SH20-9161-3
File No. S370-73

Program Product

Document Composition Facility: Users Guide

IBM

SH20-9161-3
File No. S370-73

Program Product

Document Composition Facility: Users Guide

Program Number 5748-XX9

Release 2

This publication was produced using the IBM Document Composition Facility (program number 5748-XX9) and the master was printed on the IBM 3800 Printing Subsystem.

Fourth Edition (September 1982)

This is a major revison of SH20-9161-2, which is now obsolete. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to Release 2 of the Document Composition Facility program product, 5748-XX9, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

PREFACE

This manual contains a description of the IBM Document Composition Facility (SCRIPT/VS) program product and the information necessary to use it. No prior operating system knowledge is required for general use of SCRIPT/VS.

The information in this publication applies equally to OS/VS1, OS/VS2 MVS, DOS/VSE, VM/370, and ATMS-III unless specifically stated otherwise.

Use of SCRIPT/VS in an ATMS-III, CMS, or TSO environment requires the Foreground Environment Feature; use in a background environment requires the Document Library Facility program product (program number 5748-XXE).

The chapters of this publication contain:

- "Chapter 1. An Introduction to SCRIPT/VS" on page 1: A general description of SCRIPT/VS. This chapter includes a discussion of what SCRIPT/VS is and what it does.
- "Chapter 2. Using the SCRIPT Command" on page 13: A description of the SCRIPT command and each of its options.
- "Chapter 3. Basic Text Processing" on page 33: A description of how to specify basic text formatting functions, such as indention, tabs, blank space, forcing a new page, and formatting modes. This chapter also describes some guidelines for entering input lines in a SCRIPT/VS file.
- "Chapter 4. Defining a Page Layout" on page 55: A description of how to define the parameters of a page, such as page length, column width, line length, and page numbering. This chapter also describes how to have text repeated at the top and bottom of each page: running titles, headings, and footings.
- "Chapter 5. Multicolumn Page Layout" on page 67: A description of how to establish a multicolumn format for the body of a page.
- "Chapter 6. Head Levels and Table of Contents" on page 73: A description of how to specify and modify SCRIPT/VS head levels (that is, chapter and topic headings), and how SCRIPT/VS creates a table of contents from the head levels.
- "Chapter 7. Indexing" on page 79: A description of how to create an index by placing index entry information in the text of a document.
- "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87: A description of additional formatting features of SCRIPT/VS, including character translation, keeping text together, marking revised material, and drawing boxes. This chapter also describes footnotes, and conditional column and page ejects.
- "Chapter 9. The SCRIPT/VS Formatting Environment" on page 109: A description of the SCRIPT/VS formatting environment.
- "Chapter 10. Conditional Processing" on page 111: A description of how to alter the order in which input lines are processed. The techniques discussed include conditional control words, branching, and conditional sections.
- "Chapter 11. Combining SCRIPT/VS Files" on page 119: A description of how to combine SCRIPT/VS input files, merge input from several files, and use SCRIPT/VS to interactively create an output document.

- "Chapter 12. Symbols in Your Document" on page 129: A description of the SCRIPT/VS symbol processing capability: how to name symbols, store them in a symbol library, use system symbols, and use symbol arrays. This chapter describes many useful applications for symbols.

- "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147: A description of the SCRIPT/VS macro processing capability: how to build a macro, use symbols within a macro, conditionally process parts of the macro, and store macros in a macro library.

- "Chapter 14. GML Support in SCRIPT/VS" on page 159: A description of how to name a GML tag, build an application processing function (APF) associated with the tag, and map the tag to the APF. This section should be read in conjunction with the Document Composition Facility Generalized Markup Language: Starter Set Reference.

- "Chapter 15. Using SCRIPT/VS with Other Programs" on page 167: A description of how to use SCRIPT/VS with other text processing programs, either as a postprocessor used to format the output of another program, or as a preprocessor used to prepare SCRIPT/VS files for input to another text processing program.

- "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171: A description of the SCRIPT/VS dictionary and how SCRIPT/VS uses it for automatic hyphenation and spelling verification. This chapter also describes how to build an addenda dictionary, used to supplement the SCRIPT/VS dictionary.

- "Chapter 17. Producing Input for STAIRS/VS" on page 179: A description of how to use SCRIPT/VS to prepare input for STAIRS/VS.

