL_LIST>(Subsystem Qualifier\Comment)
```

This changes the default heading for this qualifier summary list only.

- You can override the default headings for all qualifier summary lists in your document by using the <QUAL_LIST_DEFAULT_HEADS> tag, as follows:

```
<QUAL_LIST_DEFAULT_HEADS>(Subsystem Qualifier\Comment)
```

To suppress either heading, enter the alternate heading text as a null argument:

```
<QUAL_LIST_DEFAULT_HEADS>(\)
```

- If items in the second column of your list result in output that does not fit horizontally in the SOFTWARE design you are using, you can specify the keyword argument WIDE in any of the following positions:

```
<QUAL_LIST>(WIDE)
```

This example uses the default headings, and shifts both columns of the list to the left, as far left as the doctype design allows.

```
<QUAL_LIST>(heading-text\WIDE)
```

This example modifies the default heading for the first column, uses the default heading for the second column, and shifts both columns of the list to the left, as far left as the document design allows.

```
<QUAL_LIST>(heading-text\second-heading-text\WIDE)
```

This example modifies the default headings for both the first and second columns, and shifts both columns of the list to the left, as far left as the doctype design allows.

SOFTWARE Doctype Tag Reference

<QUAL_LIST>

```
<QUAL_LIST>(SPECIAL\18)
<QUAL_LIST_HEADS>(Qualifiers)
```

If the default column width of the first column of the qualifier summary list is too narrow (for example, when qualifier names are long or include lengthy argument specifications), you can widen that column by specifying the SPECIAL keyword as the first argument to the <QUAL_LIST> tag.

The <QUAL_LIST> tag will then accept the *column-width* argument as a second argument that specifies the width of the first column of the list in picas (there are six picas to an inch). If you use the <QUAL_LIST> tag in this manner, you must explicitly enter the headings for the qualifier summary list using the <QUAL_LIST_HEADS> tag.

EXAMPLES

The following example shows how to use the <QUAL_LIST> tag to create a qualifier summary list.

```
1 <QUAL_LIST>
  <QPAIR>(/BOOK\None.)
  <QPAIR>(/[NO]CHECK\CHECK)
  <QPAIR>(/[NO]FAMILY=keyword\NOFAMILY)
  <QPAIR>(/OUTPUT=file-spec\OUTPUT=input-file-name)
  <QPAIR>(/PROFILE=file-spec\None.)
  <QPAIR>(/TYPE=keyword\See text.)
<ENDQUAL_LIST>
```

This example may produce the following output.

Command Qualifiers	Defaults
/BOOK	None.
/[NO]CHECK	/CHECK
/[NO]FAMILY=keyword	/NOFAMILY
/OUTPUT=file-spec	/OUTPUT=input-file-name
/PROFILE=file-spec	None.
/TYPE=keyword	See text.

The following example shows how you can control the headings of the two columns of output by specifying the headings you want in arguments one and two. Compare the results here with the example in the discussion of the <QUAL_LIST_HEADS> tag.

```
2 <QUAL_LIST>(Input Save Set Qualifiers\Default)
  <QPAIR>(/[NO]REWIND\REWIND)
  <QPAIR>(/SAVE_SET\None.)
  <QPAIR>(/SELECT=(file-spec[...])\None.)
<ENDQUAL_LIST>
```

This example may produce the following output:

Input Save Set Qualifiers	Default
/[NO]REWIND	/REWIND
/SAVE_SET	None.
/SELECT=(file-spec[, ...])	None.

<QUAL_LIST_HEADS>

Labels the headings you want to use for one or both of the columns in a qualifier format list when <QUAL_LIST> (SPECIAL) is used in unusual cases for formatting control.

FORMAT <QUAL_LIST_HEADS> (*heading-1* \ *heading-2*)

ARGUMENTS *heading-1*

Specifies the heading text for the left column.

heading-2

Specifies the heading text for the right column. If you do not specify this heading, you will obtain the default heading, "Defaults."

related tags

- <QUAL_LIST>
- <QPAIR>

restrictions

Valid only within the context of a <QUAL_LIST> tag.

required terminator

None.

EXAMPLE

The following example shows how to use the <QUAL_LIST_HEADS> tag to create qualifier summary list headings when the <QUAL_LIST> tag is used with the SPECIAL argument.

```
<P>The following is a partial list of the qualifiers you may use
with the BACKUP command:
<QUAL_LIST>(SPECIAL\18)
<QUAL_LIST_HEADS>(Output File Qualifiers\Qualifier Defaults)
<QPAIR>( /OWNER_UIC[=option] \ /OWNER_UIC=DEFAULT)
<QPAIR>( /REPLACE\None.)
<ENDQUAL_LIST>
```

This example may produce the following output.

The following is a partial list of the qualifiers you may use with the BACKUP command:

Output File Qualifiers
/OWNER_UIC[=option]
/REPLACE

Qualifier Defaults
/OWNER_UIC=DEFAULT
None.

SOFTWARE Doctype Tag Reference

<QUALDEF>

<QUALDEF>

Begins the text that defines an item in a qualifier definition list.

FORMAT <QUALDEF>

ARGUMENTS *None.*

related tags • <QUALDEFLIST>
 • <QUALITEM>

restrictions Valid only within the context of a <QUALDEFLIST> tag.

required terminator *None.*

DESCRIPTION The <QUALDEF> tag is used within a qualifier definition list to label the beginning of the text that defines a list item. This text describes the item listed by the previous <QUALITEM> tag. The text begun by the <QUALDEF> tag is terminated by the next <QUALITEM> or <ENDQUALDEFLIST> tag.

EXAMPLE

See the example in the discussion of the <QUALDEFLIST> tag.

<QUALDEFLIST>

Begins a definition list describing command qualifiers.

FORMAT

<QUALDEFLIST> [({ *alternate-heading*
NOHEAD
NONE })]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading. The default heading provided by VAX DOCUMENT for the <QUALDEFLIST> tag can vary. See the description section for more information on default qualifier definition list headings.

NOHEAD

Suppresses the output of the default heading for the <QUALDEFLIST> tag.

NONE

Causes the text "None." to be output beneath the heading for the qualifier definition list to indicate that no qualifiers are available. Note that when using the NONE keyword the <ENDQUALDEFLIST> tag should not be used.

related tags

- <ARGDEFLIST>
- <PARAMDEFLIST>
- <QUALDEF>
- <QUALITEM>
- <SET_TEMPLATE_HEADING>
- The global <DEFINITION_LIST> tag

required terminator

<ENDQUALDEFLIST>

Required unless the NONE keyword was specified as an argument to the <QUALDEFLIST> tag.

restrictions

None.

DESCRIPTION

The <QUALDEFLIST> tag creates a definition list of command qualifiers; it is similar in format and syntax to the global <DEFINITION_LIST> tag. See *The VAX DOCUMENT User Manual, Volume 1* for more information.

SOFTWARE Doctype Tag Reference

<QUALDEFLIST>

The <QUALDEFLIST> tag enables two tags to create a qualifier definition list. The <QUALITEM> tag labels the list item being defined, and the <QUALDEF> tag begins the definition of the list item. These tags are functionally the same as the <DEFLIST_ITEM> and <DEFLIST_DEF> tags enabled by the global <DEFINITION_LIST> tag. A default heading is provided when the <QUALDEFLIST> tag is used within a reference template; no default heading is provided when the <QUALDEFLIST> tag is used outside of a reference template.

You can create your own heading for an individual qualifier definition list by specifying that heading as the *alternate-heading* argument. A heading specified in this way overrides any existing default headings.

Use the <SET_TEMPLATE_HEADING> tag to alter the default headings used by all subsequent <QUALDEFLIST> tags. See the reference description of the <SET_TEMPLATE_HEADING> tag for more information.

The default headings for the <QUALDEFLIST> are listed by their context in the following informal table.

Context	Default Heading
Command Template	Qualifiers
Tag Template	Qualifiers
Routine Template	No default heading
Statement Template	No default heading
Outside of a template	No default heading

EXAMPLE

The following example shows a qualifier definition list within the context of the Command template.

```
<COMMAND_SECTION>
.
<QUALDEFLIST>
<QUALITEM>/LOG\NOLOG (D)
<QUALDEF>Specifies whether or not output logging should be used.
The default is /NOLOG.
<QUALITEM>/ECHO\NOECHO (D)
<QUALDEF>Specifies whether or not input echoing should be used.
The default is /NOECHO.
<ENDQUALDEFLIST>
.
<ENDCOMMAND_SECTION>
```

SOFTWARE Doctype Tag Reference <QUALDEFLIST>

This example may produce the following output:

QUALIFIERS

/LOG

/NOLOG (D)

Specifies whether or not output logging should be used. The default is /NOLOG.

/ECHO

/NOECHO (D)

Specifies whether or not input echoing should be used. The default is /NOECHO.

SOFTWARE Doctype Tag Reference

<QUALITEM>

<QUALITEM>

Labels one to seven items to be defined in a qualifier definition list.

FORMAT

```
<QUALITEM> (item-1 [ \ item-2 ]  
                  \ item-3  
                  \ item-4  
                  \ item-5  
                  \ item-6  
                  \ item-7 )
```

ARGUMENTS *item-n*

Specifies the item in the qualifier list to be defined. This tag accepts a minimum of one *item-n* argument and a maximum of seven *item-n* arguments. When more than one argument is specified, each subsequent *item-n* argument after the initial argument is formatted flush left under the first argument.

related tags

- <QUALDEFLIST>
- <QUALDEF>

restrictions

Valid only within the context of a <QUALDEFLIST> tag.

required terminator

<QUALDEF>

DESCRIPTION

Use the <QUALITEM> tag to specify each of the items to be defined in a qualifier list. One to seven arguments can be specified as items requiring a single definition in the qualifier definition list. If more than one *item-n* argument is specified, the *item-n* arguments are stacked from top to bottom in the order in which they were specified.

You may find it convenient to use the *item-n* arguments to the <QUALDEFLIST> tag in pairs. The first item in the pair could be the positive form of the qualifier, and the second item could be the negative form of the qualifier (for example, /LOG and /NOLOG). Use the global <ALIGN_AFTER> to format the *item-n* arguments differently than the default flush left formatting.

A sample use of this tag is illustrated in the following example. See The VAX DOCUMENT User Manual, Volume 1 for more information on the <ALIGN_AFTER> tag.

SOFTWARE Doctype Tag Reference

<QUALITEM>

EXAMPLE

The following examples show several uses of the <QUALITEM> tag. The first two uses of the <QUALITEM> tag show how positive and negative forms of a qualifier may be grouped together. The third use of the <QUALITEM> tag shows how the global <ALIGN_AFTER> tag can be used to achieve special formatting.

```
<QUALDEFLIST>
<QUALITEM>/LOG\NOLOG (D)
<QUALDEF>Specifies whether or not output logging should be used.
The default is /NOLOG.
<QUALITEM>/ECHO\NOECHO (D)
<QUALDEF>Specifies whether or not input echoing should be used.
The default is /NOECHO.
<QUALITEM>/DEVICE=device-type\
<ALIGN_AFTER>(DEVICE=)VT100\
<ALIGN_AFTER>(DEVICE=)VT220\
<QUALDEF>Specifies the type of device to be used.
<ENDQUALDEFLIST>
```

This example may produce the following output.

QUALIFIERS

/LOG

/NOLOG (D)

Specifies whether or not output logging should be used. The default is /NOLOG.

/ECHO

/NOECHO (D)

Specifies whether or not input echoing should be used. The default is /NOECHO.

/DEVICE=device-type

VT100

VT220

Specifies the type of device to be used.

SOFTWARE Doctype Tag Reference

<RUNNING_FEET>

<RUNNING_FEET>

Creates a single line heading at the bottom of each page in a document processed using the SOFTWARE.SPECIFICATION doctype.

FORMAT <RUNNING_FEET> (*title-text*)

ARGUMENTS *title-text*

Specifies the text to be used as a running heading at the foot of the page.

related tags

- <RUNNING_TITLE>

restrictions

Available only in the SOFTWARE.SPECIFICATION doctype.

required terminator

None.

DESCRIPTION

Use the <RUNNING_FEET> tag to place a heading at the bottom of every page. This heading is called a "footer" because it appears at the foot of the page. When the same footer is used for several pages, the footers are collectively called "running feet."

This tag accepts one argument, which is the text which should appear as the heading at the bottom of the page. This text is output exactly as entered, including spacing and capitalization.

Use the <RUNNING_TITLE> tag to create a heading at the top of the page. See the reference description of the <RUNNING_TITLE> tag for more information on that tag.

EXAMPLE

The following example shows how to use the <RUNNING_FEET> tag to place the heading "Getting the Piece of Paper" at the bottom of each page. Note that the running footer will be output exactly as entered.

```
<RUNNING_FEET>(Getting the Piece of Paper)
<HEAD2>(Getting the Piece of Paper)
<P>
You can buy clean paper in most major supermarkets, department stores,
and hardware stores. You should try to get ruled paper so that
your letter will be neat and easy to read.
```

<RUNNING_TITLE>

Creates a one- or two-line running heading at the top of each page in a document processed using the SOFTWARE.SPECIFICATION doctype.

FORMAT

<RUNNING_TITLE> ({ *OFF*
title-1 [*\ title-2*]
[*\ FIRST_PAGE*] })

ARGUMENTS***title-1***

Specifies the text of a running title. If a two-line title is specified, this title is output on the upper title line.

title-2

Specifies the optional bottom line of a running title that has two lines.

FIRST_PAGE

Specifies that the running title is to begin output on the first output page. If this keyword is not specified, the running title is output on the page after the current page.

OFF

Specifies that any existing running titles created using the <RUNNING_TITLE> tag should be disabled for the page on which this tag occurs and on any subsequent pages.

related tags

- <RUNNING_FEET>

restrictions

Valid only in the SOFTWARE.SPECIFICATION doctype.

required terminator

None.

DESCRIPTION

Use the <RUNNING_TITLE> tag to specify a one- or two-line title at the top of the page. Use the FIRST_PAGE argument to the <RUNNING_TITLE> tag to begin the title lines on the first page of output, rather than on the page after the current page as is the default.

Use the OFF argument to disable any existing running titles created using the <RUNNING_TITLE> tag. These titles will then be disabled for the page on which this tag occurs and on any subsequent pages.

Use the <RUNNING_FEET> tag to create a heading that appears at the bottom of the page. See the reference description of the <RUNNING_FEET> tag for more information on that tag.

SOFTWARE Doctype Tag Reference

<RUNNING_TITLE>

EXAMPLES

The following example shows how to use the <RUNNING_TITLE> tag to create the two-line running title "An E. B. Bartz Course:" and "Writing Quality Correspondence." Since the FIRST_PAGE argument is used, the two-line running title will appear at the top of the first page.

1 <RUNNING_TITLE>(An E. B. Bartz Course:\Writing Quality Correspondence\FIRST_PAGE)
<HEAD1>(How to Write a Letter)
<P>
The first thing that you should do in writing a letter is to get
a clean piece of paper and a well-sharpened pencil.

The following example shows how you can disable a running title by using the OFF argument to the <RUNNING_TITLE> tag.

2 <COMMENT>(turn off running titles for the following example page)
<RUNNING_TITLE>(OFF)
<HEAD>(An Example of a Letter)
.
.
.

<SIGNATURES>

Begins a list of signatures that are to appear in the front matter portion of a document processed using the SOFTWARE.SPECIFICATION doctype.

FORMAT <SIGNATURES> [(NEWPAGE)]

ARGUMENTS **NEWPAGE**
Specifies that the signature list is to begin on a new page.

related tags

- <AUTHOR>
- <BYLINE>
- The global <FRONT_MATTER> tag

restrictions Available only in the SOFTWARE.SPECIFICATION doctype following the global <FRONT_MATTER> tag.

required terminator *None.*

DESCRIPTION The <SIGNATURES> tag begins a list of persons who are to sign a document. Each person's name is listed by using the <BYLINE> tag following the <SIGNATURES> tag. The <BYLINE> tag places the name of the person, and additional information about that person (such as his or her title or affiliation), below a line on which the person is to sign.

See the reference description of the <BYLINE> tag for more information on that tag.

EXAMPLE

See the example in the description of the <BYLINE> tag.

SOFTWARE Doctype Tag Reference

<SYNTAX>

<SYNTAX>

Allows you to use special characters to describe language syntaxes.

FORMAT

<SYNTAX> [({ *heading-text* [\ *WIDE*] })]

ARGUMENTS

heading-text

Specifies a heading. The doctype controls the font used to display the heading. By default, this tag has no heading. You may wish to create a heading using the `<SYNTAX_DEFAULT_HEAD>` tag.

WIDE

Specifies that the syntax statement can exceed the normal right margin of the text. If you are using doctype designs that indent the text body, a wide syntax example will extend into the left margin.

related tags

- `<DISPLAY>`
 - `<SYNTAX_DEFAULT_HEAD>`
 - The global `<CODE_EXAMPLE>` tag
 - The global `<FORMAT>` tag
-

restrictions

You cannot use tab characters, index tags (such as the `<X>` and `<Y>` tags), or text element tags (such as `<P>`, `<LIST>`, or `<NOTE>`) within this type of example.

required terminator

`<ENDSYNTAX>`

DESCRIPTION

The `<SYNTAX>` tag allows you to accurately describe language syntax. Languages can include programming languages, command languages, application defined languages, and so forth. This tag also separates the syntax example from the remaining text, retains blank spaces and open lines, and labels the example (if you specified one) using a doctype-specific font different from the current text font.

EXAMPLE

The following example shows how to use the <SYNTAX> tag to describe a language's syntax.

```
<P>The COPY command has the following syntax:  
<SYNTAX>  
    COPY input-file output-file  
<ENDSYNTAX>
```

This example may produce the following output.

The COPY command has the following syntax:

```
COPY input-file output-file
```

SOFTWARE Doctype Tag Reference

<SYNTAX_DEFAULT_HEAD>

<SYNTAX_DEFAULT_HEAD>

Creates a default heading for the <SYNTAX> tag.

FORMAT

<SYNTAX_DEFAULT_HEAD> ({ *heading-text* }
 OFF)

ARGUMENTS

heading-text

Specifies a heading to be used by all subsequent <SYNTAX> tags. The doctype controls the font used to display this heading. By default, the <SYNTAX> tag has no heading.

OFF

Disables any heading established by a previous use of the <SYNTAX_DEFAULT_HEAD> tag.

related tags

- <SYNTAX>

restrictions

None.

required terminator

None.

DESCRIPTION

Use the <SYNTAX_DEFAULT_HEAD> tag to create a default heading for subsequent uses of the <SYNTAX> tag. Use the <SYNTAX_DEFAULT_HEAD> tag to create the same default heading above each use of the <SYNTAX> tag.

You can disable all subsequent default headings by specifying the *OFF* argument to the <SYNTAX_DEFAULT_HEAD> tag.

EXAMPLES

The following example shows how the <SYNTAX_DEFAULT_HEAD> tag can be used to create a default heading for all subsequent <SYNTAX> tags.

```
❏ <P>
The COPY command has the following general syntax:
<SYNTAX>
COPY input-file output-file
<ENDSYNTAX>

<COMMENT>(Set up default headings for syntax statements....)
<SYNTAX_DEFAULT_HEAD>(What the User Types)
<P>
An actual user would type the following:
<SYNTAX>
$ COPY MYFILE.TXT NEWFILE.TXT
<ENDSYNTAX>
```

SOFTWARE Doctype Tag Reference

<SYNTAX_DEFAULT_HEAD>

This example may produce the following output:

The COPY command has the following general syntax:

```
COPY input-file output-file
```

An actual user would type the following:

What the User Types

```
$ COPY MYFILE.TXT NEWFILE.TXT
```

The following example shows how to disable <SYNTAX_DEFAULT_HEAD> tag for all subsequent <SYNTAX> tags.

```
2 <COMMENT>(Set up default headings for syntax statements....)
  <SYNTAX_DEFAULT_HEAD>(What the User Types)
  <P>
  An actual user would type the following:
  <SYNTAX>
  $ COPY MYFILE.TXT NEWFILE.TXT
  <ENDSYNTAX>

  <COMMENT>(Disable default headings for syntax statements....)
  <SYNTAX_DEFAULT_HEAD>(OFF)

  <P>The following is a semantic statement of the COPY operation.
  <SYNTAX>
  COPY [the existing file specification to] [the new file specification]
  <ENDSYNTAX>
```

This example may produce the following output:

An actual user would type the following:

What the User Types

```
$ COPY MYFILE.TXT NEWFILE.TXT
```

The following is a semantic statement of the COPY operation.

```
COPY [the existing file specification to] [the new file specification]
```

14.2 Tags Common to all Reference Templates

The tags in this section are available in all SOFTWARE doctype reference templates.

This section describes the tags you can use within any of the SOFTWARE doctype templates. You can use these tags after you use the template-enabling tags (such as `<COMMAND_SECTION>` and `<ROUTINE_SECTION>`) and before you use the template-disabling tags (such as `<ENDCOMMAND_SECTION>` and `<ENDROUTINE_SECTION>`).

If you are unfamiliar with the use of the SOFTWARE doctype templates, refer to Chapter 8 for more information.

<DESCRIPTION>

Begins a section of descriptive text providing detailed information on the current reference element.

FORMAT <DESCRIPTION> [(*alternate-heading*)]

ARGUMENTS *alternate-heading*
Specifies a heading. The default heading is "Description." For information on how to modify the default headings for all subsequent <DESCRIPTION> tags, refer to the description of the <SET_TEMPLATE_HEADING> tag in this section.

related tags

- <OVERVIEW>
- <SET_TEMPLATE_HEADING>
- Any reference element tag: <COMMAND>, <ROUTINE>, <STATEMENT>, <FUNCTION>, or <SDML_TAG>

restrictions Available only in the context of a reference section tag (<COMMAND_SECTION>, <ROUTINE_SECTION>, <STATEMENT_SECTION>, or <TAG_SECTION>).

required terminator <ENDDescription>

DESCRIPTION The <DESCRIPTION> tag separates and labels detailed descriptive text concerning the current reference element (command, routine, and so on). This text can describe the following aspects of the reference element:

- Suggested use
- Special considerations
- Use with other reference elements

You should not use a <P> tag immediately after the <DESCRIPTION> tag. The <DESCRIPTION> tag generates the initial open line so that you need only begin typing the first paragraph of text.

SOFTWARE Doctype Tag Reference

<DESCRIPTION>

EXAMPLE

The following example shows a sample use of the <OVERVIEW> and <DESCRIPTION> tags within the context of the Command template. The Command template <COMMAND> tag and the global <FORMAT> and <PARAMDEFLIST> tags are included to make a more complete example. Note the inclusion of the reference element tag (<COMMAND>) forces a page break by default.

```
<COMMAND_SECTION>
<COMMAND>(SET TIME)
<OVERVIEW>
Resets the system time to the time specified as a parameter to this command.
<ENDOVERVIEW>
<FORMAT>
<FCMD>(SET TIME) <FPARMS>(time-specification)
<ENDFORMAT>
<PARAMDEFLIST>
<PARAMITEM>(time-specification)
<PARAMDEF> Specifies the time to which the system should be set.
<ENDPARAMDEFLIST>
<DESCRIPTION>
The SET TIME command lets you reset the system clock on your VMS system.
This command is generally used to make allowances for daylight savings time
and other such events.
<ENDDescription>
<ENDCOMMAND_SECTION>
```

This example may produce the following output:

SET TIME

Resets the system time to the time specified as a parameter to this command.

FORMAT **SET TIME** *time-specification*

PARAMETERS *time-specification*
Specifies the time to which the system should be set.

DESCRIPTION The SET TIME command lets you reset the system clock on your VMS system. This command is generally used to make allowances for daylight savings time and other such events.

SOFTWARE Doctype Tag Reference

<OVERVIEW>

<OVERVIEW>

Provides a summary description of a reference element.

FORMAT

<OVERVIEW>

related tags

- <DESCRIPTION>
-

restrictions

Can only be used in the context of a SOFTWARE reference template.

required terminator

<ENDOVERVIEW>

DESCRIPTION

The <OVERVIEW> tag lets you create a subsection that typically contains a brief definition of the reference element. Use the <DESCRIPTION> tag to create a separate subsection that contains more detailed information to the user concerning the reference element. See the description of the <DESCRIPTION> tag for more information.

You do not need to use a <P> tag immediately after the <OVERVIEW> tag. The <OVERVIEW> tag generates the initial open line; you need only type the first paragraph of text.

EXAMPLE

See the example in the discussion of the <DESCRIPTION> tag in this section.

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_HEADING>

headings defined for templates control the actual case to be output (all lowercase, all uppercase, or retained case).

related tags

- <SET_TEMPLATE_PARA>
- <SET_TEMPLATE_LIST>
- <SET_TEMPLATE_TABLE>

restrictions

Valid only within the context of a reference template.

required terminator

None.

EXAMPLE

In the following example, the default heading text associated with the <PARAMDEFLIST> tag is redefined within the context of a tag template to have the heading "Function Types."

```
<TAG_SECTION>(Functions)
<SET_TEMPLATE_HEADING>(PARAMDEFLIST\Function Types)
<PARAMDEFLIST>
<PARAMITEM>(XYZ)
<PARAMDEF> The XYZ argument returns the current software status code.
<ENDPARAMDEFLIST>
```

This example may produce the following output.

FUNCTION TYPES

XYZ
The XYZ argument returns the current software status code.

<SET_TEMPLATE_LIST>

Creates a user-defined set of tags for listing information.

FORMAT **<SET_TEMPLATE_LIST>** (*list-tag-name*
 \ *default-list-heading*
 \ *list-item-tag-name*
 \ *list-type*
 [\ *heading-level*])

ARGUMENTS ***list-tag-name***

Specifies the name of the tag that will introduce the list of items within the list template being defined. This tag name must be a valid tag name less than 28 characters; it must not be the same as an existing SDML tag name.

default-list-heading

Specifies the default text heading to be output by the list template being defined.

item-tag-name

Specifies the name of the tag to be used to indicate individual items in the list being defined. This name must be a valid tag name.

list-type

Specifies the keyword that indicates the type of list the list being defined is based on. These keywords create the same list format as the same keywords used with the global <LIST> tag. These keywords are as follows:

SIMPLE
NUMBERED
UNNUMBERED

heading-level

Specifies the template heading level to be associated with the tag. Valid values are 1 and 2, indicating a primary template heading or a secondary template heading. If this argument is not specified, the value defaults to 2, and the secondary template heading is used.

related tags

- <SET_TEMPLATE_PARA>
 - <SET_TEMPLATE_HEADING>
 - <SET_TEMPLATE_TABLE>
 - The global <LIST> tag
-

restrictions

Can only be used within the context of a <COMMAND_SECTION>, <ROUTINE_SECTION>, <STATEMENT_SECTION>, or <TAG_SECTION> tag.

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_LIST>

required terminator

None.

EXAMPLES

The following examples in this section assume that the following tag has been specified to enable the tags <SHOPPING_LIST>, <SITEM>, and <ENDSHOPPING_LIST> tags.

```
<SET_TEMPLATE_LIST>(SHOPPING_LIST\Shopping List\SITEM\NUMBERED)
```

The following example generates the default heading "Shopping List," and starts a numbered list. Each <SITEM> tag generates another numeric item in the list, as follows:

1 <SHOPPING_LIST>
<SITEM>one item
<SITEM>second
<ENDSHOPPING_LIST>

This example may produce the following output:

shopping list

```
1 one item
2 second
```

The following example shows how to override the default heading shopping list heading and produce the text "None."

2 <SHOPPING_LIST>(Bloomingdales\NONE)

This example may produce the following output:

bloomingdales

None.

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_PARA>

1 <SET_TEMPLATE_PARA>(SIDE_EFFECTS\Side Effects:)

```
<SIDE_EFFECTS>  
Modifying the arguments to the PLACE command changes the positioning of the  
page number.  
<ENDSIDE_EFFECTS>
```

This example may produce the following output.

side effects:

Modifying the arguments to the PLACE command changes the positioning of the page number.

The following example shows how to specify the template tag <SIDE_EFFECTS> with the alternate heading "Desired Effect=".

2 <SIDE_EFFECTS>(Desired Effect=)
Modifying the arguments to the PLACE command changes the positioning of the page number.
<ENDSIDE_EFFECTS>

This example may produce the following output.

desired effect=

Modifying the arguments to the PLACE command changes the positioning of the page number.

The following example shows how to specify the NONE keyword to the <SIDE_EFFECTS> template tag.

3 <SIDE_EFFECTS>(NONE)

This example may produce the following output.

side effects:

None.

<SET_TEMPLATE_TABLE>

Defines a set of template tags for setting information in two- or three-column lists.

FORMAT **<SET_TEMPLATE_TABLE>** (*table-tag-name*
 default-table-heading
 table-row-tag-name
 column-count
 column-widths
 [*table-column-headings*])

ARGUMENTS ***table-tag-name***

Specifies the user-defined name of the tag that begins the user-defined table. This tag name must be a valid tag name less than 28 characters; it must not be the same as an existing SDML tag name.

default-table-heading

Specifies a default heading to be output over the entire user-defined table. This heading may be overridden by an alternate heading specified as an argument to the **<table-tag-name>** tag.

table-row-tag-name

Specifies the name of the tag to be used to indicate individual table rows in the table being defined. This name must be a valid tag name. For example, if the *table-row-tag-name* argument is specified as SAMP_ROW, the individual table row tag will be **<SAMP_ROW>**. The user-defined tag created by this argument is similar to the global **<TABLE_ROW>** tag.

column-count

Specifies the number of columns in the user-defined table. The accepted arguments are as follows:

- 2 — Specifies that the table is to have two columns.
- 3 — Specifies that the table is to have three columns.

column-widths

Specifies the approximate widths of the table columns. The width of the last table column is determined by VAX DOCUMENT, so, if you specify a two-column table, you must specify only a single column width argument as shown in the following code example:

```
<SET_TEMPLATE_TABLE>(KEYVALS\Keyword Values\KEYVAL\2\10\Keyword\Value)
```

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_TABLE>

If you specify a three-column table, you must specify two column-width arguments as shown in the following code example:

```
<SET_TEMPLATE_TABLE>(KEYVAL_TABLE\Keyword Ranges\KEYVAL\3\10\10\Keyword\High\Lower)
```

table-column-headings

Specifies the default headings for each column in the user-defined table. If you specify a two-column table, you can specify up to two heading arguments. If you specify a three-column table, you can specify up to three heading arguments.

related tags

- <SET_TEMPLATE_LIST>
- <SET_TEMPLATE_PARA>

restrictions

Valid only within the context of a reference template.

required terminator

None.

DESCRIPTION

Use the <SET_TEMPLATE_TABLE> tag to create your own user-defined set of tags for making a table with optional headings within a template. Tables created using this tag may have either two or three columns. This tag requires five arguments and accepts the optional *table-column-headings* argument.

If you choose to omit the *table-column-headings* argument, then the table will not have any column headings and will not output rules within the table.

EXAMPLES

In the following example the <SET_TEMPLATE_TABLE> tag sets the default heading to be "best songs," enables a two-column table, sets the text supplied to the two <45RPM> tags in the table, and then terminates the table.

```
1 <SET_TEMPLATE_TABLE>(RECORDTABLE\Best Songs\45RPM\2\12\Performer\Song Title)
<RECORDTABLE>
<45RPM>(Sinatra\Strangers in the Night)
<45RPM>(Moody Blues\Nights in White Satin)
<ENDRECORDTABLE>
```

This example may produce the following output.

best songs

Performer	Song Title
Sinatra	Strangers in the Night
Moody Blues	Nights in White Satin

The following example shows how to specify the NONE keyword to the <RECORDTABLE> tag.

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_TABLE>

2 <RECORDTABLE> (NONE)

This example may produce the following output.

best songs

None.

The following example shows how to define a three column table with headings. The tags defined are the <OPCODES>, <ENDOPCODES> and <OPS> tags.

3 <SET_TEMPLATE_TABLE> (OPCODES\codes\OPS\3\6\2-byte\3-byte\4-byte)
<OPCODES>
<OPS>(abc\def\ghi)
<ENDOPCODES>

This example may produce the following output.

codes

2-byte	3-byte	4-byte
abc	def	ghi

14.3 The Command Reference Template Tags

This section alphabetically describes the tags available in the Command template. Use these tags to create a command reference section. In most manuals, a command section is an encyclopedic reference section that describes each command the software system offers, along with the command's format, its arguments, and examples of its use.

SOFTWARE Doctype Tag Reference

<COMMAND>

EXAMPLES

The following example shows a command section begun using the <COMMAND_SECTION> tag. Within this command section the <COMMAND> tag is used to begin the command description for the OPEN command.

1 <COMMAND_SECTION>
<COMMAND>(OPEN)
<OVERVIEW>

In the following example, the <COMMAND> tag has two arguments. The command name, CLOSE, appears at the beginning of the command description. The text specified in the second argument, "Close a File," is printed on the same line as the command name, separated by an em dash (—).

2 <COMMAND_SECTION>
<COMMAND>(CLOSE\Close a File)

SOFTWARE Doctype Tag Reference

<COMMAND_SECTION>

- <RESTRICTIONS>
- <PROMPTS>
- <DESCRIPTION>
- <QUALDEFLIST>
- <EXAMPLE_SEQUENCE>

restrictions

None.

required terminator

<ENDCOMMAND_SECTION>

DESCRIPTION

The <COMMAND_SECTION> tag begins the command reference template. You can place a command section within a chapter or an appendix, or following a part page (that is, within a document section begun with the <PART_PAGE> tag). You code a command section in a chapter or an appendix in the same manner; command sections in parts are handled differently.

If your command section follows a part page, and you include text between the part page and the command section, you should specify the NEWPAGE keyword as the third argument to the <COMMAND_SECTION> tag. This causes the command section to begin on a new page. The following code fragment shows a command section that begins on a new page:

```
<COMMAND_SECTION>(\CD\NEWPAGE)
<HEAD1>(Command Format)
```

When you use the <COMMAND_SECTION> tag within a chapter or an appendix, and you want to place text after the command section in that chapter or appendix, you must end the command section with the <ENDCOMMAND_SECTION> tag and place the text after that tag. This text will by default begin on a new page of output.

Specify the NONENPAGE argument to the <ENDCOMMAND_SECTION> tag if you do not want the text to begin on a new page of output. The following code fragment shows the end of a command section that specifies that the subsequent text not be placed on a new page:

```
<ENDCOMMAND_SECTION>(NONENPAGE)
```

When the <ENDCOMMAND_SECTION> tag is specified within the context of a chapter or appendix, it resets the default running titles to those in effect for the chapter or appendix, so the last page of the last command description in the command section may not carry the last command's name as the running heading. Instead it may carry the running title used by the chapter or appendix.

The <COMMAND_SECTION> tag can be used more than once in a document. By specifying arguments to that tag, and by using the <SET_TEMPLATE_COMMAND> tag to specify additional attributes, you can tailor command sections to the specific requirements of your documentation.

EXAMPLES

The following example shows how to begin a command section within a document part.

```
1 <PART>
  <PART_PAGE>
  <TITLE>(Part III\Command Dictionary)
  <ENDPART_PAGE>(RENUMBER)
  <COMMAND_SECTION>(Command Dictionary\CD)
  <SET_TEMPLATE_COMMAND>(DCL_COMMAND)

  <DCL_COMMAND>(GOTO)
  <OVERVIEW>
  Transfers control to a labeled statement in a command procedure.
  <ENDOVERVIEW>
  .
  .
  .
  <ENDCOMMAND_SECTION>
```

The tags in the previous example perform the following functions:

- The global <PART> tag begins the part.
- The global <PART_PAGE> tag creates a part page.
- The global <TITLE> tag is used in the context of the <PART_PAGE> tag to create a title on the part page.
- The RENUMBER argument to the global <ENDPART_PAGE> tag specifies that the pages should be renumbered beginning with the part page. This causes the first page of text following the part page to be numbered page 3 (Page 1 is the unnumbered page the part page title is placed on, page 2 is the back of page 1, and page 3 is the first numbered page after the part page).
- The <COMMAND_SECTION> tag begins the command section and specifies the running title "Command Dictionary" as the running title for the command section. If the <SET_TEMPLATE_COMMAND> tag were used with the DOUBLERUNNINGHEADS argument, the title "Command Dictionary" would be used as the top running title.

The <COMMAND_SECTION> tag also specifies that the prefix "CD" should be used to construct numbers for pages and for formal figures, tables, and examples within the command section (for example, CD-11, CD-32, Table CD-1, Example CD-2, and so on).

- The <SET_TEMPLATE_COMMAND> tag specifies that all command descriptions in this command section will be identified using the tag <DCL_COMMAND> rather than the default tag <COMMAND>. The <DCL_COMMAND> tag will have the default attributes of the <COMMAND> tag.

The following example shows how you can create a command section in which each command description (begun with a <COMMAND> tag) is in a separate SDML file, and all these descriptions are included into a primary command description file. For example, the file MYCOMMANDS.SDML contains the following SDML tags:

```
<INCLUDE>(CLOSE.SDML)
<INCLUDE>(OPEN.SDML)
<INCLUDE>(READ.SDML)
<INCLUDE>(WRITE.SDML)
```

SOFTWARE Doctype Tag Reference

<COMMAND_SECTION>

Each of the included files contains one command reference description begun with a <COMMAND> tag. For these files to process correctly, they must be preceded with the <COMMAND_SECTION> tag that enables the <COMMAND> tag. These files can have the necessary tags processed before them by specifying the /INCLUDE qualifier on the command line to include a startup definition file.

2 <COMMAND_SECTION>(Command Dictionary\CD)
<SET_TEMPLATE_COMMAND>(COMMAND\DOUBLERUNNINGHEADS)

If this startup file were named COMMAND_DICT_STARTUP.SDML, it could be included using the DOCUMENT /INCLUDE qualifier as in the following example:

```
$ DOCUMENT mycommands SOFT.REF LN03 /INCLUDE=COMMAND_DICT_STARTUP.SDML
```

When each individual file in MYCOMMANDS.SDML is processed, the correct sequence of tags will be read in to begin the command section.

You can process multiple files together by using the <INCLUDE> tag to include them into a single master file (such as MYCOMMANDS.SDML), or you can include them into a book build profile.

You use the <ELEMENT> tags to include multiple files into a profile. For example, the book build profile file CDPRO.SDML could contain the following tags:

```
<PROFILE>  
<ELEMENT>(CLOSE.SDML)  
<ELEMENT>(OPEN.SDML)  
<ELEMENT>(READ.SDML)  
<ELEMENT>(WRITE.SDML) <COMMENT>(contains <ENDCOMMAND_SECTION> tag)  
<ENDPROFILE>
```

Note that the profile file should include the <ENDCOMMAND_SECTION> tag in the appropriate file, so that the template will be terminated and the book build will process correctly.

<FCMD>

Specifies a command or command keyword and an optional parameter list within the context of the <FORMAT> tag.

FORMAT <FCMD> (*command-keyword*[\ *parameter-list*])

ARGUMENTS *command-keyword*

Specifies the command or command keyword. This text is output in the left portion of the format description.

parameter-list

Specifies the one or more parameters of the command or command keyword. This text is output to the right of the command in the format description with no space between the *command-keyword* and the *parameter-list* text.

Parameters you specify using the *parameter-list* argument are output differently than parameters specified using the <FPARMS> tag. See the examples in this tag description for illustrations of these differences.

related tags

- <COMMAND_SECTION>
- <FORMAT>
- <FPARMS>
- <FPARM>
- <STATEMENT_FORMAT>

restrictions

- Valid only within the context of the <FORMAT> or <STATEMENT_FORMAT> tags.
- If you do not provide a second argument to the <FCMD> tag, you should explicitly declare the absence of parameters by using the <FPARMS> tag as follows:

<FCMD>(COMMAND-KEYWORD) <FPARMS>()

If you do not specify a second argument to the <FCMD> tag and the <FPARMS> tag is not specified, VAX DOCUMENT issues a warning message.

required terminator

None.

SOFTWARE Doctype Tag Reference

<FCMD>

DESCRIPTION

Use the <FCMD> tag to label a command or command keyword within the context of the <FORMAT> tag. Use the *parameter-list* argument to this tag to create a list of one or more parameters that are formatted with no space between the command keyword and the parameter list.

Use the <FPARMS> tag in conjunction with the <FCMD> tag to create a command format with the command keyword and the one or more parameters separated by a space.

If the text of the *parameter-list* argument does not fit on a single line in the format section, the text formatter selects suitable line breaks based on the presence of spaces in the *parameter-list* text. Hyphenated text is not broken across lines.

EXAMPLES

The following examples show various uses of the <FCMD> tag.

The following example specifies a command keyword with no parameters. Use the <FPARMS> after the <FCMD> to explicitly specify the absence of any parameters.

1

```
<FORMAT>
<FCMD>(EXIT) <FPARMS>()
<ENDFORMAT>
```

This example may produce the following output.

FORMAT

EXIT

The following example also specifies a command keyword with no parameters. This coding however specifies a null second argument to the <FCMD> tag rather than using the <FPARMS> tag to explicitly specify the absence of any parameters.

2

```
<FORMAT>
<FCMD>(EXIT\ )
<ENDFORMAT>
```

This example may produce the following output.

FORMAT

EXIT

The following example specifies both the *command-keyword* and the *parameter-list* arguments to the <FCMD> tag. Note in the output sample how these two arguments are formatted together with no intervening spaces.

3

```
<FORMAT>
<FCMD>(F$ELEMENT\ (element-number, delimiter, string))
<ENDFORMAT>
```

This example may produce the following output.

FORMAT

F\$ELEMENT(*element-number, delimiter, string*)

SOFTWARE Doctype Tag Reference

<FCMD>

The following example specifies a command keyword with a list of two parameters using the <FCMD> and <FPARMS> tags. Note that coding the parameters using the <FPARMS> tag results in a space being output between the command keyword and the first parameter.