- "Chapter 18. Diagnostic Aids" on page 181: A description of how to identify errors in your input file and correct them. This chapter includes a description of the SCRIPT/VS input substitution trace facility, which enables you to observe the results of SCRIPT/VS processing at various points as your input file is being processed.

- "Chapter 19. EasySCRIPT" on page 189: A description of EasySCRIPT functions and usage.

- "Chapter 20. Compatibility with Earlier Releases of SCRIPT" on page 195: A description of the similarities and differences between SCRIPT/VS Release 2, SCRIPT/VS Release 1, and SCRIPT/370.

- "Chapter 21. ATMS Conversion" on page 211: A description of the ATMS Conversion program provided with SCRIPT/VS for use with the Document Library Facility.

- "Chapter 22. Compatibility with TSO/FORMAT" on page 217: A description of the similarities and differences between SCRIPT/VS and TSO/FORMAT.

- "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219: A detailed description of each SCRIPT/VS control word: its format, parameters, usage notes, and examples of use.

- "Appendix A. SCRIPT/VS Summary" on page 341: A summary of SCRIPT/VS: file names, SCRIPT command parameters, control words, system symbols, special characters, character sets, and 3800 Printer fonts.

- "Appendix B. Device and Font Table Maintenance" on page 367: A description of how to define a new logical output device or a new font to SCRIPT/VS.

- "Appendix C. Stem Processing" on page 373: A list of the word prefixes and suffixes recognized in each language for spelling verification.

- "Appendix D. Fonts Supplied with SCRIPT/VS" on page 377: An illustrated list of the fonts provided with SCRIPT/VS for use with the IBM 3800 Printing Subsystem.

- "Appendix E. Formatting Considerations for the 3800 Printer" on page 385: A description of the use of SCRIPT/VS with the IBM 3800 Printing Subsystem.

- "Appendix F. Maintaining User Dictionaries" on page 391: A description of the dictionary maintenance program, which is used to create and modify user dictionaries for spelling checking and hyphenation.

- "Appendix G. Performance Considerations" on page 395: A discussion of performance considerations.

RELATED PUBLICATIONS

- Document Composition Facility and Document Library Facility General Information Manual, GH20-9158. This manual describes the SCRIPT/VS and Library program products and summarizes their functions and capabilities. It also summarizes the operating environment requirements for these products.

- Document Composition Facility Introduction to the Generalized Markup Language: Using the Starter Set, SH20-9186. This manual provides an introduction to GML and a primer on document markup with the GML starter set provided with SCRIPT/VS.

- Document Composition Facility Generalized Markup Language: Starter Set Reference, SH20-9187. This manual describes the GML starter set provided with SCRIPT/VS.

- Document Composition Facility Generalized Markup Language: Concepts and Design Guide, SH20-9188. This manual discusses GML concepts and provides guidelines for designing your own GML.

- Document Library Facility Guide, SH20-9165. This manual explains how to set up, use, and maintain the library. It also explains how to call SCRIPT/VS as a subroutine, and how to convert ATMS documents into SCRIPT/VS input files.

- Document Composition Facility: Diagnosis, LY20-8067. This manual is licensed; that is, it remains the property of IBM and is provided under the terms of the licensing agreement for the Document Composition Facility. It is for IBM service personnel and customers who diagnose programming errors.

- Document Composition Facility: User's Quick Reference, SX26-3723. This reference card summarizes the SCRIPT command, the SCRIPT/VS language, and other facilities of SCRIPT/VS.

- Document Composition Facility: GML Quick Reference, SX26-3719. This reference card summarizes the GML starter set and how to use SCRIPT/VS in each interactive environment.

- Virtual Machine Facility/370: Introduction, GC20-1800. This manual contains an introduction to CMS (the Conversational Monitor System), which is one of the interactive systems in which SCRIPT/VS operates. Other manuals that include detailed information about CMS are:

  - VM/370: CP Command Reference for General Users, GC20-1820

  - VM/370: CMS User's Guide, GC20-1819

  - VM/370: CMS Command and Macro Reference, GC20-1818

- VM/370: Terminal User's Guide, GC20-1810

- OS/VS2 TSO Terminal User's Guide, GC28-0645. This manual gives detailed user information about OS/VS2 TSO (Time Sharing Option), which is one of the interactive systems in which SCRIPT/VS operates. It describes the TSO EDIT command and related facilities for text entry and editing and for text data set management. Other manuals that include detailed information about TSO are:

  - OS/VS2 TSO Command Language Reference, GC28-0646
  - OS/VS2 TSO Command Language Reference Summary, GX28-0647

- Advanced Text Management System-III (ATMS-III): General Information Manual, GH20-2404. This manual contains an introduction to ATMS (the Advanced Text Management System), which is one of the interactive systems in which SCRIPT/VS operates. Other manuals that include detailed information about ATMS are:

  - ATMS-III: Program Reference Manual, SH20-2424
  - ATMS-III: Terminal Operator's Guide, SH20-2425
  - ATMS-III: Terminal Operator's Exercise/Reference Guide, SH20-2426
  - ATMS-III: Operations Guide, SH20-2427

- Introducing the IBM 3800 Printing Subsystem and Its Programming, GC26-3829. This manual provides general information about the 3800 Printer. It describes what the 3800 Printer is and provides information about the standard and optional features available for the 3800 Printer. The IBM 3800 Printing Subsystem Programmer's Guide, GC26-3846 (for OS/VS1 and OS/VS2 MVS) and GC26-3900 (for DOS/VSE), includes detailed information about programming for the 3800 Printer.

RELEASE 2

This revision documents the functional changes that have been made to the Document Composition Facility for Release 2. The major changes and additions are:

- Descriptions of the new CTF, INDEX, TLIB, and @user-option command options have been added to "Chapter 2. Using the SCRIPT Command" on page 13.
- Chapters describing the new indexing, GML, and STAIRS/VS support have been added.
- Descriptions of SCRIPT/VS's formatting capabilities have been added or changed to reflect the new and changed SCRIPT/VS control words.
- Descriptions of the syntax of the following new control words have been added to "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219:

  .AA [Associate APF]
  .AN [And]
  .CT [Continued Text]
  .DF [Define Font]
  .DL [Dictionary List]
  .EL [Else]
  .FL [Float]
  .GS [GML Services]
  .IE [Index Entry]
  .IX [Index]
  .ME [Macro Exit]
  .OR [Or]
  .PI [Put Index]
  .PM [Page Margins]
  .RN [Reference Numbers]
  .SO [STAIRS/VS Output]
  .TH [Then]
  .TS [Translate String]
  .TU [Translate Uppercase]
  .UW [Unverified Word]
  .WZ [Widow Zone]

- A summary of the formatting differences between Release 1 and Release 2 control words has been added to "Chapter 20. Compatibility with Earlier Releases of SCRIPT" on page 195.
- Clarifications have been made to:
  - Using the SCRIPT Command
  - GML Support in SCRIPT/VS
  - Automatic Hyphenation and Spelling Verification
  - Diagnostic Aids
  - SCRIPT/VS Control Word Descriptions

All of these additions and changes have been marked with revision bars in the left margin.

<u>CHAPTER 1. AN INTRODUCTION TO SCRIPT/VS</u>

SCRIPT/VS is a text processing program that executes in:

- An interactive environment under:
  - The Conversational Monitor System (CMS), of the IBM Virtual Machine Facility/370 (VM/370)
  - The Time Sharing Option (TSO) of OS/VS2 MVS
  - The Advanced Text Management System-III (ATMS-III) in a Customer Information Control System/Virtual Storage (CICS/VS) environment

  Use of SCRIPT/VS in the interactive environments requires the Foreground Environment Feature of the Document Composition Facility.

- A batch processing environment under:
  - OS/VS2 MVS
  - OS/VS1
  - DOS/VSE

  Use of SCRIPT/VS in the background batch environments requires the Document Library Facility.

SCRIPT/VS formats text for printing on terminals, impact printers, or nonimpact printers. SCRIPT/VS provides flexible composition for printing on a computer printer as an alternative to independent typesetting machines or sending typesetting jobs to an outside vendor.

SCRIPT/VS can also be used to prepare documents for processing by other programs, such as formatters that support photocomposers, and to prepare data for use as input to the Storage and Information Retrieval System/Virtual Storage (STAIRS/VS) program product.

When you use SCRIPT/VS with CMS, you need to be able to do the following:

- Log on and enter CMS commands
- Create and edit files using a CMS editor
- Manage CMS disk storage

For more information about VM/370, see:

<u>VM/370: Terminal User's Guide</u>

<u>VM/370: CMS User's Guide</u>

When you use SCRIPT/VS with TSO, you need to be able to do the following:

- Log on and enter TSO commands
- Create and edit files using a TSO editor
- Manage TSO disk storage

For more information about TSO, see:

<u>OS/VS2 TSO Terminal User's Guide</u>

<u>OS/VS2 TSO Command Language Reference</u>

When you use SCRIPT/VS with ATMS-III, you need to be able to do the following:

- Log on and enter ATMS-III commands
- Create and edit files using the ATMS-III editor
- Manage ATMS-III disk storage

For more information about ATMS-III, see:

> <u>ATMS-III Terminal Operator's Guide</u>
>
> <u>ATMS-III Terminal Operator's Exercise/Reference Guide</u>
>
> <u>ATMS-III Program Reference Manual</u>

When using SCRIPT/VS in a batch environment, input can come from:

- Files created by the ATMS-III, CMS, TSO, or VSPC editors
- A word processing system attached to the host system via a telecommunications network
- A user-written program that calls SCRIPT/VS as a subroutine

SCRIPT/VS reads input data containing text and control information, formats the data into pages, and produces formatted output on a system printer or other suitable output device. Information that may appear in the SCRIPT/VS input file includes:

- "text," which is the content of the document.
- "symbols," which are character strings starting with an ampersand (&) that are resolved to a different character string when the line is processed. The new string may be text, another symbol, or control information. For example, in this document the symbol "&3800" resolves to "3800 Printer."
- "control words," which are two-letter codes that are recognized when the first character in the input line is a period (.). For example, to cause a page eject ".PA" is specified in column one of an input line.
- "macros," which are groups of control words and symbol substitutions. (Macros are often used to accomplish functions not provided by a single control word or to change how a control word is processed.) A macro is defined using the .DM [Define Macro] control word. For example, you can define a ".TOP" macro such that it contains a ".PA" control word followed by a ".CE" control word. Then, anytime the ".TOP" macro is encountered, the ".PA" control word is executed to begin a new page, and the ".CE" control word is executed to center the next line of text.
- "GML markup,[1]" which uses "tags" to identify the associated text as a particular part of a document, such as paragraph or heading. GML (Generalized Markup Language) provides a syntax and usage rules for marking up a document, and allows you to develop a vocabulary of tags for describing your documents. A tag is identified by the GML delimiter, which is by default the colon (:), anywhere in an input line. For example, in the GML starter set provided with SCRIPT/VS, the tag :p identifies a paragraph.

Normally, a SCRIPT/VS input file is a sequential file on direct access storage that can be modified using an editing program. SCRIPT/VS can process the file and produce formatted output that reflects changes to the text or markup.

---

[1] To "mark up" a source document is to add information to it that tells SCRIPT/VS how to process it.

SCRIPT/VS knows the width and depth of the output page. It fills up a page with text, then begins printing a new page automatically. It continues processing until it reaches the end of the input data.

Many text processing programs can do these things. SCRIPT/VS offers additional flexibility in the following forms:

- SCRIPT/VS data files are independently maintained. Any editor that can produce files in a format acceptable to SCRIPT/VS may be used to create or modify these files.

- SCRIPT/VS can combine many input files to produce a single, integrated output document. The imbedded files can be arranged in any sequence. While they are being processed, SCRIPT/VS treats each input file as though it were part of a single continuous input file.

- SCRIPT/VS has high-level macro and symbol capabilities. With SCRIPT/VS you can define your own control words or GML tags, conditionally process text, perform variable symbol substitutions, and do integer arithmetic.

- New SCRIPT/VS users can become productive quickly, because the control words and GML tags are easy to use.

Generalized Markup Language

Generalized Markup Language (GML) provides the syntax and usage rules for developing your own vocabulary of "tags" for describing the parts, or "elements," of a document without respect to particular processing. With GML tags you can describe the type of element; you can also enter "attributes" to describe other characteristics of an element.

The following example of GML markup describes a figure "element," whose type is FIG (figure), that causes the figure element to be enclosed on all four sides.

```
:fig frame='box'.
```

Since GML markup does not specify processing, it must be interpreted before any processing can occur. In GML, "interpreting" markup means performing the correct application processing function (APF) on the element the markup describes. In SCRIPT/VS, APFs are implemented as sets of control words in the form of macro definitions. The association, or "mapping," between the GML markup and the APFs is usually made in a document called a "profile," which is processed by SCRIPT/VS before the file marked up in GML is processed.