```
4 <FORMAT>
  <FCMD>(APPEND) <FPARMS>(input-file-spec output-file-spec)
  <ENDFORMAT>
```

This example may produce the following output.

FORMAT **APPEND** *input-file-spec output-file-spec*

The following example specifies the <FCMD> tag using the global <KEYWORD> tag as part of the *parameter-list* argument text. Note how the global <KEYWORD> tag alters the output. The output of this example is shown in Output Sample 5.

```
5 <FORMAT>
  <FCMD>(SET PROTECTION\[=code]<keyword>(/DEFAULT))
  <ENDFORMAT>
```

This example may produce the following output.

FORMAT **SET PROTECTION**[*=code*]**//DEFAULT**

SOFTWARE Doctype Tag Reference

<FORMAT>

<FORMAT>

Begins a section that highlights the syntax of a command, including its keywords and parameters.

FORMAT

<FORMAT> [(*alternate-heading*)]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading for this use of the <FORMAT> tag. The default heading provided by VAX DOCUMENT is "Format." See the <SET_TEMPLATE_HEADING> tag description for information on modifying the default headings for all <FORMAT> tags.

related tags

- <COMMAND_SECTION>
- <FCMD>
- <FPARMS>
- <FPARM>
- <STATEMENT_FORMAT>

required terminator

<ENDFORMAT>

DESCRIPTION

The <FORMAT> tag creates sections of text that describe the format of a command, routine, or tag. It also enables the <FCMD>, <FPARMS>, and <FPARM> tags, which label specific portions of a format statement.

The <FORMAT> tag is a global tag and is not restricted to command sections. The <FORMAT> tag enables additional tags in the context of the <ROUTINE_SECTION> and <TAG_SECTION> tags for specialized format descriptions. These additional tags are not available in the context of the <COMMAND_SECTION> tag.

You can use the <FORMAT> tag and the tags it enables in a variety of ways to represent the syntax of a command. The following list of code examples show some of the most regularly used format section tag combinations:

- <FCMD>(command-keyword) <FPARMS>(parameter-list)

This is the standard form, in which the command keyword and its parameter list are separated by a blank space. If the parameter list must be printed on more than one line, additional lines are aligned at the beginning of the parameter list.

- <FCMD>(command-keyword\parameter-list)

This form should be used for command functions in which a command and its parameters are not separated by blank spaces.

- <FCMD>(command-keyword) <FPARMS>(parameter-1) <FPARM>(parameter-2)

This form is useful for commands with long parameter lists and with parameter names that are either long or need additional information.

EXAMPLES

The following example shows how to use the <FCMD> and <FPARMS> tags to present a command format.

1 <FORMAT>
<FCMD>(CANCEL BREAK) <FPARMS>([address-expression[,hellipsis]])
<ENDFORMAT>

This example may produce the following output.

FORMAT **CANCEL BREAK** *[address-expression[, . . .]]*

The following example uses multiple formats for a single command, with a special heading for each format.

2 <FORMAT>(Deleting all Logical Names)
<FCMD>(ASN) <FPARMS>(= logical-name [keywords])
<ENDFORMAT>
<FORMAT>(Format for Deletion)
<FCMD>(ASN) <FPARMS>(=)
<ENDFORMAT>

This example may produce the following output.

DELETING **ASN** = *logical-name [keywords]*
ALL LOGICAL
NAMES

FORMAT FOR **ASN** =
DELETION

SOFTWARE Doctype Tag Reference

<FPARM>

<FPARM>

Adds a single parameter line to a list of parameters in a command syntax format section.

FORMAT <FPARM> (*parameter-name*)

ARGUMENTS *parameter-name*

Specifies a single-line item to be formatted in a format parameter list.

related tags

- <COMMAND_SECTION>
- <FORMAT>
- <FCMD>
- <FPARMS>
- <STATEMENT_FORMAT>

restrictions

Can only be specified following an <FCMD> tag and <FPARMS> tag sequence in a format section.

required terminator

None.

EXAMPLE

The following example shows a sample use of the <FPARM> tag.

```
<FORMAT>
<fcmd>(SET TERMINAL) <fparms>([[device-name[:]])
<fparm>( /DEVICE_TYPE=<list>(stacked\braces)
      <le><keyword>(UNKNOWN)
      <le><keyword>(LA34)
      <le><keyword>(VT100)
      <le><keyword>(PRO_SERIES)<endlist>)
<fparm>( <list>(stacked\brackets)
      <le>/LINE_EDITING
      <le>/NOLINE_EDITING<endlist>)
<fparm>( <list>(stacked\brackets)
      <le>/PASSALL
      <le>/NOPASSALL<endlist>)
<ENDFORMAT>
```

SOFTWARE Doctype Tag Reference

<FPARM>

This example may produce the following output:

FORMAT	SET TERMINAL	<i>[device-name[:]]</i> /DEVICE_TYPE= { UNKNOWN LA34 VT100 PRO_SERIES }[/LINE_EDITING /NOLINE_EDITING][/PASSALL /NOPASSALL
---------------	---------------------	---

SOFTWARE Doctype Tag Reference

<FPARMS>

<FPARMS>

Specifies the parameters to a command or keyword.

FORMAT <FPARMS> (*parameter-list*)

ARGUMENTS *parameter-list*

Specifies one or more parameters to a command or keyword.

You can also use this command to explicitly declare that a command or keyword marked with the <FCMD> tag has no parameters, as shown in this example.

```
<FCMD>(EXIT) <FPARMS>()
```

Parameters you specify using the <FPARMS> tag are printed differently than parameters specified in the *parameter-list* argument to the <FCMD> tag. See the examples in this tag description for illustrations of the different output styles.

related tags

- <COMMAND_SECTION>
 - <FORMAT>
 - <FCMD>
 - <FPARM>
 - <STATEMENT_FORMAT>
-

restrictions

The <FPARMS> tag should be used with the <FCMD> tag in the context of the <FORMAT> tag.

required terminator

None.

DESCRIPTION

The <FPARMS> tag identifies the parameters to a command or keyword. The command or keyword is identified with the tag <FCMD>. These tags should only be used in a format section. That is, they should only occur following the <FORMAT> tag and before the <ENDFORMAT>.

When you use this tag to define a parameter list that contains multiple words or text strings, you should include blank spaces between the words and text strings (including punctuation) according to the syntax rules of the command you are describing.

If the text of the *parameter-list* argument will not fit on a single line in the format section, the text formatter selects suitable line breaks based on the presence of spaces in the *parameter-list* text. Hyphenated text is not broken across lines.

SOFTWARE Doctype Tag Reference

<FPARMS>

Use the <FPARM> tag to select explicit line breaks in a parameter list.

EXAMPLES

The following example uses the <FPARMS> tag to list additional command keywords within the parameters portion of a command format.

1 <FCMD>(ON)
<FPARMS>(condition <keyword>(THEN [\$]) command)

This example may produce the following output.

FORMAT

ON *condition* **THEN** [**\$**] *command*

The following example shows how to format a line in which command parameters appear before the command keyword.

2 <FCMD>()<FPARMS>(input-specifier output-specifier<keyword>(OVERLAY))

This example may produce the following output.

FORMAT

input-specifier output-specifier/OVERLAY

2 <PROMPTS>
<PROMPT>(Old password:\old-password\7)
<PROMPT>(New password:\new-password\7)
<PROMPT>(Verification:\new-password\7)
<ENDPROMPTS>

This example may produce the following output:

prompts

Old password: old-password
New password: new-password
Verification: new-password

SOFTWARE Doctype Tag Reference

<PROMPTS>

<PROMPTS>

Begins a summary of interactive prompts.

FORMAT

<PROMPTS> [({ *alternate-heading*
NONE
alternate-heading \ NONE })]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading for this use of the <PROMPTS> tag. The default heading provided by VAX DOCUMENT is "Prompts." If you want to use a different heading, specify it here. Also see the <SET_TEMPLATE_HEADING> tag description for information on modifying the default headings for all the <PROMPTS> tags.

NONE

Indicates that there are no prompts for this command. If you use the NONE keyword, do not use the <ENDPROMPTS> tag to end the <PROMPTS> list.

related tags

- <COMMAND_SECTION>
- <PROMPT>

required terminator

<ENDPROMPTS>

EXAMPLES

The following example illustrates a command with a single prompt.

1 <PROMPTS>(Prompt)
<PROMPT>(File:\filespec)
<ENDPROMPTS>

This example may produce the following output.

prompt

File: filespec

The following example illustrates a command with no prompts. Note that the terminator, <ENDPROMPTS>, is not specified.

2 <PROMPTS>(NONE)

This example may produce the following output.

prompts

None.

<RESTRICTIONS>

Labels the restrictions on the use of a reference element within a SOFTWARE reference template.

FORMAT

<RESTRICTIONS> [({ *alternate-heading*
NONE
LIST
alternate-heading \ LIST })]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading for this use of the <RESTRICTIONS> tag. The default heading provided by VAX DOCUMENT is "Restrictions." If you want to use a different heading, specify it here. Also see the <SET_TEMPLATE_HEADING> tag description for information on modifying the default headings for all <RESTRICTIONS> tags.

NONE

Indicates that there are no restrictions on the use of the command. If you use the NONE keyword, you should not use the <ENDRESTRICTIONS> tag to end the <RESTRICTIONS> list.

LIST

Marks the beginning of a list of restrictions. If you use this argument, list each restriction following the tag <RITEM> .

related tags

- <COMMAND_SECTION>
- <RITEM>

restrictions

If NONE is specified, do not specify <ENDRESTRICTIONS> .

required terminator

<ENDRESTRICTIONS>

EXAMPLES

The following example shows how to set a simple paragraph of text for the restrictions section using the <RESTRICTIONS> tag.

```

1 <RESTRICTIONS>
    You must have OPER privilege to execute this command.
  <ENDRESTRICTIONS>
```

This example may produce the following output.

restrictions

You must have OPER privilege to execute this command.

SOFTWARE Doctype Tag Reference

<RESTRICTIONS>

The following example uses the NONE keyword to indicate that there are no restrictions on the use of the command.

2 <RESTRICTIONS>(NONE)

This example may produce the following output.

restrictions

None.

The following example illustrates a set of restrictions in a list. Note the use of the LIST argument and how it enables the <RITEM> tag.

3 <RESTRICTIONS>(LIST)
<RITEM>When you use /EDIT, no other qualifiers are allowed.
<RITEM>The /ALL qualifier requires OPER privileges.
<ENDRESTRICTIONS>

This example may produce the following output.

restrictions

- When you use /EDIT, no other qualifiers are allowed.
- The /ALL qualifier requires OPER privileges.

<RETURN_VALUE>

Labels a character string return value.

FORMAT **<RETURN_VALUE>** [(*alternate-heading*)]

ARGUMENTS ***alternate-heading***
Specifies a heading to override the current default text heading for this use of the <RETURN_VALUE> tag. The default heading provided by VAX DOCUMENT is "Return Value." See the reference description of the <SET_TEMPLATE_HEADING> tag for information on how to modify the default headings for all <RETURN_VALUE> tags.

related tags *None.*

restrictions Can only be used in the context of the Command template.

required terminator <ENDRETURN_VALUE>

EXAMPLE The following example shows how to use the <RETURN_VALUE> tag.

```
<RETURN_VALUE>
abc-def-ghi
<ENDRETURN_VALUE>
```

This example may produce the following output.

return value abc-def-ghi

SOFTWARE Doctype Tag Reference

<RITEM>

<RITEM>

Labels an item in a list of restrictions.

FORMAT

<RITEM>

related tags

- <RESTRICTIONS>

restrictions

Can only be used in the context of the <RESTRICTIONS> tag.

required terminator

None.

EXAMPLE

The following example shows how to create a list of restrictions. Note how the <RESTRICTIONS> tag is specified with the LIST keyword argument. Note how this enables the <RITEM> tag.

```
<RESTRICTIONS>(LIST)
<RITEM>You must not unplug this appliance while it is operating.
<RITEM>This appliance should not be immersed in water.
<ENDRESTRICTIONS>
```

This example may produce the following output.

restrictions

- You must not unplug this appliance while it is operating.
- This appliance should not be immersed in water.

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_COMMAND>

DESCRIPTION The <SET_TEMPLATE_COMMAND> tag lets you create a tag with the functions of the <COMMAND> tag, but with a name that is more meaningful to you.

This tag also lets you change the default attributes associated with the <COMMAND> tag, or give your new tag different attributes than the <COMMAND> tag.

EXAMPLES

1 <COMMAND_SECTION>(FDL Commands\NEWPAGE)
<SET_TEMPLATE_COMMAND>(FDL_COMMAND\NONEWPAGE\STACK\DOUBLERUNNINGHEADS)
<FDL_COMMAND>(STARTUP_FILE\STF)

These tags begin a command section and define the tag <FDL_COMMAND> as the command descriptor tag. The attributes set are as follows:

- Commands do not start on new pages.
- When two arguments are specified to the tag <FDL_COMMAND>, the arguments are stacked. That is, the second argument is printed under the first.
- Each page carries a two-line running title. The top line is "FDL Commands" and the second line is the name of the command description that is current at the top of the page.

Note that this command sequence is most appropriate for a series of command descriptions that occur within a chapter.

2 <PART>
<PART_PAGE>
<TITLE>(PART II\XYZ Commands)
<ENDPART_PAGE>(RENUMBER)
<COMMAND_SECTION>(XYZ Commands\XYZ)
<SET_TEMPLATE_COMMAND>(XYZ_COMMAND)

These tags begin a document part in which only command descriptions occur. The RENUMBER argument on the <ENDPART_PAGE> tag specifies that the first page of the new Part (the part page itself) should be numbered page 1. The <COMMAND_SECTION> tag sets the folio prefix to XYZ; that is, pages will be numbered XYZ-3, XYZ-4, and so on.

No attributes are specified for the command descriptions produced by <XYZ_COMMAND>. These commands will have default attributes. The first command will start on a new page. Since RENUMBER was specified on <ENDPART_PAGE>, the first command will be on the page numbered XYZ-3.

<SET_TEMPLATE_SUBCOMMAND>

Changes the name of the <SUBCOMMAND> tag to the name you specify, and specifies formatting attributes for the new tag.

FORMAT <SET_TEMPLATE_SUBCOMMAND>(*tag-name*
[\ *NONEWPAGE*])

ARGUMENTS *tag-name*

Specifies the name of the template tag being defined. This tag name must be a valid tag name less than 31 characters; it must not be the same as an existing SDML tag name other than "SUBCOMMAND" (which is the default tag name).

NONEWPAGE

Specifies that the corresponding subcommand reference descriptions should not start on a new page of output.

related tags

- <SUBCOMMAND>
- <SUBCOMMAND_SECTION>

restrictions

This tag is only valid within the context of the <COMMAND_SECTION> tag.

required terminator

None.

DESCRIPTION The <SET_TEMPLATE_SUBCOMMAND> tag lets you change the name of the <SUBCOMMAND> tag. This helps you make the coding of your source files more meaningful.

EXAMPLE

The following example shows how to create a subcommand section within a command section. The subcommand section is used to describe keywords associated with the command.

```
<SUBCOMMAND_SECTION>(Qualifiers)
<SET_TEMPLATE_SUBCOMMAND>(QUALIFIER)
<QUALIFIER>/OUTPUT)
<overview>
```

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_SUBCOMMAND>

The following default attributes are set in the previous example:

- 1** Each subcommand description that begins with the `<QUALIFIER>` tag starts on a new page of output.
- 2** Each page carries a two-line running title. The top line is "Qualifiers" and the second line is the name of the subcommand description that is current at the top of the page.

<SUBCOMMAND>

Begins a new subcommand description.

FORMAT <SUBCOMMAND> (*subcommand-name*)

ARGUMENTS *subcommand-name*
Names the command described in the subcommand section.

related tags • <SET_TEMPLATE_SUBCOMMAND>
 • <SUBCOMMAND_SECTION>

required terminator *None.*

DESCRIPTION The <SUBCOMMAND> tag begins a subcommand description section. Subcommand sections these format defaults:

- Each <SUBCOMMAND> tag begins a new page of output.
- The subcommand name appears as the single running title on each page.

To change the name of the <SUBCOMMAND> tag, or to change its default attributes see the description of the <SET_TEMPLATE_SUBCOMMAND> tag.

EXAMPLES

1 <SUBCOMMAND_SECTION>
 <SUBCOMMAND>(OPEN)
 <OVERVIEW>

In this example, the <SUBCOMMAND_SECTION> tag enables the tags for a subcommand description. The description of the subcommand OPEN will, by default, have the following attributes:

- 1 The subcommand description begins on a new page.
- 2 If the subcommand carries for more than a page, the name "OPEN" is carried as a running top title on each page.

SOFTWARE Doctype Tag Reference

<SUBCOMMAND>

2 <SUBCOMMAND_SECTION>(File System Subcommands)
<SUBCOMMAND>(CLOSE)

In this example the <SUBCOMMAND_SECTION> tag starts a subcommand section and gives it the title "File System Subcommands." In this command section, until the <ENDSUBCOMMAND_SECTION> tag, each page of output carries this title at the top of page, with the name of the current subcommand just below it.

SOFTWARE Doctype Tag Reference

<SUBCOMMAND_SECTION_HEAD>

<SUBCOMMAND_SECTION_HEAD>

Specifies the heading for text that precedes a subcommand section.

FORMAT <SUBCOMMAND_SECTION_HEAD> (*heading*)

ARGUMENTS *heading*
Specifies a heading that precedes introductory text for the subcommand section.

related tags

- <SUBCOMMAND>
- <SET_TEMPLATE_SUBCOMMAND>

restrictions This tag is only valid within the context of the <SUBCOMMAND_SECTION> tag.

required terminator *None.*

DESCRIPTION The <SUBCOMMAND_SECTION_HEAD> tag lets you provide introductory text for a section of descriptions of commands that are subordinate to a specific command description.

EXAMPLE This example illustrates a subcommand section within a command section. The subcommand section is used to describe subordinate commands.

```
<SUBCOMMAND_SECTION>(File System Subcommands\NEWPAGE)
<SUBCOMMAND_SECTION_HEAD>(Subcommand Descriptions)
<p>This section provides information about each of the subcommands
you can enter while you are conversing with the file subsystem.
<SUBCOMMAND>(CLOSE)
<overview>
```

14.4 The Routine Reference Template Tags

This section alphabetically describes the tags available in the Routine template. Use these tags to create a routine reference section. In most manuals, a routine reference section is an encyclopedic reference section that describes each routine the software system offers, the routine's format, its arguments, and examples of its use.

SOFTWARE Doctype Tag Reference

<ARGTEXT>

<ARGTEXT>

Labels definition text in an argument definition list which replaces the information contained in a pair of <ARGITEM> and <ARGDEF> tags.

FORMAT

<ARGTEXT>

related tags

- <ARGDEFLIST>
- <ARGDEF>
- <ARGITEM>

restrictions

Valid only within the context of the <ARGDEFLIST> tag within the Routine template.

required terminator

<ENDARGTEXT>

DESCRIPTION

The <ARGTEXT> tag labels definition text in an argument definition list which replaces the information contained in a pair of <ARGITEM> and <ARGDEF> tags. This tag is most useful when the arguments to be described are already listed elsewhere in the reference documentation.

EXAMPLE

The following example shows how to use the <ARGTEXT> tag to replace a pair of <ARGDEF> and <ARGITEM> tags. Note that this tag is restricted to argument definition lists used in the context of the Routine template.

```
<routine_section>
```

```
<argdeflist>
```

```
<argtext>
```

```
The arguments to the SYS$NONE routine are identical to those used  
by the SYS$NULL routine. See the description of the SYS$NULL routine for  
more information on these arguments.
```

```
<endargtext>
```

```
<endargdeflist>
```

```
<endroutine_section>
```

This example may produce the following output.

ARGUMENTS

The arguments to the SYS\$NONE routine are identical to those used by the SYS\$NULL routine. See the description of the SYS\$NULL routine for more information on these arguments.

<FARG>

Adds a single argument line to a list of arguments in a routine syntax format section.

FORMAT <FARG> (*argument-name*)

ARGUMENTS *argument-name*
 Specifies a single-line item to be formatted for a routine's argument list.

related tags

- <ROUTINE_SECTION>
- <FARGS>
- <FFUNC>
- <FORMAT>
- <FRTN>

restrictions Valid only within the context of the <FORMAT> tag following an <FRTN> tag and an <FARGS> tag sequence.

required terminator *None.*

EXAMPLE The following example shows how to use the <FARG> tag to format a routine that allows a number of keyword arguments, each of which may specify several values.

```
<FORMAT>
<frtn>($FAB)<fargs>(ALQ = allocation-quantity,)
<farg>(BKS = bucket-size,)
<farg>(DEQ = extension-quantity,)
<farg>(DNA = default-filespec-address,)
<farg>(FNA = filespec-string-address,)
<farg>(ORG = <list>(stacked\braces)
           <le>REL
           <le>SEQ
           <le>IDX<endlist>,)
<farg>(RAT = <list>(stacked\braces)
           <le>CR
           <le><BLK FTN>
           <le>PRN<endlist>,)
<farg>(RTV = window-size,)
<farg>(SHR = <list>(stacked)
           <le><PUT GET DEL>
           <le>UPD NIL MSE UPI<endlist>,)
<farg>(XAB = xab-address)
<ENDFORMAT>
```

This example may produce the following output.

SOFTWARE Doctype Tag Reference

<FARG>

FORMAT
SAMPLE

\$FAB *ALQ = allocation-quantity,*
BKS = bucket-size,
DEQ = extension-quantity,
DNA = default-filespec-address,
FNA = filespec-string-address,
*ORG = { REL
 SEQ
 IDX } ,*
*RAT = { CR
 <BLK FTN>
 PRN } ,*
RTV = window-size,
*SHR = { <PUT GET DEL
 UPD NIL MSE UPI> } ,*
XAB = xab-address

<FARGS>

Provides the argument portion of a routine syntax statement within the context of the <FORMAT> tag.

FORMAT <FARGS> (*argument-list*[\ *STACK*])

ARGUMENTS *argument-list*

Lists the routine arguments. If there are no arguments, specify the argument list as null as follows:

<FRTN>(\$HIBER) <FARGS>()

Parameters specified in this argument differ on output from parameters specified using the second argument to the <FRTN> tag. For distinctions between these output differences, see the "Examples" section in the <FRTN> tag description.

STACK

Specifies that the routine's argument list should not be formatted on the same line as the routine name, but placed on the next line.

This keyword is required only for long routine names in certain doctypes. See if the output looks wrong before you use this argument.

related tags

- <FRTN>
- <FARG>
- <FFUNC>

restrictions

The <FARGS> tag should be used with the <FRTN> tag in the context of the <FORMAT> tag.

required terminator

None.

DESCRIPTION

The <FARGS> tag formats the argument portion of a routine in the context of the <FORMAT> tag.

When you use this tag to contain an argument list with multiple words or text strings, enter blank spaces between the words and text strings (including punctuation) according to the syntax rule of the routine you are describing. If the *parameter-list* argument text does not fit on one line, suitable line breaks are chosen based on the presence of spaces. Hyphenated text is not broken across lines.

You can select explicit line breaks in an argument list by using the <FARG> tag.

SOFTWARE Doctype Tag Reference

<FARGS>

EXAMPLES

The following examples show how you might use the <FARG> tag to code a format argument list within the context of a <FORMAT> tag in a routine section.

In the following example, the <FARGS> tag specifies a long argument list. By convention, if routine arguments are normally delimited by commas, you should place blank spaces in your SDML file preceding the commas. This allows suitable page breaks to be chosen during page composition.

1 <FRTN>(LIB\$CRF_OUTPUT) <FARGS>(ctl-tbl ,width ,pag1
,pag2 ,mode-ind ,del-sav-ind)

This example may produce the following output.

FORMAT	LIB\$CRF_OUTPUT	<i>ctl-tbl ,width ,pag1 ,pag2 ,mode-ind ,del-sav-ind</i>
---------------	------------------------	--

The following example shows how to position an argument list when a routine name is very long. You will want to use this form only if examination of your output indicates a formatting problem.

2 <FRTN>(SMG\$BEGIN_PASTEBOARD_UPDATE) <FARGS>(pasteboard-id\stack)

This example may produce the following output.

FORMAT	SMG\$BEGIN_PASTEBOARD_UPDATE	<i>pasteboard-id</i>
---------------	-------------------------------------	----------------------

SOFTWARE Doctype Tag Reference

<FFUNC>

EXAMPLES

The following example shows how to use the <FFUNC> tag within a format section in which a function has several forms.

1 <FORMAT>
<ffunc>(MTH\$SQRT\(*x*)>
<ffunc>(MTH\$DSQRT\(*x*)>
<ffunc>(MTH\$GSQRT\(*x*)>
<ffunc>(MTH\$HSQRT\(*x*)>
<ENDFORMAT>

This example may produce the following output.

FORMAT

MTH\$SQRT(*x*)
MTH\$DSQRT(*x*)
MTH\$GSQRT(*x*)
MTH\$HSQRT(*x*)

The following example shows how to use the three-argument form of the <FFUNC> tag.

2 <FORMAT>
<FFUNC>(status=AAA\$CODE\(*arg-one* , *arg-two* ,*arg-three* ,*arg-four* ,*arg-five*
,*arg-six* ,*arg-seven*))

This example may produce the following output.

FORMAT

status=AAA\$CODE(*arg-one* , *arg-two* ,*arg-three*
,*arg-four* ,*arg-five* ,*arg-six*
,*arg-seven*)

<FORMAT>

Begins a section that illustrates the syntax of a routine including keywords and arguments.

FORMAT **<FORMAT>** *[(alternate-heading)]*

ARGUMENTS *alternate-heading*

Specifies a heading to override the current default text heading for this use of the <FORMAT> tag. The default heading provided by VAX DOCUMENT is "Format." See the reference description of the <SET_TEMPLATE_HEADING> tag for information on how to modify the default headings for all <FORMAT> tags.

related tags

- <FRTN>
 - <FARGS>
 - <FARG>
 - <FFUNC>
-

required terminator

<ENDFORMAT>

DESCRIPTION

The <FORMAT> tag creates sections of text that describe the format of a command, routine, or tag. This tag is a global tag and is not restricted to routine sections. However in the context of the <ROUTINE_SECTION> and <TAG_SECTION> tags, the <FORMAT> tag enables certain additional tags for specialized format descriptions. These additional tags are not available outside of the context of routine or tag sections.

The global <FORMAT> tag enables the <FCMD>, <FPARMS>, and <FPARM>, which label specific portions of a format statement. In a routine section, the <FORMAT> tag also enables the tags <FRTN>, <FARGS>, <FARG>, and <FFUNC>.

You can use the <FORMAT> tag and the tags it enables in a variety of ways to show the syntax of a routine. The following list of code examples show some of the most regularly used format section tag combinations:

- <FRTN>(routine-keyword) <FARGS>(argument-list)

This is the standard form, in which the routine keyword and its argument list are separated by a blank space. If the argument list, on output, is more than a single line, additional lines will be aligned at the beginning of the argument list.

SOFTWARE Doctype Tag Reference

<FORMAT>

- <FRTN>(routine-keyword\argument-list)

This form should be used for routine functions, in which a routine and its arguments are not separated by blank spaces.

- <FRTN>(return-val\routine-keyword\argument-list)

This form provides three distinct elements for a routine description: the return value, the function name (shown here as the “routine-keyword”), and the argument list.

- <FRTN>(keyword-part) <FARGS>(argument-1) <FARG>(argument-2)

This form is useful for routines with long argument lists and in which the argument names themselves are either long or need additional information.

- <FFUNC>(routine-keyword\argument-list)

This form does not separate the routine name from its argument list and so is appropriate for function routines.

- <FFUNC>(return-val\routine-keyword\argument-list)

This form provides three distinct values (the return value, the function name, and the argument list).

EXAMPLES

The following example shows how to use the <FRTN> and <FARGS> tags in the context of the <FORMAT> tag to format a routine name and argument list. In this example, the argument list is null.

1 <FORMAT>
<FRTN>(SYS\$HIBER) <FARGS>()
<ENDFORMAT>

The following example shows how to use multiple <FFUNC> tags within a format section.

2 <FORMAT>
<ffunc>(MTH\$SQRT\ (x))
<ffunc>(MTH\$DSQRT\ (x))
<ffunc>(MTH\$GSQRT\ (x))
<ffunc>(MTH\$HSQRT\ (x))
<ENDFORMAT>

SOFTWARE Doctype Tag Reference

<FRTN>

EXAMPLES

The following example shows how to specify a single routine keyword. The <FARGS> tag is explicitly specified as null.

1 <FORMAT>
<FRTN>(\$HIBER) <FARGS>()
<ENDFORMAT>

This example may produce the following output.

FORMAT

\$HIBER

The following example specifies the routine keyword and its arguments using both the <FRTN> and <FARGS> tags. Note that blank spaces precede the commas that delimit the arguments. The spacing provides reasonable page-break points for use if the argument list does not fit in a single line of output.

2 <frtn>(\$ENQ) <fargs>([efn] ,lkmode ,lksb ,[flags]
,[resname] ,[parid] ,[astadr])

This example may produce the following output.

FORMAT

\$ENQ *[efn] ,lkmode ,lksb ,[flags] ,[resname] ,[parid]
,[astadr]*

The following example shows the output when two arguments are specified to the <FRTN> tag.

3 <FRTN>(NBR\$AAA\command.rt.dx)

This example may produce the following output.

FORMAT

NBR\$AAA*(command.rt.dx)*

The following example shows the three-argument form of the <FRTN> tag.

4 <FRTN>(status=AAA\$CODE\arg-one , arg-two ,arg-three ,arg-four ,arg-five
,arg-six ,arg-seven)

This example may produce the following output.

FORMAT

status=AAA\$CODE *arg-one , arg-two ,arg-three
,arg-four ,arg-five ,arg-six
,arg-seven*

<RETTEXT>

Provides general information about the attributes of the value returned by the routine.

FORMAT**<RETTEXT>****required
terminator**

<ENDRETTEXT>**restrictions**

This tag is valid only following the **<RETURNS>** tag.

DESCRIPTION

The **<RETTEXT>** tag lets you place one or more paragraphs of information after some usage of the **<RETURNS>** tag. This information begins immediately after the **<RETTEXT>** tag and continues until the **<ENDRETTEXT>** tag is encountered.

This tag is typically used when the arguments passed to the **<RETURNS>** tag are not sufficiently descriptive.

EXAMPLE

See the example in the discussion of the **<RETURNS>** tag.

SOFTWARE Doctype Tag Reference

<RETURNS>

<RETURNS>

Provides information about the value returned by a routine.

FORMAT **<RETURNS>** (*usage-information*
 \ *data-type* \ *access*
 \ *mechanism*[\ *optional-info*])

<RETURNS> (*HEADONLY*)

ARGUMENTS ***usage-information***
Specifies a keyword indicating the category of data to which the return value belongs. These keywords are system dependent and are specified by agreed-upon conventions.

data-type
Indicates the data type of the return value, for example, longword, byte, G_floating, and so on.

access
Indicates the access applied to the return value, for example, read-only, write-only, and so on.

mechanism
Specifies the mechanism by which the return value is passed; for example, by descriptor, by reference, or by value.

optional-info
Specifies additional information which may be appended to the *mechanism* argument output.

HEADONLY
Indicates that specific usage information is not relevant, and that text will be provided to describe the return attributes of the routine.

related tags • <RETTEXT>

required terminator *None.*

DESCRIPTION The returns section of the Routine reference template and its specialized argument list are used only by convention for those documenting callable routines.

EXAMPLES

The following example shows how to specify multiple arguments to the <RETURNS> tag.

1 <RETURNS>(floating_point\
F_Floating, D_Floating, or G_Floating\write only\by value in R0)

This example may produce the following output.

RETURNS

VMS Usage: **floating_point**
type: **F_Floating, D_Floating, or G_Floating**
access: **write only**
mechanism: **by value in R0**

The following example shows a use of the HEADONLY argument. The return value from a routine or set of related routines may be too complex to express using the conventional arguments.

2 <RETURNS>(headonly)
<RETTEXT>The square roots of F_Floating, D_Floating, and G_Floating input parameters are returned by immediate value in R0 and R1. The square root of an H_Floating parameter is returned by reference in the output parameter <VARIABLE>(sqrt).
<ENDRETTEXT>

This example may produce the following output.

RETURNS

The square roots of F_Floating, D_Floating, and G_Floating input parameters are returned by immediate value in R0 and R1. The square root of an H_Floating parameter is returned by reference in the output parameter *sqrt*.

SOFTWARE Doctype Tag Reference

<ROUTINE>

<ROUTINE>

Begins a new routine description.

FORMAT <ROUTINE> (*routine-name*[\ *info1*[\ *info2*]])

ARGUMENTS *routine-name*
Specifies the name of the routine to be described.

info1

info2

Specifies an optional text description of the routine's function.

related tags

- <ROUTINE_SECTION>
- <SET_TEMPLATE_ROUTINE>

required terminator

None.

DESCRIPTION Use the <ROUTINE> tag to begin a description section for a single routine in the context of the <ROUTINE_SECTION> tag. This tag has the following default format:

- Each <ROUTINE> tag begins a new page of output.
- Each output page carries a single running title, which is the current routine name.
- If the optional *info* arguments are used with the <ROUTINE> tag, these arguments are output after the *routine-name* argument and are separated from the *routine-name* argument by an em dash (—).

If you wish to replace the <ROUTINE> tag with a tag whose name is more meaningful to you (for example, <DCL_ROUTINE>), or if you wish to change the default attributes of the <ROUTINE> tag, you should use the <SET_TEMPLATE_ROUTINE> tag. See the description of the <SET_TEMPLATE_ROUTINE> tag in this chapter for more information.

EXAMPLES

In the following example, the <ROUTINE_SECTION> tag enables the tags for a routine description.

1 <ROUTINE_SECTION>
<ROUTINE>(\$OPEN)
<OVERVIEW>

The description of the routine \$OPEN will have the following default attributes:

- The routine description will start a new page.
- If the routine carries for more than a page, the name "\$OPEN" will be carried as a running top title on each page.

2 <ROUTINE>(\$CLOSE\Close a File)

When two arguments are specified to the <ROUTINE> tag, the routine name, \$CLOSE, appears at the beginning of the routine description. The text specified in the second argument, "Close a File," follows the routine name at the top of the page, separated from the routine name by an em dash (—).

3 <ROUTINE>(SMG\$BEGIN_PASTEBOARD_UPDATE\\Begin Batching of Pasteboard Updates)

This example illustrates special coding required when a routine's name is extremely long and may require special formatting. If the second argument is null, the third argument is stacked under the first argument. No em dash is output:

```
SMG$BEGIN_PASTEBOARD_UPDATE
Begin Batching of Pasteboard Updates
```

Use this form only if the output from the other formats appears wrong.

4 <ROUTINE>(OTS\$SCOPY_R_DX\Copy a Source String \Passed by Reference to a Destination String)

This example illustrates special coding required when a routine's name and descriptive name creates unreasonable output using the default formatting attributes. If you specify three arguments, the first and second arguments are output on the first line, separated by an em dash. The third argument is stacked under the first argument (instead of wrapping) following the em dash:

```
OTS$SCOPY_R_DX---Copy a Source String
Passed by Reference to a Destination String
```

You will want to use this form only if examination of your output indicates a formatting problem.

SOFTWARE Doctype Tag Reference

<ROUTINE_SECTION>

- <RETURNS>
- <RETTEXT>
- <ARGDEFLIST>
- <ARGITEM>
- <ARGDEF>
- <ARGTEXT>
- <DESCRIPTION>
- <RSDEFLIST>
- <RSITEM>
- <EXAMPLE_SEQUENCE>

restrictions

None.

required terminator

<ENDROUTINE_SECTION>

DESCRIPTION

The <ROUTINE_SECTION> tag begins a routine reference section. You can place more than one routine section within a single document, however, each previous routine section must be ended before the next one is begun.

You can tailor the default format of the routine reference template to meet your own documentation requirements. You can either alter the default attributes of the <ROUTINE_SECTION> tag by specifying one of the arguments to that tag, or you can use the <SET_TEMPLATE_ROUTINE> tag to alter the default attributes for the <ROUTINE> tag that begins each new routine description.

You can place a routine section within a chapter or an appendix, or following a part page (that is, within a document section begun with the <PART_PAGE> tag). You code a routine section in a chapter or an appendix in the same manner; command sections in parts are handled differently.

If your routine section follows a part page, and you include text between the part page and the routine section, specify the NEWPAGE keyword as the third argument to the <ROUTINE_SECTION> tag. This causes the routine section to begin on a new page. The following code fragment shows a routine section that begins on a new page:

```
<ROUTINE_SECTION>(\AB\NEWPAGE)
<HEAD1>(Routine Format)
```

When you use the <ROUTINE_SECTION> tag within a chapter or an appendix, and you want to place text after the routine section in that chapter or appendix, you must end the routine section with the <ENDROUTINE_SECTION> tag and place the text after that tag. By default, this text begins on a new page of output.

SOFTWARE Doctype Tag Reference

<ROUTINE_SECTION>

Specify the `NONEWPAGE` argument to the `<ENDROUTINE_SECTION>` tag if you do not want the text to begin on a new page of output. The following code fragment shows the end of a routine section that specifies that the subsequent text not placed on a new page:

```
<ENDROUTINE_SECTION>(NONEWPAGE)
```

When the `<ENDROUTINE_SECTION>` tag is specified within the context of a chapter or appendix, it resets the default running titles to those in effect for the chapter or appendix, so the last page of the last routine description in the routine section may not carry the last routine's name as the running heading. Instead it may carry the running title used by the chapter or appendix.

EXAMPLES

The following example shows how to begin a routine section within a document part.

```
1 <PART>
  <PART_PAGE>
  <TITLE>(PART II\AAA Routine Descriptions)
  <ABSTRACT>This Part contains complete reference descriptions of
  the AAA routines.
  <ENDABSTRACT>
  <ENDPART_PAGE>(RENUMBER)
  <ROUTINE_SECTION>(AAA Routines\AAA)
  <SET_TEMPLATE_ROUTINE>(AAA_ROUTINE\DOUBLERUNNINGHEADS)

  <AAA_ROUTINE>(AAA$CLOSE)

  <OVERVIEW>
  Closes the specified file.
  <ENDOVERVIEW>
  .
  .
  <ENDROUTINE_SECTION>
```

The tags in the previous example perform the following functions:

- The global `<PART>` tag begins the part.
- The global `<PART_PAGE>` tag creates a part page.
- The global `<TITLE>` tag is used in the context of the `<PART_PAGE>` tag to create a title on the part page.
- The `RENUMBER` argument to the global `<ENDPART_PAGE>` tag specifies that the pages should be renumbered beginning with the part page. This causes the first page of text following the part page to be numbered page 3 (Page 1 is the unnumbered page the part page title is placed on, page 2 is the back of page 1, and page 3 is the first numbered page after the part page).
- The `<ROUTINE_SECTION>` tag begins the routine section and specifies the running title "AAA Routines" as the running title for the routine section.

The `<ROUTINE_SECTION>` tag also specifies that the prefix "AAA" should be used to construct numbers for pages and for formal figures, tables, and examples within the routine section (for example, AAA-11, AAA-32, Table AAA-1, Example AAA-2, and so on).

SOFTWARE Doctype Tag Reference

<ROUTINE_SECTION>

- The <SET_TEMPLATE_ROUTINE> tag specifies that all routine descriptions in this routine section will be identified using the tag <AAA_ROUTINE> rather than the default <ROUTINE> tag. The <AAA_ROUTINE> tag will have the default attributes of the <ROUTINE> tag.

The DOUBLERUNNINGHEADS argument to the <SET_TEMPLATE_ROUTINE> tag specifies that the routine section should have a double running heading at the top of the page. The top heading is the *running-title* specified as an argument to the <ROUTINE_SECTION> tag, and the lower heading is the name of the current routine.

The following example shows how you can create a routine section in which each routine description (begun with a <ROUTINE> tag) is in a separate SDML file, and all these descriptions are included into a primary routine description file. For example, the file MYROUTINES.SDML contains the following SDML tags:

```
<INCLUDE>(AAA$CLOSE.SDML)
<INCLUDE>(AAA$OPEN.SDML)
<INCLUDE>(AAA$READ.SDML)
<INCLUDE>(AAA$WRITE.SDML)
```

Each of the included files contains one routine reference description begun with a <ROUTINE> tag. For these files to process correctly, they must be preceded with the <ROUTINE_SECTION> tag that enables the <ROUTINE> tag. These files can have the necessary tags processed before them by specifying the /INCLUDE qualifier on the command line to include a startup definition file. This startup file might include the following tags.