Information on GML markup is contained in the Document Composition Facility Generalized Markup Language: Starter Set Reference. This manual explains the control words which invoke the actual processing, and the symbol and macro facilities that enable you to create APFs and profiles.

SCRIPT/VS RELEASE 2 HIGHLIGHTS

The Document Composition Facility Release 2 enhancements enable you to:

- Operate SCRIPT/VS under ATMS-III in a CICS/VS environment.

- Perform hyphenation and spelling verification in American English, United Kingdom English, Canadian English, French, Canadian French, German, Italian, Dutch, and Spanish.

- Generate an index from index entries specified within the text at points of reference. (The page numbers for these index entries are automatically generated.)

- Format data in Condensed Text Format (CTF) so it can be used as input for the STAIRS/VS program product.

Release 2 also extends the Generalized Markup Language (GML) capabilities to allow multiline markup and attribute manipulation, and enables you to create tags and APFs more easily.

HOW SCRIPT/VS WORKS

Control Words and Their Parameters

A SCRIPT/VS control word is identified by a period in column one of the input line, except when the .LI (Literal) control word specifies that a period in column one should be regarded as text. A ".*" at the start of an input line identifies a comment line. Comment lines do not appear in the output.

Each input line is scanned from left to right for a control word separator, usually a semicolon (;). If a control word separator is found, the character string to its left is processed; the character string to its right is saved. (The character string can contain control words and text.) This process is repeated until the entire input line is completely scanned. For example,

```
.sk .33i;.in .5i for 3;This is a line...
```

will cause approximately one-third of an inch of vertical space to be skipped before the next input lines are printed (.sk .33i). It will also cause the next three output lines to be indented one-half inch (.in .5i for 3).

Control words may have numeric or keyword parameters that further qualify the action to be performed. For example, the .CE [Center] control word accepts the keywords "ON" and "OFF" and is specified as follows:

```
.ce on
```

The .SP [Space] control word accepts numeric parameters and is specified as follows:

```
.sp 2i
```

Some control words that accept keyword parameters also accept numeric parameters. The .CE [Center] control word also allows you to specify a number of input lines to center. For example,

```
.ce 10
```

Each control word description lists the parameters that it accepts. See "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219 for a detailed description of each SCRIPT/VS control word.

Defaults and Initial Settings

SCRIPT/VS can format an input file without any control words or GML tags specified. In this case, the initial settings for page dimensions and formatting controls are used. The initial settings are associated with the logical output device specified with the DEVICE option of the SCRIPT command.

Each control word description includes initial setting and default values.

SCRIPT/VS INPUT FILE CHARACTERISTICS

SCRIPT/VS input files have the following default characteristics:

- In a CMS environment, input files usually
  - have a filetype of SCRIPT.
  - are composed of up to 65,535 fixed- or variable-length records, with a maximum of 132 bytes.
  - include uppercase and lowercase letters, numbers, and special characters.
  - do not contain line numbers. If the lines are numbered in positions 1 to 8 of each record, these numbers will be ignored.

  Normally, any CMS editor will create files of appropriate format for filetype SCRIPT.
- In a TSO environment, input files usually
  - have a data set organization of PO or PS.
  - are composed of fixed- or variable-length records, blocked or unblocked, with a maximum of 132 bytes.
  - include uppercase and lowercase letters, numbers, and special characters.
  - contain records with or without line numbers. If the input lines are numbered:
    - A variable-length record has the line number in positions 1 to 8 of each record.
    - A fixed-length record has the line number in the last eight positions.

  Normally, any TSO editor will create files of appropriate format.
- In an ATMS-III environment, input files
  - are located in ATMS-III working or permanent storage
  - are composed of variable-length records, with a maximum of 230 text characters
  - include uppercase and lowercase letters, numbers, and special characters
  - include ATMS-III page and unit numbers
- In a background environment, the input file is stored as a document in the Document Library Facility library. For more information, see the Document Library Facility Guide.

LOGICAL AND PHYSICAL OUTPUT DEVICES

When SCRIPT/VS formats a document it takes into consideration the characteristics of the intended physical output device, called the "logical output device," which may be a terminal, a line printer, or a nonimpact page printer. The actual destination of the formatted output may be any of the devices supported by SCRIPT/VS.

If you specify, via the DEVICE option, a specific logical device, SCRIPT/VS will assume an appropriate output destination. Conversely, if you specify a specific output destination, SCRIPT/VS will assume an appropriate logical device.