```
2 <ROUTINE_SECTION>(AAA Routines\AAA)
  <SET_TEMPLATE_ROUTINE>(ROUTINE\DOUBLERUNNINGHEADS)
```

If this startup file were named AAAROUTINE_STARTUP.SDML, it could be included using the DOCUMENT /INCLUDE qualifier as in the following example:

```
$ DOCUMENT myroutines SOFT.REF LN03 /INCLUDE=AAAROUTINE_STARTUP.SDML
```

When each individual file in MYROUTINES.SDML is processed, the correct sequence of tags will be read in to begin the routine section.

You can process multiple files together by using the <INCLUDE> tag to include them into a single master file (such as MYROUTINES.SDML), or you can include them into a book build profile.

You use the <ELEMENT> tags to include multiple files into a profile. For example, the book build profile file AAAPRO.SDML could contain the following tags:

```
<PROFILE>
<ELEMENT>(AAA$CLOSE.SDML)
<ELEMENT>(AAA$OPEN.SDML)
<ELEMENT>(AAA$READ.SDML)
<ELEMENT>(AAA$WRITE.SDML) <COMMENT>(contains <ENDROUTINE_SECTION> tag)
<ENDPROFILE>
```

Note that the profile file should include the <ENDROUTINE_SECTION> tag in the appropriate file, so that the template will be terminated and the book build will process correctly.

SOFTWARE Doctype Tag Reference

<RSDEFLIST>

<RSDEFLIST>

Begins a return status definition list in the routine template.

FORMAT

<RSDEFLIST> [({ *NONE*
alternate-heading[\ *NONE*] })]
[*alternate-heading*]\ *TEXT*]]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading for this use of the <RSDEFLIST> tag. The default heading provided by VAX DOCUMENT is "Return Values." See the reference description of the <SET_TEMPLATE_HEADING> tag for information on how to modify the default headings for all <RSDEFLIST> tags.

TEXT

Specifies that a block of text should appear in place of a list of return status values. If you use this keyword, it must be positioned as the second argument to the <RSDEFLIST> tag.

To use the default heading and to indicate that text follows, specify the following:

<RSDEFLIST>(\TEXT)

NONE

Indicates that the routine does not return any values.

related tags

- <RSDEFLIST>
- <RSITEM>

restrictions

If *NONE* is specified as an argument to the <RSDEFLIST> tag, the <ENDRSDEFLIST> tag should not be used.

required terminator

<ENDRSDEFLIST>

DESCRIPTION

The <RSDEFLIST> tag begins a two-column table listing the return status of the routines listed. VAX DOCUMENT formats this table as a multipage table, which means that the <TABLE_ROW_BREAK> (FIRST) and <TABLE_ROW_BREAK> (LAST) tags can be used to control page breaks within the table if such control is needed.

EXAMPLES

The following example illustrates a return status definition list for a routine with two possible return values.

1 <RSDEFLIST>
<RSITEM>(SS\$_NORMAL\Service successfully completed.)
<RSITEM>(SS\$_ACCVIO\Access violation.)
<ENDRSDEFLIST>

This example may produce the following output.

RETURN VALUES

SS\$_NORMAL	Service successfully completed.
SS\$_ACCVIO	Access violation.

The following example uses the NONE keyword argument to indicate that a routine does not return any status values. Note that <ENDRSDEFLIST> must not be specified.

2 <RSDEFLIST>(NONE)

This example may produce the following output.

RETURN VALUES

None.

The following example illustrates a single line of descriptive text in a Return Status section.

3 <RSDEFLIST>(\TEXT)
Any condition values returned by the Record Management Service (RMS),
Parse.
<ENDRSDEFLIST>

This example may produce the following output.

RETURN VALUES

Any condition values returned by the Record Management Service (RMS),
Parse.

SOFTWARE Doctype Tag Reference

<RSITEM>

<RSITEM>

Specifies the return status value of a routine and lists its meaning.

FORMAT <RSITEM> (*code* \ *code-description*)

ARGUMENTS

code

Specifies the system keyword assigned to the status value.

code-description

Specifies the descriptive text explaining the meaning of the return status value.

related tags

- <RSDEFLIST>

restrictions

Valid only within the context of the <RSDEFLIST> tag.

required terminator

None.

EXAMPLES

These examples show only code fragments and no actual output. See the <RSDEFLIST> tag description for sample output.

1 <RSDEFLIST>
<RSITEM>(SS\$_NORMAL\Normal, successful completion.)
<RSITEM>(SS\$_ACCVIO\Access violation.)
<ENDRSDEFLIST>

This example illustrates a Return Status section for a routine that has two possible return values.

2 <RSDEFLIST>
<RSITEM>(SS\$_NORMAL\Normal successful completion.)
<RSITEM>(SMG\$_WRONNUMARG\Wrong number of arguments.)
<RSITEM>((2\LEFT)Any condition values returned by
LIB\$SCOPY_DXDX, SMG\$ADD_KEY_DEF, or by CLI\$ routines.)
<ENDRSDEFLIST>

This example illustrates the Return Status definition list for a routine that has two specific return values and uses the tag to place additional explanatory text within the table.

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_ROUTINE>

required terminator

None.

DESCRIPTION

The <SET_TEMPLATE_ROUTINE> tag lets you change the name of the <ROUTINE> tag, making the coding of your source files more meaningful.

This tag also lets you change the default attributes associated with the <ROUTINE> tag or your new tag.

EXAMPLES

1

```
<ROUTINE_SECTION>(FDL Routines\\NEWPAGE)
<SET_TEMPLATE_ROUTINE>(FDL_ROUTINE\NONEWPAGE\STACK\DOUBLERUNNINGHEADS)
<FDL_ROUTINE>($STARTUP_FILE\$STF)
```

This tag sequence begins a routine section and defines the tag <FDL_ROUTINE> as the routine descriptor tag. The attributes set are as follows:

- Routines do not start on new pages. However, because NEWPAGE is specified to <ROUTINE_SECTION>, the first reference description will begin on a new page.
- When two arguments are specified to the tag <FDL_ROUTINE>, the arguments are stacked; that is, the second argument is placed under the first on output.
- Each page carries a two-line running title. The top line is "FDL Routines" and the second line is the name of the routine description that is current at the top of the page.

Note that this routine sequence is most appropriate for a series of routine descriptions that occur within a chapter.

2

```
<PART>
<PART_PAGE>
<TITLE>(PART II\XYZ Routines)
<ENDPART_PAGE>(RENUMBER)
<ROUTINE_SECTION>(XYZ Routines\XYZ)
<SET_TEMPLATE_ROUTINE>(XYZ_ROUTINE)
```

This tag sequence begins a document part in which only routine descriptions occur. The RENUMBER argument on the <ENDPART_PAGE> tag specifies that the first page of the new Part (the part page itself) should be numbered page 1. The <ROUTINE_SECTION> tag sets the folio prefix to XYZ; that is, pages will be numbered XYZ-3, XYZ-4, and so on.

No attributes are specified for the routine descriptions produced by <XYZ_ROUTINE>. These routines will have default attributes. The first routine will start on a new page. Because RENUMBER was specified on <ENDPART_PAGE>, the first routine will be on the page numbered XYZ-3.

14.5 The Statement Reference Template Tags

This section alphabetically describes the tags available in the Statement template. Use these tags to create a statement reference section. In most manuals, a statement section is an encyclopedic reference section that describes each statement the software system offers, the statement's format, its arguments, and examples of its use.

SOFTWARE Doctype Tag Reference

<CONSTRUCT>

<CONSTRUCT>

Specifies a variable construct and gives its expansion.

FORMAT <CONSTRUCT> [(*construct-name*)]

ARGUMENTS *construct-name*

Specifies the name of a construct. If this argument is omitted, text following the <CONSTRUCT> tag is formatted as if text had been specified; that is, the text is set at the same margin as additional construct items.

related tags

- <CONSTRUCT_LIST>
- <STATEMENT_LINE>

restrictions

Valid only within the context of the <STATEMENT_FORMAT> tag.

required terminator

None.

EXAMPLE

The following example shows statement format for a statement with a single construct list. Note the use of <LIST> tags to specify the output and the <KEYWORD> tag to control the representation of programming keywords within variable name text.

```
<statement_format>
<FCMD>(REMAP) <FPARMS>((map-nam) remap-item,<hellipsis>)
<construct_list>(remap-item:)
<construct>(remap-item:)<list>(stacked\braces)
  <le>num-vbl-nam
  <LE>str-array-nam ( [ int-exp,<hellipsis>] ) [ = int-exp ]
  <LE>[ data-type ] <keyword>(FILL) [ ( int-exp ) ] [ = int-exp ]
  <LE><keyword>(FILL%) [ ( int-exp ) ]
  <LE><keyword>(FILL$) [ ( int-exp ) ] [ = int-exp ]
  <endlist>
<endconstruct_list>
<endstatement_format>
```

This example may produce the following output.

Format

REMAP (*map-nam*) *remap-item*, . . .

SOFTWARE Doctype Tag Reference
<CONSTRUCT>

remap-item: { *num-vbl-nam*
str-array-nam ([*int-exp*, . . .]) [= *int-exp*]
[*data-type*] **FILL** [(*int-exp*)] [= *int-exp*]
FILL% [(*int-exp*)]
FILL\$ [(*int-exp*)] [= *int-exp*] }

SOFTWARE Doctype Tag Reference

<CONSTRUCT_LIST>

<CONSTRUCT_LIST>

Begins a list of construct items and definitions that expand on variables specified in the context of the <STATEMENT_FORMAT> tag.

FORMAT <CONSTRUCT_LIST> (*construct-name*)

ARGUMENTS *construct-name*

Specifies the text of the longest name that will be referenced in the construct list; that is, a name specified as an argument to <CONSTRUCT> .

related tags

- <CONSTRUCT>
- <STATEMENT_LINE>

restrictions

Valid only within the context of a <STATEMENT_FORMAT> tag section.

required terminator

<ENDCONSTRUCT_LIST>

DESCRIPTION A *construct list* is a set of semantic rules for a programming language. Each item in a construct list is a semantic entity that may expand to one or more sets of additional constructs or entities. Using the global <LIST> tag in a construct list within a statement section, you can present the semantic rules for language statements in a highly-structured way.

EXAMPLES The following example shows a construct list with only one construct.

```
1 <statement_format>
  <fcmd>(<list>(stacked\braces)
  <le>COM
  <le>COMMON<endlist> <fparms>([ ( com-nam ) ] {[data-type ] com-item },<hellipsis>)
  <construct_list>(com-item:)
  <construct>(com-item:)
  <list>(stacked\braces)
  <le><VARIABLE>(num-unsubs-vbl-nam)
  <le><VARIABLE>(num-array-nam ( int-const,<hellipsis>))
  <le><VARIABLE>(str-unsubs-vbl-nam = int-const)
  <le><VARIABLE>(str-array-nam ( int-const,<hellipsis> ) [ = int-const ])
  <le><VARIABLE>(<keyword>(FILL) [ ( int-const ) ][ = int-const ])
  <le><VARIABLE>(<keyword>(FILL%) [ ( int-const ) ])
  <le><VARIABLE>(<keyword>(FILL$) [ ( int-const ) ][ = int-const ])
  <endlist>
  <endconstruct_list>
  <endstatement_format>
```

SOFTWARE Doctype Tag Reference

<CONSTRUCT_LIST>

This example may produce the following output.

Format

**{ COM
COMMON }** [(*com-nam*)] { [*data-type*] *com-item* }, ...

com-item: { *num-unsubs-vbl-nam*
num-array-nam (*int-const*, ...)
str-unsubs-vbl-nam = *int-const*
str-array-nam (*int-const*, ...) [= *int-const*]
FILL [(*int-const*)][= *int-const*]
FILL% [(*int-const*)]
FILL\$ [(*int-const*)][= *int-const*] }

The following example illustrates a construct list with more than one construct. The argument to <CONSTRUCT_LIST> is used to set the margins for the individual items identified by <CONSTRUCT> tags. Note that the text of the longest item, *formal-param*: is specified.

```

2 <FCMD>( <list>(stacked\braces)
    <le>END SUB
    <le>SUBEND<endlist>) <FPARMS>()
<construct_list>(formal-param:)
<construct>(pass-mech:) <list>(stacked\braces)
    <le><keyword>(BY DESC)
    <le><keyword>(BY REF)<endlist>
<construct>(formal-param:)
[ data-type ] <list>(stacked\braces)
    <le>unsubs-vbl-nam
    <le>array-nam ( <list>(stacked\brackets)
        <le>int-const
        <le><keyword>(,)<endlist>
        <list>(stacked)
        <le><keyword>(,<hellipsis>)
        <le><keyword>( <hellipsis>)<endlist><endlist>
    <endconstruct_list>
<endstatement_format>

```

This example may produce the following output.

SOFTWARE Doctype Tag Reference

<CONSTRUCT_LIST>

Format

{ **END SUB**
 SUBEND }

pass-mech: { **BY DESC**
 BY REF }

formal-param: [*data-type*]

{ *unsubs-vbl-nam*
 array-nam ([*int-const*] [*, ...*]) }

<FCMD>

Specifies a statement or function keyword and an optional parameter list within the context of the <STATEMENT_FORMAT> tag.

FORMAT <FCMD> (*statement-keyword*[\ *parameter-list*])

ARGUMENTS *statement-keyword*

Specifies the name of the first part of the format statement; typically this is the principal statement name or the function name. This text is output in the left portion of the format description.

parameter-list

Specifies one or more parameters of the statement or function keyword. This text is output to the right of the statement or function in the format description with no space between the *statement-keyword* and the *parameter-list* text.

Parameters you specify using the *parameter-list* argument are output differently than parameters specified using the <FPARMS> tag. See the examples in this tag description for illustrations of these differences.

related tags

- <FFUNC>
- <FPARM>
- <FPARMS>
- <STATEMENT_FORMAT>
- <STATEMENT_SECTION>

restrictions

- Valid only within the context of the <FORMAT> or <STATEMENT_FORMAT> tags.
- If you do not provide a second argument to the <FCMD> tag, you should explicitly declare the absence of parameters by using the <FPARMS> tag as follows:

<FCMD>(STATEMENT-KEYWORD) <FPARMS>()

If you do not specify a second argument to the <FCMD> tag and the <FPARMS> tag is not specified, VAX DOCUMENT issues a warning message.

required terminator

None.

SOFTWARE Doctype Tag Reference

<FCMD>

DESCRIPTION

Use the <FCMD> tag to label a statement or function keyword within the context of the <STATEMENT_FORMAT> tag. Use the *parameter-list* argument to this tag to create a list of one or more parameters that are formatted with no space between the statement or function keyword and the parameter list.

Use the <FPARMS> tag in conjunction with the <FCMD> tag to create a statement format with the statement or function keyword and the one or more parameters separated by a space.

If the text of the *parameter-list* argument does not fit on a single line in the statement format section, the text formatter selects suitable line breaks based on the presence of spaces in the *parameter-list* text. Hyphenated text is not broken across lines.

EXAMPLES

The following are examples of various uses of the <FCMD> tag in the context of the <STATEMENT_FORMAT> tag.

The following example specifies a statement keyword with no parameters. Use the <FPARMS> after the <FCMD> to explicitly specify the absence of any parameters.

1 <STATEMENT_FORMAT>
<FCMD>(EXIT) <FPARMS>()
<ENDSTATEMENT_FORMAT>

This example may produce the following output.

Format

EXIT

The following example also specifies a statement keyword with no parameters. This coding however specifies a null second argument to the <FCMD> tag rather than using the <FPARMS> tag to explicitly specify the absence of any parameters.

2 <STATEMENT_FORMAT>
<FCMD>(EXIT\)
<ENDSTATEMENT_FORMAT>

This example may produce the following output.

Format

EXIT

The following example specifies both the *statement-keyword* and the *parameter-list* arguments to the <FCMD> tag. Note in the output sample how these two arguments are formatted together with no intervening spaces.

```

3 <STATEMENT_FORMAT>
  <FCMD>(RETURN\ident_field, numeric_field)
  <ENDSTATEMENT_FORMAT>

```

This example may produce the following output.

Format

RETURN*ident_field, numeric_field*

The following example specifies a statement keyword with a single parameter using the <FCMD> and <FPARMS> tags. Note that coding the parameters using the <FPARMS> tag results in a space being output between the command keyword and the parameter.

```

4 <STATEMENT_FORMAT>
  <FCMD>(RECORD) <FPARMS>(rec-nam)
  <ENDSTATEMENT_FORMAT>

```

This example may produce the following output.

Format

RECORD *rec-nam*

The following example specifies the <FCMD> tag using the global <KEYWORD> tag as part of the *parameter-list* argument text. Note how the global <KEYWORD> tag alters the output. The output of this example is shown in Output Sample 5.

```

5 <STATEMENT_FORMAT>
  <FCMD>(RETURN\ident_field, numeric_field,<keyword>([/LENGTH]))
  <ENDSTATEMENT_FORMAT>

```

This example may produce the following output.

Format

RETURN*ident_field, numeric_field,[/LENGTH]*

EXAMPLES

The following example shows a statement format in which a function has several forms.

```
1 <STATEMENT_FORMAT>
  <ffunc>(real-vbl\=ABS<VARIABLE>((real-exp)))
<ENDSTATEMENT_FORMAT>
```

This example may produce the following output.

Format

real-vbl=ABS(real-exp)

The following example shows the three-argument form of the <FFUNC> tag.

```
2 <STATEMENT_FORMAT>
  <FFUNC>(status\=AAA$CODE\ (arg-one , arg-two ,arg-three ,arg-four ,arg-five
  ,arg-six ,arg-seven))
<ENDSTATEMENT_FORMAT>
```

This example may produce the following output.

Format

*status=AAA\$CODE(arg-one , arg-two ,arg-three ,arg-four ,arg-five
,arg-six ,arg-seven)*

SOFTWARE Doctype Tag Reference

<FORMAT_SUBHEAD>

<FORMAT_SUBHEAD>

Introduces one of a set of multiple formats in a statement format section.

FORMAT <FORMAT_SUBHEAD> (*heading-text*)

ARGUMENTS *heading-text*

Specifies the text of the heading to be used before a specific format presentation.

related tags

- <FCMD>
- <FPARMS>
- <STATEMENT_FORMAT>

restrictions

Valid only within the context of the <STATEMENT_FORMAT> tag if the MULTIPLE keyword was specified as the second argument to the <STATEMENT_FORMAT> tag.

required terminator

None.

EXAMPLE

The following example shows how to use the <FORMAT_SUBHEAD> tag in a series of formats for a single statement.

```
<statement_format>(\multiple)
<FORMAT_SUBHEAD>(String Variable To Array)
<FCMD>(CHANGE) <FPARMS>(str-exp <KEYWORD>(TO) num-array)
<FORMAT_SUBHEAD>(Array to String Variable)
<FCMD>(CHANGE) <FPARMS>(num-array <KEYWORD>(TO) str-vbl)
<endstatement_format>
```

This example may produce the following output.

Format

String Variable To Array

CHANGE *str-exp* **TO** *num-array*

Array to String Variable

CHANGE *num-array* **TO** *str-vbl*

SOFTWARE Doctype Tag Reference

<FPARM>

<FPARM>

Adds a single parameter line to a list of parameters in a statement syntax format section.

FORMAT <FPARM> (*parameter-name*)

ARGUMENTS *parameter-name*
Specifies a single-line item to be formatted in a format parameter list.

related tags

- <STATEMENT_SECTION>
- <STATEMENT_FORMAT>
- <FCMD>
- <FPARMS>

restrictions Can only be specified following an <FCMD> tag and <FPARMS> tag sequence in a format section.

required terminator *None.*

EXAMPLE The following example shows a sample use of the <FPARM> tag.

```
1 <FORMAT>
  <fcmd>(SET TERMINAL) <fparms>([[device-name[:]])
  <fparm>( /DEVICE_TYPE=<list>(stacked\braces)
    <le><keyword>(UNKNOWN)
    <le><keyword>(LA34)
    <le><keyword>(VT100)
    <le><keyword>(PRO_SERIES)<endlist>)
  <fparm>( <list>(stacked\brackets)
    <le>LINE_EDITING
    <le>NOLINE_EDITING<endlist>)
  <fparm>( <list>(stacked\brackets)
    <le>PASSALL
    <le>NOPASSALL<endlist>)
  <ENDFORMAT>
```

This example may produce the following output

SOFTWARE Doctype Tag Reference
<FPARM>

FORMAT	SET TERMINAL	<i>[device-name[:]]</i> <i>/DEVICE_TYPE=</i> <i>{</i> <i> UNKNOWN</i> <i> LA34</i> <i> VT100</i> <i> PRO_SERIES</i> <i>}</i> <i>[/LINE_EDITING</i> <i>/NOLINE_EDITING]</i> <i>[/PASSALL</i> <i>/NOPASSALL]</i>
---------------	---------------------	---

SOFTWARE Doctype Tag Reference

<FPARMS>

<FPARMS>

Specifies the parameters to a statement or function within the context of the <STATEMENT_FORMAT> tag.

FORMAT <FPARMS> (*parameter-list*)

ARGUMENTS *parameter-list*

Specifies one or more parameters to a statement or function.

You can also use this argument to explicitly declare that a statement or function marked with the <FCMD> tag has no parameters, as shown in the following example.

```
<FCMD>(EXIT) <FPARMS>()
```

Parameters you specify using the <FPARMS> tag are printed differently than parameters specified in the *parameter-list* argument to the <FCMD> tag. See the examples in this tag description for illustrations of the different output styles.

related tags

- <FCMD>
- <FFUNC>
- <STATEMENT_FORMAT>
- <STATEMENT_SECTION>

restrictions

The <FPARMS> tag should be used with the <FCMD> tag in the context of the <STATEMENT_FORMAT> tag.

required terminator

None.

DESCRIPTION The <FPARMS> tag formats the parameter portion of a statement or function in the context of the <STATEMENT_FORMAT> tag.

When you use this tag to define a parameter list that contains multiple words or text strings, include blank spaces between the words and text strings (including punctuation) according to the syntax rules of the function or statement you are describing.

If the *parameter-list* argument text will not fit on one line, suitable line breaks are chosen based on the presence of spaces. Hyphenated text is not broken across lines.

Use the <FPARM> tag to select explicit line breaks in a parameter list.

EXAMPLES

The following example uses the <FPARMS> tag to list an additional statement keyword within the parameters portion of a statement format.

1 <FCMD>(END RECORD) <FPARMS>([rec-nam])

This example may produce the following output.

Format

END RECORD [*rec-nam*]

The following example shows how to represent variable choices within the <FPARMS> parameter-list.

2 <statement_format>
<FCMD>(MAT READ) <FPARMS>(<list>(stacked\braces)
 <le> array <list>(stacked\brackets)<le> (int-exp1
 <list>(stacked\brackets)<le> , int-exp2 <endlist>) <endlist>
 <endlist>,<hellipsis>)
<endstatement_format>

This example may produce the following output.

Format

MAT READ { array [(*int-exp1* [, *int-exp2*])] } , . . .

SOFTWARE Doctype Tag Reference

<FUNCTION>

<FUNCTION>

Begins a new function description.

FORMAT <FUNCTION> (*function-name*)

ARGUMENTS *function-name*
Specifies the name of the function to be described.

related tags

- <STATEMENT>
- <SET_TEMPLATE_STATEMENT>

required terminator *None.*

DESCRIPTION Use the <FUNCTION> tag to begin a description section for a single function in the context of the <STATEMENT_SECTION> tag. This tag has the following default format:

- Each <FUNCTION> tag begins a new page of output.
- Each output page carries a single running title, which is the current function name.

Use the <SET_TEMPLATE_STATEMENT> tag to replace the <FUNCTION> tag with a tag whose name is more meaningful to you (for example, <ABC_FUNCTION>), or to change the default attributes of the <FUNCTION> tag. See the description of the <SET_TEMPLATE_STATEMENT> tag in this chapter for more information.

Note that the <FUNCTION> and the <STATEMENT> tags work in exactly the same manner. The two tag names are provided so that you may encode your source file more generically.

EXAMPLE In the following example, the <STATEMENT_SECTION> tag enables the tags for a function description. The description of the function OPEN will, by default, have the following attributes:

- The function description begins on a new page.
- If the function carries for more than a page, the name "OPEN" is carried as a running top title on each page.

<STATEMENT_SECTION>
<FUNCTION>(OPEN)
<OVERVIEW>

SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_STATEMENT>

**required
terminator**

None.

DESCRIPTION The <SET_TEMPLATE_STATEMENT> tag lets you change the name of the <STATEMENT> tag to a name that is more meaningful in your source files. This tag also lets you change the default attributes associated with the <STATEMENT> tag or your new tag.

EXAMPLES

1 <STATEMENT_SECTION>(BASIC Statements\\NEWPAGE)
<SET_TEMPLATE_STATEMENT>(BASIC_STATEMENT\\NONEWPAGE\\DOUBLERUNNINGHEADS)
<BASIC_STATEMENT>(STARTUP_FILE\\STF)

This tag sequence begins a statement section and defines the tag <BASIC_STATEMENT> as the statement descriptor tag. The attributes set are as follows:

- Statements do not start on new pages.
- Each page carries a two-line running title. The top line is "BASIC Statements" and the second line is the name of the statement description that is current at the top of the page.

Note that this statement sequence is most appropriate for a series of statement descriptions that occur within a chapter.

2 <PART>
<PART_PAGE>
<TITLE>(PART II\\XYZ Statements)
<ENDPART_PAGE>(RENUMBER)
<STATEMENT_SECTION>(XYZ Statements\\XYZ)
<SET_TEMPLATE_STATEMENT>(XYZ_STATEMENT)

This tag sequence begins a document part in which only statement descriptions occur. The RENUMBER argument on the <ENDPART_PAGE> tag specifies that the first page of the new part (the part page itself) should be numbered page 1. The <STATEMENT_SECTION> tag sets the folio prefix to XYZ; that is, pages will be numbered XYZ-3, XYZ-4, and so on.

No attributes are specified for the statement descriptions produced by <XYZ_STATEMENT>; these statements have default attributes. The first statement will start on a new page. Because RENUMBER was specified on <ENDPART_PAGE>, the first statement will be on the page numbered XYZ-3.

<STATEMENT>

Begins a new statement description.

FORMAT <STATEMENT> (*statement-name*)

ARGUMENTS *statement-name*
Specifies the name of the statement to be described.

related tags

- <FUNCTION>
- <SET_TEMPLATE_STATEMENT>
- <STATEMENT_SECTION>

required terminator *None.*

DESCRIPTION Use the <STATEMENT> tag to begin a description section for a single statement in the context of the <STATEMENT_SECTION> tag. This tag has the following default format:

- Each <STATEMENT> tag begins a new page of output.
- Each output page carries a single running title, which is the current statement name.

Use the <SET_TEMPLATE_STATEMENT> tag to replace the <STATEMENT> tag with a tag whose name is more meaningful to you (for example, <ABC_STATEMENT>), or if you wish to change the default attributes of the <STATEMENT> tag. See the description of the <SET_TEMPLATE_STATEMENT> tag in this chapter for more information.

EXAMPLES

In the following example, the <STATEMENT_SECTION> tag enables the tags for a statement description. By default, the description of the statement OPEN has the following attributes:

- The statement description begins on a new page.
- If the statement carries for more than a page, the name "OPEN" is carried as a running top title on each page.

1 <STATEMENT_SECTION>
 <STATEMENT>(OPEN)
 <OVERVIEW>

SOFTWARE Doctype Tag Reference

<STATEMENT_FORMAT>

<STATEMENT_FORMAT>

Begins a section that illustrates the syntax of a statement or function, including keywords and parameters.

FORMAT <STATEMENT_FORMAT> *[[alternate-heading]*
[\ MULTIPLE]]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading for this use of the <STATEMENT_FORMAT> tag. The default heading provided by VAX DOCUMENT is "Format." See the reference description of the <SET_TEMPLATE_HEADING> tag for information on how to modify the default headings for all <STATEMENT_FORMAT> tags.

MULTIPLE

Indicates that the statement has more than one format, and that each format will be introduced using the <FORMAT_SUBHEAD> tag. This keyword is valid only as the second argument to the <STATEMENT_FORMAT> tag.

related tags

- <FCMD>
- <FFUNC>
- <FPARMS>
- <CONSTRUCT_LIST>
- <STATEMENT_LINE>
- <FORMAT_SUBHEAD>

required terminator

<ENDSTATEMENT_FORMAT>

DESCRIPTION

The <STATEMENT_FORMAT> tag creates sections of text that describe the format of a statement or function. Like the global <FORMAT> tag, the <STATEMENT_FORMAT> tag enables the <FCMD> and <FPARMS> tags to label specific portions of a statement format section.

The following is a list of some of the most regularly used statement format section tag combinations:

- <FCMD>(statement-keyword) <FPARMS>(parameter-list)

This is the standard form in which the statement keyword and its parameter list are separated by a blank space. If the parameter list, on output, is more than a single line, additional lines are aligned at the beginning of the parameter list.

- <FCMD>(statement-keyword\parameter-list)

SOFTWARE Doctype Tag Reference

<STATEMENT_FORMAT>

This form should be used for statement functions, in which a statement and its parameters are not separated by blank spaces.

Following the formatted statement, the <CONSTRUCT_LIST> and <STATEMENT_LINE> tags can be used to expand on the meanings of variable names specified in the format. Such variable names can be coded using the <VARIABLE> or <KEYWORD> tags.

EXAMPLES

The following example shows a statement format section that uses the <FFUNC> tag to present the format of a statement.

```
1 <STATEMENT_FORMAT>
  <FFUNC>(real-vbl\=ABS<VARIABLE>((real-exp)))
  <endstatement_format>
```

This example may produce the following output.

Format

real-vbl=ABS(real-exp)

The following example illustrates the use of multiple formats for a single statement, with special headings for each. Note the special use of the <FORMAT_SUBHEAD> tag to introduce each specific format.

```
2 <statement_format>(\multiple)
  <FORMAT_SUBHEAD>(String Variable To Array)
  <FCMD>(CHANGE) <FPARMS>(str-exp <KEYWORD>(TO) num-array)
  <FORMAT_SUBHEAD>(Array to String Variable)
  <FCMD>(CHANGE) <FPARMS>(num-array <KEYWORD>(TO) str-vbl)
  <endstatement_format>
```

This example may produce the following output.

Format

String Variable To Array

CHANGE str-exp TO num-array

Array to String Variable

CHANGE num-array TO str-vbl

SOFTWARE Doctype Tag Reference

<STATEMENT_LINE>

<STATEMENT_LINE>

Indicates the position of a valid statement line within the context of a statement format or a construct list.

FORMAT <STATEMENT_LINE> [(text[\ *INDENT*])]

ARGUMENTS *text*
Specifies the text for a valid statement line. If no text is specified, the default statement line output is as follows:

[statement] . . .

INDENT

Indicates that the statement line is to be indented from the margin at which the current statement format or construct list is being set.

related tags

- <FCMD>
- <FPARMS>
- <CONSTRUCT>

restrictions

Valid only in the context of the Statement reference template.

required terminator

None.

EXAMPLES

The following example shows how to use <STATEMENT_LINE> within a formatted statement section.

```
1 <STATEMENT_FORMAT>
  <FCMD>(RECORD) <FPARMS>(rec-nam)
  <STATEMENT_LINE>(rec-component)
  <ELLIPSIS>
  <FCMD>(END RECORD) <FPARMS>([ rec-nam ])
  <ENDSTATEMENT_FORMAT>
```

This example may produce the following output.

Format

```
RECORD  rec-nam
          rec-component
          .
          .
          .
END RECORD  [ rec-nam ]
```

The following example shows how to use the <STATEMENT_LINE> tag within a construct list. Note that the first use of the tag specifies INDENT as the second argument.

```
2 <STATEMENT_FORMAT>
  <CONSTRUCT_LIST>(group-clause:)
  <construct>(group-clause:)
    <keyword>(GROUP) group-nam [ ( int-const,<hellipsis> ) ]
    <statement_line>(rec-component\indent)
    <ellipsis>
    <statement_line>(<keyword>(END GROUP) [ group-nam ])
  <ENDCONSTRUCT_LIST>
<ENDSTATEMENT_FORMAT>
```

This example may produce the following output.

Format

```
group-clause:  GROUP group-nam [ ( int-const, . . . ) ]
                rec-component
                .
                .
                .
END GROUP [ group-nam ]
```

SOFTWARE Doctype Tag Reference

<STATEMENT_LINE>

The following example shows the default output for the <STATEMENT_LINE> tag when it is used in a statement format section.

```
3 <statement_format>
  <FCMD>(SUB) <FPARMS>(sub-name [ pass-mech ] [ ( [ formal-param ],
    <hellipsis> ) ] )
  <statement_line>
  <FCMD>(<list>(stacked\braces)
    <le>END SUB
    <le>SUBEND<endlist>) <FPARMS>()
  <endstatement_format>
```

This example may produce the following output.

Format

```
SUB sub-name [ pass-mech ] [ ( [ formal-param ], . . . ) ]
      [ statement ]...
```

```
{ END SUB }
{ SUBEND }
```

<STATEMENT_SECTION>

Begins a statement reference section, enables tags reserved for use in statement sections, and sets paging attributes.

FORMAT <STATEMENT_SECTION> *[[[**running-title**]
[\ *number-prefix*]
[\ **NEWPAGE**]]]*

ARGUMENTS ***running-title***
Specifies a top-level running heading to be used throughout the statement section. If this argument is not specified, the running headings are determined as described in Section 8.4.2.

number-prefix
Specifies a character-string prefix to be used to construct page numbers (folios) and formal figure, table, and example numbers. If this argument is not specified, the page and formal element numbering are determined as described in Section 8.4.2.

NEWPAGE
Indicates that the statement section should begin on a new page. This argument is only meaningful in two cases:

- When you have previously entered the <SET_TEMPLATE_STATEMENT> tag with the NONEWPAGE keyword to specify that each new statement in this statement section should not begin on a new page
- When you want to place one or more pages of text between the end of a part page and the beginning of a statement section.

related tags

- <SET_TEMPLATE_STATEMENT>
- <SET_TEMPLATE_PARA>
- <SET_TEMPLATE_LIST>
- <SET_TEMPLATE_TABLE>
- <SET_TEMPLATE_HEADING>
- <STATEMENT>
- <FUNCTION>
- <OVERVIEW>
- <STATEMENT_FORMAT>
- <FORMAT_SUBHEAD>
- <FCMD>
- <FPARMS>

SOFTWARE Doctype Tag Reference

<STATEMENT_SECTION>

- <FFUNC>
- <CONSTRUCT_LIST>
- <CONSTRUCT>
- <STATEMENT_LINE>

restrictions

None.

required terminator

<ENDSTATEMENT_SECTION>

DESCRIPTION

The <STATEMENT_SECTION> tag begins the statement reference template. You can locate a statement section within a chapter or an appendix, or following a part page (that is, within a document section begun with the <PART_PAGE> tag). You code a statement section in a chapter or an appendix in the same manner; statement sections in parts are handled differently.

If your statement section follows a part page, and you include text between the part page and the statement section, specify the NEWPAGE keyword as the third argument to the <STATEMENT_SECTION> tag. This causes the statement section to begin on a new page. The following code fragment shows a statement section that begins on a new page:

```
<STATEMENT_SECTION>(\SD\NEWPAGE)
<HEAD1>(Statement Format)
```

When you use the <STATEMENT_SECTION> tag within a chapter or an appendix, and you want to place text after the statement section in that chapter or appendix, you must end the statement section with the <ENDSTATEMENT_SECTION> tag and place the text after that tag. By default, this text begins on a new page of output.

Specify the NONEWPAGE argument to the <ENDSTATEMENT_SECTION> tag if you do not want the text to begin on a new page of output. The following code fragment shows the end of a statement section that specifies that the subsequent text not be placed on a new page:

```
<ENDSTATEMENT_SECTION>(NONEWPAGE)
```

When the <ENDSTATEMENT_SECTION> tag is specified within the context of a chapter or appendix, it resets the default running titles to those in effect for the chapter or appendix, so the last page of the last statement description in the statement section may not carry the last statement's name as the running heading. Instead it may carry the running title used by the chapter or appendix.

The <STATEMENT_SECTION> tag can be used more than once in a document. By specifying arguments to that tag, and by using the <SET_TEMPLATE_STATEMENT> tag to specify additional attributes, you can tailor statement sections that meet the specific requirements of your documentation.

SOFTWARE Doctype Tag Reference

<STATEMENT_SECTION>

EXAMPLES

The following example shows how to begin a statement section within a document part.

```
1 <PART>
  <PART_PAGE>
  <TITLE>(Part III\BASIC Statements and Functions)
  <ENDPART_PAGE>(RENUMBER)
  <STATEMENT_SECTION>(Statements and Functions\SF)
  <SET_TEMPLATE_STATEMENT>(BASIC_STATEMENT)

  <BASIC_STATEMENT>(GROUP)
  <OVERVIEW>
  Creates a group within a database.
  <ENDOVERVIEW>
.
.
  <ENDSTATEMENT_SECTION>
```

The tags in the previous example perform the following functions:

- The global <PART> tag begins the part.
- The global <PART_PAGE> tag creates a part page.
- The global <TITLE> tag is used in the context of the <PART_PAGE> tag to create a title on the part page.
- The RENUMBER argument to the global <ENDPART_PAGE> tag specifies that the pages should be renumbered beginning with the part page. This causes the first page of text following the part page to be numbered page 3 (Page 1 is the unnumbered page the part page title is placed on, page 2 is the back of page 1, and page 3 is the first numbered page after the part page).
- The <STATEMENT_SECTION> tag begins the statement section and specifies the running title "Statements and Functions" as the running title for the statement section. If the <SET_TEMPLATE_STATEMENT> tag were used with the DOUBLERUNNINGHEADS argument, the title "Statements and Functions" would be used as the top running title.

The <STATEMENT_SECTION> tag also specifies that the prefix "SF" should be used to construct numbers for pages and for formal figures, tables, and examples within the statement section (for example, SF-11, SF-32, Table SF-1, Example SF-2, and so on).

- The <SET_TEMPLATE_STATEMENT> tag specifies that all statement descriptions in this statement section will be identified using the tag <BASIC_STATEMENT> rather than the default tags <FUNCTION> or <STATEMENT>. The <BASIC_STATEMENT> tag created by the <SET_TEMPLATE_STATEMENT> tag will have the default attributes of the default tags (the <FUNCTION> and <STATEMENT> tags).

The following example shows how you can create a statement section in which each statement description (begun with a <STATEMENT> or <FUNCTION> tag) is in a separate SDML file, and all these descriptions are included into a primary statement description file. For example, the file MYSTATEMENTS.SDML contains the following SDML tags:

```
<INCLUDE>(CLOSE_GROUP.SDML)
<INCLUDE>(OPEN_GROUP.SDML)
<INCLUDE>(READ_GROUP.SDML)
<INCLUDE>(WRITE_GROUP.SDML)
```

SOFTWARE Doctype Tag Reference

<STATEMENT_SECTION>

Each of the included files contains one statement reference description begun with a <STATEMENT> or <FUNCTION> tag. For these files to process correctly, they must be preceded with the <STATEMENT_SECTION> tag that enables the <STATEMENT> and <FUNCTION> tag. These files can have the necessary tags processed before them by specifying the /INCLUDE qualifier on the command line to include a startup definition file. This startup file might include the following tags:

2 <STATEMENT_SECTION>(Group Statements\GS)
<SET_TEMPLATE_STATEMENT>(STATEMENT\DOUBLERUNNINGHEADS)

If this startup file were named GS_STATEMENT_STARTUP.SDML, it could be included using the DOCUMENT /INCLUDE qualifier as in the following example:

```
$ DOCUMENT mystatements SOFT.REF LN03 /INCLUDE=GS_STATEMENT_STARTUP.SDML
```

When each individual file in MYSTATEMENTS.SDML is processed, the correct sequence of tags is read in to begin the statement section.

You can process multiple files together by using the <INCLUDE> tag to include them into a single master file (such as MYSTATEMENTS.SDML), or you can include them into a book build profile.

Use the <ELEMENT> tags to include multiple files into a profile. For example, the book build profile file GS_STATEPRO.SDML could contain the following tags:

```
<PROFILE>  
<ELEMENT>(CLOSE_GROUP.SDML)  
<ELEMENT>(OPEN_GROUP.SDML)  
<ELEMENT>(READ_GROUP.SDML)  
<ELEMENT>(WRITE_GROUP.SDML  
    <COMMENT>( **contains <ENDSTATEMENT_SECTION> tag**)  
<ENDPROFILE>
```

Note that the profile file should include the <ENDSTATEMENT_SECTION> tag in the appropriate file, so that the template will be terminated and the book build will process correctly.

14.6 The Tag Reference Template Tags

This section alphabetically describes the tags available in the Tag template.

Use these tags to create a tag reference section. In most manuals, a tag reference section is an encyclopedic reference section that describes each tag the software system offers, along with the tag's format, its arguments, and examples of its use.

SOFTWARE Doctype Tag Reference

<FORMAT>

<FORMAT>

Begins a section that highlights the syntax of a tag, including keywords and arguments.

FORMAT <FORMAT> [(*alternate-heading*)]

ARGUMENTS *alternate-heading*

Specifies a heading to override the current default text heading for this use of the <FORMAT> tag. The default heading provided by VAX DOCUMENT is "Format." See the <SET_TEMPLATE_HEADING> tag description for information on how to modify the default headings for all <FORMAT> tags.

related tags

- <FTAG>
- <TAG_SECTION>

restrictions

None.

required terminator

<ENDFORMAT>

DESCRIPTION

The <FORMAT> tag creates sections of text that describe the format of a command, routine, or tag. This tag is a global tag and is not restricted to tag sections. However, in the context of the <ROUTINE_SECTION> and <TAG_SECTION> tags, the <FORMAT> tag enables certain additional tags for specialized format descriptions. These additional tags are not available outside of the context of routine or tag sections.

The global <FORMAT> tag enables the <FCMD>, <FPARMS>, and <FPARM>, which label specific portions of a format statement. In a tag section, the <FORMAT> tag also enables the <FTAG> tag to describe the syntax of VAX DOCUMENT tags.

EXAMPLE

The following example illustrates a simple format section using the <FTAG> tag to describe the format of the <FRUIT_TYPE> tag.

```
<FORMAT>
<FTAG>(FRUIT_TYPE\fruit-arg)
<ENDFORMAT>
```

This example may produce the following output.

FORMAT <FRUIT_TYPE> (*fruit-arg*)

<FTAG>

Specifies the name of a tag and its arguments within the context of a <FORMAT> tag.

FORMAT <FTAG> (*tag-name*[\ *argument-list* [\ *OPTIONAL*]])

ARGUMENTS *tag-name*

Is the name of the tag. This name must be a valid tag name.

argument-list

Lists the arguments, if any. If this argument is not specified, it indicates that the tag has no arguments.

When you specify arguments, you can use the <ARG_SEP> tag to denote the argument separator character, the backslash (\).

OPTIONAL

Indicates that all arguments are optional. When this keyword is specified, the *argument-list* argument is placed in square brackets to indicate that those arguments are optional.

related tags

- <FORMAT>
- <ARG_SEP>

restrictions

Valid only within the context of a <FORMAT> tag section in the Tag Template.

required terminator

None.

DESCRIPTION

The <FTAG> tag is used to describe the format of a tag and its arguments within a format section. This tag uses the <ARG_SEP> tag to create tag separator characters (\) within the *argument-list* argument.

The <ARG_SEP> tag has no other function than to output the tag separator character (the backslash) within a Tag template format section.

SOFTWARE Doctype Tag Reference

<FTAG>

EXAMPLES

The following example shows how you can use the <FTAG> tag to show the syntax of a tag that has no argument list.

1

```
<format>
<ftag>(OVERVIEW)
<endformat>
```

This example may produce the following output.

FORMAT

<OVERVIEW>

The following example illustrates how to specify the format of a tag that has a single optional argument by using the OPTIONAL keyword.

2

```
<format>
<ftag>(DESCRIPTION\heading-text\OPTIONAL)
<endformat>
```

This example may produce the following output.

FORMAT

<DESCRIPTION> [(*heading-text*)]

The following example illustrates how to specify a tag that has one required and two optional arguments. Note how the <ARG_SEP> tag is used to separate the arguments.

3

```
<format>
<ftag>(ROUTINE\name[<arg_sep>info-1[<arg_sep>info2]])
<endformat>
```

This example may produce the following output.

FORMAT

<ROUTINE> (*name*[\ *info-1*[\ *info2*])]

The following example shows a tag format in which all of the tag's arguments are optional and positional.

4

```
<format>
<ftag>(ROUTINE_SECTION\[running-title]<line>
[<arg_sep>prefix]<line>
[<arg_sep>NEWPAGE]\OPTIONAL)
<endformat>
```

This example may produce the following output.

FORMAT

**<ROUTINE_SECTION> [(*running-title*)
[\ *prefix*]
[\ *NEWPAGE*])]**

<RELATED_ITEM>

Provides a text description of a set of tags or the use of a set of tags that may be related to the tag being described.

FORMAT <RELATED_ITEM>

ARGUMENTS *None.*

related tags • <RELATED_TAGS>
 • <RELATED_TAG>

restrictions Valid only within the context of a <RELATED_TAGS> tag section.

required terminator *None.*

DESCRIPTION The <RELATED_ITEM> tag lets you introduce text into a <RELATED_TAGS> tag section. This text begins right after the <RELATED_ITEM> tag and is terminated by the next related tag section tag.

EXAMPLE The following example shows how to enter text describing information related to the use of a tag.

```
<RELATED_TAGS>  
<RELATED_ITEM> The <TAG>(INTRO) and <TAG>(ENDINTRO) tags  
may be used to label the introductory material in your book.  
<ENDRELATED_TAGS>
```

This example may produce the following output.

related tags • The <INTRO> and <ENDINTRO> tags may be used to label the introductory material in your book.

SOFTWARE Doctype Tag Reference

<RELATED_TAG>

<RELATED_TAG>

Specifies a single tag that is related to the current tag.

FORMAT <RELATED_TAG> (*tag-name*)

ARGUMENTS *tag-name*
Specifies the name of a VAX DOCUMENT tag. Do not place angle brackets around the tag name, these are supplied by the <RELATED_TAG> tag upon output.

related tags • <RELATED_TAGS>
 • <RELATED_ITEM>

restrictions Valid only within the context of the <RELATED_TAGS> tag.

required terminator *None.*

DESCRIPTION Use the <RELATED_TAG> tag to list a single related tag in the context of the <RELATED_TAGS> tag. This tag automatically places angle brackets around the *tag-name* argument, so angle brackets should not be specified.

Use the <RELATED_ITEM> tag to list related tag information in another format.

EXAMPLE The following example specifies the names of two related tags in the context of the <RELATED_TAGS> tag. Note how the names of the related tags are specified without the angle brackets.

```
<RELATED_TAGS>  
<RELATED_TAG>(TAG_SECTION)  
<RELATED_TAG>(SET_TEMPLATE_TAG)  
<ENDRELATED_TAGS>
```

This example may produce the following output.

related tags • <TAG_SECTION>
 • <SET_TEMPLATE_TAG>

<RELATED_TAGS>

Provides a summary of tags whose use is related to the tag being described.

FORMAT <RELATED_TAGS> [(NONE)]

ARGUMENTS *NONE*

Indicates that there are no tags whose use is related to the tag being described. If the NONE keyword argument is used, do not specify the <ENDRELATED_TAGS> tag.

related tags

- <RELATED_TAG>
- <RELATED_ITEM>

restrictions

If NONE is specified, you must not specify <ENDRELATED_TAGS> .

required terminator

<ENDRELATED_TAGS>

EXAMPLES

The following example shows how to specify two related tags in a <RELATED_TAGS> tag section.

```
❏ <RELATED_TAGS>
   <RELATED_TAG>(TAG_SECTION)
   <RELATED_TAG>(SET_TEMPLATE_TAG)
   <ENDRELATED_TAGS>
```

This example may produce the following output.

related tags

- <TAG_SECTION>
- <SET_TEMPLATE_TAG>

The following example shows how to use the NONE keyword to show that there are no related tags. Note that, in this case, the <ENDRELATED_TAGS> tag is omitted.

SOFTWARE Doctype Tag Reference

<RELATED_TAGS>

2 <RELATED_TAGS>(NONE)

This example may produce the following output.

related tags

None.

The following example shows how text describing information related to the use of the tag can be entered into the <RELATED_TAGS> tag section using the <RELATED_ITEM> tag.

3 <RELATED_TAGS>
<RELATED_ITEM>The <tag>(SIDE_EFFECTS) and <tag>(ENDSIDE_EFFECTS) tags
accept the same arguments as the <tag>(45RPM) tag.
<ENDRELATED_TAGS>

This example may produce the following output.

related tags

- The <SIDE_EFFECTS> and <ENDSIDE_EFFECTS> tags accept the same arguments as the <45RPM> tag.

<RESTRICTIONS>

Provides a summary of restrictions on the use of a tag.

FORMAT

<RESTRICTIONS> [({ *alternate-heading*
NONE
LIST
alternate-heading \ LIST })]

ARGUMENTS

alternate-heading

Specifies a heading to override the current default text heading for this use of the <RESTRICTIONS> tag. The default heading provided by VAX DOCUMENT is "Restrictions." See the reference description of the <SET_TEMPLATE_HEADING> tag for information on how to modify the default headings for all <RESTRICTIONS> tags.

NONE

Indicates that there are no restrictions on the use of the tag. If the NONE keyword is used, do not use the <ENDRESTRICTIONS> tag to end the <RESTRICTIONS> tag section.

LIST

Indicates that a number of restrictions are to be listed. To list the restrictions, use the <RITEM> for each of the individual restriction items in the list.

related tags

- <TAG_SECTION>
- <RITEM>

restrictions

If NONE is specified, you must not specify the <ENDRESTRICTIONS> tag.

required terminator

<ENDRESTRICTIONS>

EXAMPLES

The following example shows how to set a simple paragraph of text for the restrictions section using the <RESTRICTIONS> tag.

```
❏ <RESTRICTIONS>  
Valid only in the context of the  
<tag>(COMMAND_SECTION) tag.  
<ENDRESTRICTIONS>
```

This example may produce the following output.

restrictions

Valid only in the context of the <COMMAND_SECTION> tag.

SOFTWARE Doctype Tag Reference

<RESTRICTIONS>

The following example shows how to use the NONE keyword to indicate that there are no restrictions on the use of a tag.

2 <RESTRICTIONS>(NONE)

This example may produce the following output.

restrictions

None.

The following example shows how to create a set of restrictions in a list.

3 <RESTRICTIONS>(LIST)
<RITEM>If you specify NONE, you must not specify <tag>(ENDPARAMDEFLIST).
<RITEM>Use of a default heading is restricted to the reference templates.
<ENDRESTRICTIONS>

This example may produce the following output.

restrictions

- If you specify NONE, you must not specify <ENDPARAMDEFLIST> .
- Use of a default heading is restricted to the reference templates.

<RITEM>

Labels an item in a list of restrictions.

FORMAT <RITEM>

related tags • <RESTRICTIONS>

restrictions Can only be used in the context of the <RESTRICTIONS> tag.

required terminator *None.*

EXAMPLE The following example shows how to create a list of restrictions. Note how the <RESTRICTIONS> tag is specified with the LIST keyword argument. Note how this enables the <RITEM> tag.

```
<RESTRICTIONS>(LIST)
<RITEM>You must not unplug this appliance while it is operating.
<RITEM>This appliance should not be immersed in water.
<ENDRESTRICTIONS>
```

This example may produce the following output.

restrictions

- You must not unplug this appliance while it is operating.
- This appliance should not be immersed in water.

SOFTWARE Doctype Tag Reference

<SDML_TAG>

<SDML_TAG>

Begins a new tag description.

FORMAT <SDML_TAG> (*tag-name*)

ARGUMENTS *tag-name*
Specifies the name of the tag to be described.

related tags

- <SET_TEMPLATE_TAG>
- <TAG_SECTION>

required terminator *None.*

DESCRIPTION Use the <SDML_TAG> tag to begin a description section for a single SDML tag. The <SDML_TAG> tag has the following default format:

- Each <SDML_TAG> tag begins a new page of output.
- Each output page carries a single running title, which is the current SDML tag name.

Use the <SET_TEMPLATE_TAG> tag to replace the <SDML_TAG> tag with a tag whose name is more meaningful to you (for example, <LOCAL_TAG>), or if you wish to change the default attributes of the <SDML_TAG> tag. See the description of the <SET_TEMPLATE_TAG> tag in this chapter for more information.

EXAMPLE The following example shows the Tag template begun using the <TAG_SECTION> tag. Within this tag section the <SDML_TAG> tag is used to begin the tag description for the local <LEVEL1> tag.

```
<TAG_SECTION>(Local Tags)
<SDML_TAG>(LEVEL1)
<OVERVIEW>
Labels the first level of a diagram.
<ENDOVERRIDE>
```


SOFTWARE Doctype Tag Reference

<SET_TEMPLATE_TAG>

**required
terminator**

None.

DESCRIPTION

The <SET_TEMPLATE_TAG> tag allows you to specify the name by which you want to refer to tags that you are describing in the context of a template.

This tag also allows you to alter the default attributes associated with the <SDML_TAG> tag (or the tag you replaced with the <SDML_TAG> tag using the <SET_TEMPLATE_TAG> tag).

EXAMPLE

The following example shows how to use the <SET_TEMPLATE_TAG> tag to alter the default format of the Tag template.

```
<TAG_SECTION>(Tag Template Tags)
<SET_TEMPLATE_TAG>(LOCAL_TAG\DOUBLERUNNINGHEADS)
<LOCAL_TAG>(LEVEL1)
```

This tag sequence begins a tag section and defines the tag <LOCAL_TAG> for introducing new tag descriptions. These attributes are as follows:

- Each tag description begins on a new page.
- Each page carries a two-line running title. The top line is "Tag Template Tags" and the second line is the name of the tag description that is current at the top of the page.

SOFTWARE Doctype Tag Reference

<TAG_SECTION>

- <RELATED_ITEM>
- <RESTRICTIONS>
- <RITEM>
- <TERMINATING_TAG>
- <DESCRIPTION>
- <EXAMPLE_SEQUENCE>

restrictions

None.

required terminator

<ENDTAG_SECTION>

DESCRIPTION

The <TAG_SECTION> tag begins the tag reference template. You can locate a tag section within a chapter or an appendix, or following a part page (that is, within a document section begun with the <PART_PAGE> tag). You code a tag section in a chapter or an appendix in the same manner; tag sections in parts are handled differently.

If your tag section follows a part page, and you include text between the part page and the tag section, specify the NEWPAGE keyword as the third argument to the <TAG_SECTION> tag. This causes the tag section to begin on a new page. The following code fragment shows a tag section that begins on a new page:

```
<TAG_SECTION>(\TD\NEWPAGE)
<HEAD1>(Tag Dictionary)
```

When you use the <TAG_SECTION> tag within a chapter or an appendix, and want to place text after the tag section in that chapter or appendix, you must end the tag section with the <ENDTAG_SECTION> tag and place the text after that tag. By default, this text begins on a new page of output.

Specify the NONEWPAGE argument to the <ENDTAG_SECTION> tag if you do not want the text to begin on a new page of output. The following code fragment shows the end of a tag section that specifies that the subsequent text not be placed on a new page:

```
<ENDTAG_SECTION>(NONEWPAGE)
```

When the <ENDTAG_SECTION> tag is specified within the context of a chapter or appendix, it resets the default running titles to those in effect for the chapter or appendix, so the last page of the last tag description in the tag section may not carry the last tag's name as the running heading. Instead it may carry the running title used by the chapter or appendix.

EXAMPLES

The following example shows how to begin a tag section within a document part.

```
1 <PART>
  <PART_PAGE>
  <TITLE>(Part III\Tag Dictionary)
  <ENDPART_PAGE>(RENUMBER)
  <TAG_SECTION>(Tag Dictionary\TD)
  <SET_TEMPLATE_TAG>(LOCAL_TAG)

  <LOCAL_TAG>(SITETAG)

  <OVERVIEW>
  This is a site-specific tag.
  <ENDOVERVIEW>
  .
  .
  <ENDTAG_SECTION>
```

The tags in the previous example perform the following functions:

- The global <PART> tag begins the part.
- The global <PART_PAGE> tag creates a part page.
- The global <TITLE> tag is used in the context of the <PART_PAGE> tag to create a title on the part page.
- The RENUMBER argument to the global <ENDPART_PAGE> tag specifies that the pages should be renumbered beginning with the part page. This causes the first page of text following the part page to be numbered page 3 (Page 1 is the unnumbered page the part page title is placed on, page 2 is the back of page 1, and page 3 is the first numbered page after the part page).
- The <TAG_SECTION> tag begins the tag section and specifies the running title "Tag Dictionary" as the running title for the tag section. If the <SET_TEMPLATE_TAG> tag were used with the DOUBLERUNNINGHEADS argument, the title "Tag Dictionary" would be used as the top running title.

The <TAG_SECTION> tag also specifies that the prefix "TD" should be used to construct numbers for pages and for formal figures, tables, and examples within the tag section (for example, TD-11, TD-32, Table TD-1, Example TD-2, and so on).

- The <SET_TEMPLATE_TAG> tag specifies that all tag descriptions in this tag section will be identified using the tag <LOCAL_TAG> rather than the default tag <TAG>. The <LOCAL_TAG> tag will have the default attributes of the <TAG> tag.

The following example shows how you can create a tag section in which each tag description (begun with a <SDML_TAG> tag) is in a separate SDML file, and all these descriptions are included into a primary routine description file. For example, the file MYTAGS.SDML contains the following SDML tags:

```
<INCLUDE>(CLOSE_FILE.SDML)
<INCLUDE>(OPEN_FILE.SDML)
<INCLUDE>(READ_FILE.SDML)
<INCLUDE>(WRITE_FILE.SDML)
```

SOFTWARE Doctype Tag Reference

<TAG_SECTION>

Each of the included files contains one tag reference description begun with a <SDML_TAG> tag. For these files to process correctly, they must be preceded with the <TAG_SECTION> tag that enables the <SDML_TAG> tag. These files can have the necessary tags processed before them by specifying the /INCLUDE qualifier on the command line to include a startup definition file. This startup file might include the following tags.

2 <TAG_SECTION>(File Handling Tags\TAGS)
<SET_TEMPLATE_TAG>(SDML_TAG\DOUBLERUNNINGHEADS)

If this startup file were named FILE_TAG_STARTUP.SDML, it could be included using the DOCUMENT /INCLUDE qualifier as in the following example:

```
$ DOCUMENT mytags SOFT.REF LN03 /INCLUDE=FILE_TAG_STARTUP.SDML
```

When each individual file in MYTAGS.SDML is processed, the correct sequence of tags will be read in to begin the tag section.

You can process multiple files together by using the <INCLUDE> tag to include them into a single master file (such as MYTAGS.SDML), or you can include them into a book build profile.

You use the <ELEMENT> tags to include multiple files into a profile. For example, the book build profile file TAGPRO.SDML could contain the following tags:

```
<PROFILE>  
<ELEMENT>(CLOSE_FILE.SDML)  
<ELEMENT>(OPEN_FILE.SDML)  
<ELEMENT>(READ_FILE.SDML)  
<ELEMENT>(WRITE_FILE.SDML) <COMMENT>(contains <ENDTAG_SECTION> tag)  
<ENDPROFILE>
```

Note that the profile file should include the <ENDTAG_SECTION> tag in the appropriate file, so that the template will be terminated and the book build will process correctly.

<TERMINATING_TAG>

Specifies the required terminator for a tag.

FORMAT

<TERMINATING_TAG> ({ *tag-name*
[\ *additional-text*]
NONE })

ARGUMENTS

tag-name

Specifies the name of the terminating tag.

NONE

Indicates that there is no terminating tag.

additional-text

Specifies additional text that you can insert to briefly explain the terminating tag.

related tags

- <TAG_SECTION>
- <RELATED_TAG>

restrictions

Can only be used in the context of the <TAG_SECTION> tag.

required terminator

None.

DESCRIPTION

The <TERMINATING_TAG> tag specifies the tag required to terminate the tag that is being described. You can provide additional information about the terminating tag by specifying this text as the second argument to the <TERMINATING_TAG> tag.

Use the NONE keyword argument to explicitly specify that no terminating tag is needed. This keyword places the text "None." beneath the heading output by this tag.

SOFTWARE Doctype Tag Reference

<TERMINATING_TAG>

EXAMPLES

The following example shows a terminating tag specified with both the *tag-name* argument and also the *additional-text* argument.

1 <TERMINATING_TAG>(ENDRECORDLIST\<p>This tag should be omitted if the NONE keyword is used with the <TAG>(RECORDLIST) tag.)

This example may produce the following output.

<ENDRECORDLIST>

required terminator

This tag should be omitted if the NONE keyword is used with the <RECORDLIST> tag.

The following example shows the <TERMINATING_TAG> tag used with the NONE keyword.

2 <TERMINATING_TAG>(NONE)

This example may produce the following output.

required terminator

None.

Index

A

- <ABSTRACT> tag • 2-1, 2-4
Reference • 9-2
- <ACKNOWLEDGMENTS> tag • 2-1, 2-4
Reference • 9-3
- <ARGDEFLIST> tag
Reference • 14-2 to 14-4
- <ARGDEF> tag
Reference • 14-5
- <ARGITEM> tag
Reference • 14-6 to 14-7
- <ARGTEXT> tag
Reference • 14-126
- <ARGUMENT> tag
Reference • 14-8
- <ARG_SEP> tag
See <FTAG> tag
- ARTICLE doctype • 2-1 to 2-12
 - abstracts • 2-4
 - acknowledgments • 2-4
 - author information • 2-3 to 2-4
 - back notes • 2-6
 - bibliographies • 2-7
 - headings • 2-5
 - improving format of
 - See Two-column doctype
 - quotations • 2-5
 - reference notes • 2-7
 - running headings • 2-5
 - sample output file • 2-12
 - sample SDML file • 2-11 to 2-12
 - source notes • 2-4
 - subtitles • 2-3
 - tags • 9-1 to 9-32
 - titles • 2-3
- <AUTHOR> tag • 2-1, 2-3
 - in ARTICLE doctype
Reference • 9-4 to 9-5
 - in REPORT doctype • 7-1
Reference • 13-2 to 13-3
 - in SOFTWARE doctype
Reference • 14-9 to 14-10
- <AUTHOR_ADDR> tag • 2-1, 2-3
Reference • 9-6 to 9-7

- <AUTHOR_AFF> tag • 2-1, 2-3
Reference • 9-8
 - <AUTHOR_INFO> tag • 6-4
Reference • 12-2
 - <AUTHOR_LIST> tag • 2-1, 2-3
Reference • 9-9
 - <AUTO_NUMBER> tag • 6-4
Reference • 12-3
-

B

- <BACK_NOTES> tag • 2-1
Reference • 9-12
 - <BACK_NOTE> tag • 2-1
Reference • 9-10 to 9-11
 - <BIBLIOGRAPHY> tag • 2-1, 2-7
Reference • 9-13
 - <BIB_ENTRY> tag • 2-1, 2-7
Reference • 9-14
 - <BYLINE> tag
 - in REPORT doctype • 7-1
Reference • 13-4 to 13-5
 - in SOFTWARE doctype
Reference • 14-11 to 14-12
-

C

- <CCLIST> tag • 3-1
Reference • 10-4
- <CC> tag • 3-1
Reference • 10-2 to 10-3
- <CHEAD> tag
 - in LETTER doctype • 3-1
Reference • 10-5 to 10-6
 - in REPORT doctype • 7-2
Reference • 13-6 to 13-7
- <CLOSING> tag • 3-1
Reference • 10-7
- Code fragments
 - See SOFTWARE doctype
- <COLUMN> tag • 2-1
 - in ARTICLE doctype
Reference • 9-15 to 9-16

Index

- <COLUMN> tag (cont'd.)
 - in REPORT doctype • 7–2
 - Reference • 13–8 to 13–9
- <COMMAND> tag
 - Reference • 14–95 to 14–96
- Command template
 - See SOFTWARE doctype
- <COMMAND_SECTION> tag
 - Reference • 14–97 to 14–100
- <CONSTRUCT> tag
 - Reference • 14–152
- <CONSTRUCT_LIST> tag
 - Reference • 14–154 to 14–155
- <CPOS> tag
 - Reference • 14–13

D

Data Item Description documents (DID)

See MILSPEC Doctype

- <DELETE_KEY> tag
 - Reference • 14–14
- <DESCRIPTION> tag
 - Reference • 14–81 to 14–82

DID documents

See MILSPEC Doctype

- <DISPLAY> tag
 - Reference • 14–15 to 14–16
- <DISTLIST> tag • 3–1
 - Reference • 10–8

Doctype

- list of • 1–1
- two-column
 - See Two-column doctype

Doctype-specific tags • 1–1

- <DOCUMENT_ATTRIBUTES> tag
 - in ARTICLE doctype • 2–1, 2–5
 - Reference • 9–17 to 9–19
 - in REPORT doctype • 7–2
 - Reference • 13–10 to 13–12
 - in SOFTWARE doctype
 - Reference • 14–17 to 14–19

E

End notes

See <BACK_NOTES> tag

- <EXAMPLES_INTRO> tag
 - Reference • 14–22 to 14–23
 - <EXAMPLE_SEQUENCE> tag
 - Reference • 14–20 to 14–21
 - <EXC> tag
 - Reference • 14–24
 - <EXI> tag
 - Reference • 14–25 to 14–26
 - <EXTTEXT> tag
 - Reference • 14–27
- Extracts
- See <QUOTATION> tag

F

- <FARGS> tag
 - Reference • 14–129 to 14–130
- <FARG> tag
 - Reference • 14–127
- <FCMD> tag
 - in Command template • 14–101 to 14–103
 - in Statement template • 14–157 to 14–160
- <FFUNC> tag
 - in Routine template • 14–131 to 14–132
 - in Statement template • 14–160 to 14–161
- <FORMAT> tag
 - in Command template • 14–104 to 14–105
 - in Routine template • 14–133 to 14–134
 - in Tag template • 14–182
- <FORMAT_SUBHEAD> tag
 - Reference • 14–162 to 14–163
- <FPARMS> tag
 - in Command template • 14–108 to 14–109
 - in Statement template • 14–166 to 14–167
- <FPARM> tag
 - in Command template • 14–106 to 14–107
 - in Statement template • 14–164
- <FROM_ADDRESS> tag • 3–2
 - Reference • 10–9 to 10–10
- <FRTN> tag
 - Reference • 14–135 to 14–136
- <FTAG> tag
 - Reference • 14–183 to 14–184
- <FUNCTION> tag
 - Reference • 14–168

G

<GRAPHIC> tag
Reference • 14–28

H

<HEAD> tag
in LETTER doctype • 3–2
Reference • 10–11 to 10–12
in REPORT doctype • 7–2
Reference • 13–13 to 13–14

I

<INTRO_SUBTITLE> tag • 6–4
Reference • 12–4
<INTRO_TITLE> tag • 6–4
Reference • 12–5

K

<KEYPAD> tag
Reference • 14–32 to 14–34
<KEYPAD_ENDROW> tag
Reference • 14–35
<KEYPAD_ROW> tag
Reference • 14–36 to 14–37
<KEYPAD_SECTION> tag
Reference • 14–38 to 14–40
<KEY> tag
Reference • 14–29 to 14–30
<KEY_NAME> tag
Reference • 14–31
<KEY_PLUS> tag
Reference • 14–41
<KEY_SEQUENCE> tag
Reference • 14–42 to 14–43
<KEY_TYPE> tag
Reference • 14–44

L

LETTER doctype • 3–1 to 3–5
sample output file
of a letter • 3–5
of a memo • 3–3
sample SDML file
to create a letter • 3–5
to create a memo • 3–3
tags • 10–1 to 10–24
<LEVEL> tag • 7–2
Reference • 13–15

M

MANUAL doctype • 4–1 to 4–2
sample output file • 4–2
sample SDML file • 4–1 to 4–2

Memo

See LETTER doctype
<MEMO_DATE> tag • 3–2
Reference • 10–13 to 10–14
<MEMO_FROM> tag • 3–2
Reference • 10–15 to 10–16
<MEMO_HEADER> tag • 3–2
Reference • 10–17
<MEMO_LINE> tag • 3–2
Reference • 10–18 to 10–19
<MEMO_TO> tag • 3–2
Reference • 10–20 to 10–21

Messages

See SOFTWARE doctype
<MESSAGE_SECTION> tag
Reference • 14–45 to 14–47
<MESSAGE_TYPE> tag
Reference • 14–48

Military specifications

See MILSPEC Doctype
MILSPEC doctype • 5–1 to 5–7
DOD-STD-2167 documents • 5–3 to 5–6
MIL-STD-490A documents • 5–2 to 5–3
sample output file • 5–7
sample SDML file • 5–6 to 5–7
tags • 11–1 to 11–11
templates • 5–5
MIL-STD-490A documents
See MILSPEC doctype

Index

- <MSG> tag
 - Reference • 14–51 to 14–52
- <MSG> tag
 - Reference • 14–49 to 14–50
- <MSG_TEXT> tag
 - Reference • 14–53

N

Notes

- back notes
 - See <BACK_NOTES> tag
- reference notes
 - See <REF_NOTES> tag
- source notes
 - See <SOURCE_NOTE> tag

O

- <OUTLINE> tag • 7–2
 - Reference • 13–16 to 13–17
- OVERHEADS doctype • 6–3 to 6–5
 - sample output file • 6–5
 - sample SDML file • 6–5
 - tags • 12–1 to 12–16
- Overhead slide
 - See OVERHEADS doctype
- <OVERVIEW> tag
 - Reference • 14–84

P

- <PARAMDEFLIST> tag
 - Reference • 14–55 to 14–57
- <PARAMDEF> tag
 - Reference • 14–54
- <PARAMITEM> tag
 - Reference • 14–58 to 14–59
- <PROMPTS> tag
 - Reference • 14–112
- <PROMPT> tag
 - Reference • 14–110 to 14–111

Q

- <QPAIR> tag
 - Reference • 14–60
- <QUALDEFLIST> tag
 - Reference • 14–67 to 14–69
- <QUALDEF> tag
 - Reference • 14–66
- <QUALITEM> tag
 - Reference • 14–70 to 14–71
- <QUAL_LIST> tag
 - Reference • 14–61 to 14–63
- <QUAL_LIST_DEFAULT_HEADS> tag
 - Reference • 14–64
- <QUAL_LIST_HEADS> tag
 - Reference • 14–65
- <QUOTATION> tag • 2–1
 - Reference • 9–20

R

- Reference template
 - See SOFTWARE doctype
- <REF_NOTES> tag • 2–2, 2–7
 - Reference • 9–23
- <REF_NOTE> tag • 2–2, 2–7
 - Reference • 9–21 to 9–22
- <RELATED_ITEM> tag
 - Reference • 14–185
- <RELATED_TAGS> tag
 - Reference • 14–187 to 14–188
- <RELATED_TAG> tag
 - Reference • 14–186
- REPORT doctype • 7–1 to 7–3
 - improving format of REPORT.TWOCOL
 - See Two-column doctype
 - sample output file • 7–3
 - sample SDML file • 7–2 to 7–3
 - tags • 13–1 to 13–25
- <RESTRICTIONS> tag
 - in Command template • 14–113 to 14–114
 - in Tag template • 14–189 to 14–190
- <RETTEXT> tag
 - Reference • 14–137
- <RETURNS> tag
 - Reference • 14–138 to 14–139
- <RETURN_VALUE> tag
 - Reference • 14–115

<RITEM> tag
 in Command template • 14–116
 in Tag template • 14–191

<ROUTINE> tag
 Reference • 14–140 to 14–141

Routine template
 See SOFTWARE doctype

<ROUTINE_SECTION> tag
 Reference • 14–142 to 14–145

<RSDEFList> tag
 Reference • 14–146 to 14–147

<RSITEM> tag
 Reference • 14–148

Running headings
 at bottom of page
 See <RUNNING_FEET> tag
 at top of page
 See <RUNNING_TITLE> tag

<RUNNING_FEET> tag
 in ARTICLE doctype • 2–2, 2–5
 Reference • 9–24
 in OVERHEADS doctype • 6–4
 Reference • 12–6 to 12–7
 in REPORT doctype • 7–2
 Reference • 13–18
 in SOFTWARE doctype
 Reference • 14–72

<RUNNING_TITLE> tag
 in ARTICLE doctype • 2–2, 2–5
 Reference • 9–25 to 9–26
 in OVERHEADS doctype • 6–4
 Reference • 12–8 to 12–9
 in REPORT doctype • 7–2
 Reference • 13–19 to 13–20
 in SOFTWARE doctype
 Reference • 14–73 to 14–74

S

<SALUTATION> tag • 3–2
 Reference • 10–22

<SDML_TAG> tag
 Reference • 14–192

<SECTION> tag • 7–2
 Reference • 13–21 to 13–22

<SET_APPENDIX_NUMBER> tag • 5–1
 Reference • 11–2 to 11–3

<SET_TEMPLATE_COMMAND> tag
 Reference • 14–117 to 14–118

<SET_TEMPLATE_HEADING> tag
 Reference • 14–85 to 14–86

<SET_TEMPLATE_LIST> tag
 Reference • 14–87 to 14–88

<SET_TEMPLATE_PARA> tag
 Reference • 14–89 to 14–90

<SET_TEMPLATE_ROUTINE> tag
 Reference • 14–149 to 14–151

<SET_TEMPLATE_STATEMENT> tag
 Reference • 14–169 to 14–170

<SET_TEMPLATE_SUBCOMMAND> tag
 Reference • 14–119 to 14–120

<SET_TEMPLATE_TABLE> tag
 Reference • 14–91 to 14–94

<SET_TEMPLATE_TAG> tag
 Reference • 14–193 to 14–194

<SHOW_LEVELS> tag • 7–2
 Reference • 13–23 to 13–24

<SIGNATURES> tag
 in REPORT doctype • 7–1
 Reference • 13–25
 in SOFTWARE doctype
 Reference • 14–75

<SIGNATURE_LINE> tag • 5–1
 Reference • 11–4 to 11–5

<SIGNATURE_LIST> tag • 5–1
 Reference • 11–6 to 11–7

Slides
 See OVERHEADS doctype

<SLIDE> tag • 6–4
 Reference • 12–10 to 12–11

SOFTWARE doctype • 8–1 to 8–57
 arguments, parameters and qualifiers •
 8–14 to 8–17
 code fragments • 8–8 to 8–10
 interactive or code examples • 8–18 to 8–19
 messages
 See software messages • 8–10
 reference templates • 8–19 to 8–57
 Command template • 8–35 to 8–42
 sample output file • 8–39 to 8–42
 sample SDML file • 8–37 to 8–38
 Routine template • 8–42 to 8–48
 sample output file • 8–44 to 8–48
 sample SDML file • 8–43 to 8–44
 Statement template • 8–48 to 8–53
 sample output file • 8–50 to 8–53
 sample SDML file • 8–49
 Tag template • 8–53 to 8–57
 sample output file • 8–55 to 8–57
 sample SDML file • 8–54

Index

SOFTWARE doctype (cont'd.)

- software messages • 8–10 to 8–13
- tags • 14–1 to 14–200
 - in all of doctype • 14–1 to 14–79
 - in all templates • 14–80 to 14–94
 - in Command template • 14–94 to 14–125
 - in Routine template • 14–125 to 14–151
 - in Statement template • 14–151 to 14–181
 - in Tag template • 14–181 to 14–200
- terminal keys and keypads • 8–2 to 8–8
- Software messages
 - See SOFTWARE doctype
- Software specifications
 - See MILSPEC doctype
 - See SOFTWARE doctype
- <SOURCE_NOTE> tag • 2–2, 2–4
 - Reference • 9–27
- Specifications
 - for military
 - See MILSPEC doctype
 - for software
 - See SOFTWARE doctype
- <SPECIFICATION_INFO> tag • 5–1
 - Reference • 11–8 to 11–9
- <SPEC_TITLE> tag • 5–1
 - Reference • 11–10
- <STATEMENT> tag
 - Reference • 14–171
- Statement template
 - See SOFTWARE doctype
- <STATEMENT_FORMAT> tag
 - Reference • 14–172 to 14–173
- <STATEMENT_LINE> tag
 - Reference • 14–174 to 14–176
- <STATEMENT_SECTION> tag
 - Reference • 14–177 to 14–181
- <SUBCOMMAND> tag
 - Reference • 14–121 to 14–122
- <SUBCOMMAND_SECTION> tag
 - Reference • 14–123
- <SUBCOMMAND_SECTION_HEAD> tag
 - Reference • 14–124 to 14–125
- <SUBJECT> tag • 3–2
 - Reference • 10–23
- <SUBTITLE> tag • 6–4
 - in ARTICLE doctype • 2–2, 2–3
 - Reference • 9–28
 - in MILSPEC doctype • 5–1
 - Reference • 11–11
 - in OVERHEADS doctype
 - Reference • 12–12

- <SYNTAX> tag
 - Reference • 14–76 to 14–77
- <SYNTAX_DEFAULT_HEAD> tag
 - Reference • 14–78 to 14–80

T

- Tag template
 - See SOFTWARE doctype
- <TAG_SECTION> tag
 - Reference • 14–195 to 14–198
- Technical manual
 - for military specifications
 - See MILSPEC doctype
 - for software
 - See SOFTWARE doctype
 - general purpose
 - See MANUAL doctype
- Technical report
 - See REPORT doctype
- Templates
 - See MILSPEC doctype
 - See SOFTWARE doctype
- <TERMINATING_TAG> tag
 - Reference • 14–199 to 14–200
- <TEXT_SIZE> tag • 6–4
 - Reference • 12–13 to 12–14
- <TITLE> tag • 6–4
 - in ARTICLE doctype • 2–2, 2–3
 - Reference • 9–29
 - in OVERHEADS doctype
 - Reference • 12–15
- <TITLE_SECTION> tag • 2–2
 - Reference • 9–30
- <TOPIC> tag • 6–4
 - Reference • 12–16
- <TO_ADDRESS> tag • 3–2
 - Reference • 10–24
- Transparency
 - See OVERHEADS doctype
- Two-column doctype
 - improving the format of • 2–8

V

- <VITA> tag • 2–2, 2–3
 - Reference • 9–31 to 9–32

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

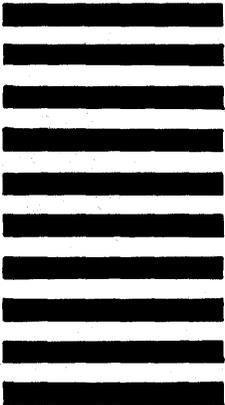


No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
The Manager, Office Program
ZK02-1/N20
110 SPIT BROOK ROAD
NASHUA, NH 03062 - 9990



Do Not Tear - Fold Here

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

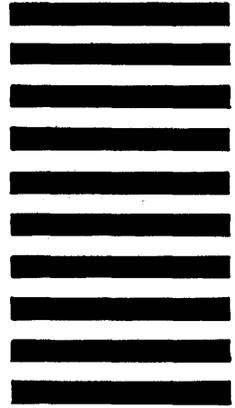
City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital



No Postage
Necessary
if Mailed
in the
United States



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
The Manager, Office Program
ZK02-1/N20
110 SPIT BROOK ROAD
NASHUA, NH 03062 - 9990



Do Not Tear - Fold Here