You may specify any combination of output destination and logical device. For example, when formatting documents that are to be saved for printing at a later date, specify the destination "FILE" and the logical output device of your choice.

VERTICAL AND HORIZONTAL SPACE UNITS

Many SCRIPT/VS control words accept parameters that specify vertical or horizontal dimensions or distances. As Figure 1 illustrates, these dimensions may be expressed in any of several different space units:

Centimeter	The hundredth part of a meter. There are 2.54 centimeters in one inch.
Cicero	A standard measurement in the Didot Point System, used in most countries except Great Britain and the United States. The Cicero is 0.1776 inches, and there are twelve Didot points in one Cicero.
Em	The width of the "m" space in the font you are using. This width may be 1/10 inch (2.54 millimeters) 1/12 inch (2.117 millimeters) 1/15 inch (1.693 millimeters)
Inch	One-twelth of a foot. There are 39.37 inches in a meter.
Millimeter	One-thousandth of a meter. There are 25.4 millimeters in an inch.
Pica	A standard printer's measurement in Great Britain and the United States. A pica is 0.1663 inches. There are approximately twelve points in a pica.

Numbers without space unit identifiers are assumed to be in character spaces horizontally and line spaces vertically.

Unit Name	Specified As	Examples
Centimeter	aCM	1.27cm, 25cm
Cicero	nCp	c12 = 12 Didot points 2c3 = 2 ciceros and 3 points
Em	nM	6m, 11m
Inch	aI	3.5i, .75i
Millimeter	aMM	12.7mm, 100mm
Pica	nPp	p6 = 6 points 3p2 = 3 picas and 2 points

Where:

a	is a number of inches, millimeters, or centimeters. The number may be fractional, with up to two decimal positions.
n	is a number of whole ems, ciceros, or picas.
p	is a number of points. (There are twelve points in a cicero or pica, and approximately 72 points in an inch.)

Figure 1. Space Units Notation: All vertical and horizontal dimensions specified with SCRIPT/VS control words and options may be given in any of the forms shown here.

Note: It is not always possible to satisfy space requests exactly on all devices. In this case, the nearest available amount is used.

FONTS

In SCRIPT/VS, a "font" is a set of characters having the same size and type style.

When formatting for the 3800 Printer, text can be printed in several different character sets, as specified in the CHARS option of the SCRIPT command. (The CHARS option is discussed in "Chapter 2. Using the SCRIPT Command" on page 13.)

3800 character sets may be either "monopitch" fonts, wherein all characters have the same width, or "proportional" fonts, wherein characters have different widths.[2]

The "pitch" of a font is the number of "characters per inch" in a line of printed text. The 3800 Printer has three pitch values:

10-Pitch (10 characters per inch)
12-Pitch (12 characters per inch)
15-Pitch (15 characters per inch)

SCRIPT/VS extends the concept of fonts to output devices other than the 3800 Printer. Capitalization and underscoring can be treated as different fonts on all devices, and overstriking allows you to print boldface text on impact printers and typewriter terminals. (See "Defining Internal Fonts" on page 107.)

CALLING THE SCRIPT/VS PROCESSOR

Interactive Environments

You call the SCRIPT/VS processor by issuing the SCRIPT command and specifying the name of the file SCRIPT/VS is to process. To issue this command in one of the three interactive environments that support SCRIPT/VS, use one of the following formats:

- In CMS: SCRIPT filename (options
- In TSO: SCRIPT dsname options
- In ATMS-III: script docname (options

The SCRIPT command format and options are described in "Chapter 2. Using the SCRIPT Command" on page 13.

Batch Environments

For details about calling SCRIPT/VS in a batch environment, see Document Library Facility Guide.

Using SCRIPT/VS as a Subroutine

In a batch environment, with the Document Library Facility program product, an application program can call SCRIPT/VS as a subroutine. For details, see Document Library Facility Guide.

---

[2] For example, the character "I" may be narrower than the character "H," and the "M" and the "W" may be wider than the "N."

Using SCRIPT/VS as a Preprocessor

SCRIPT/VS can be used to prepare an input file for use as input to another text processing program.

Using SCRIPT/VS to Process Input for STAIRS/VS

You can cause SCRIPT/VS to format text in a Condensed Text Format which is acceptable as input to the STAIRS/VS program product. For details, see "Chapter 17. Producing Input for STAIRS/VS" on page 179.

FORMATTING CONSIDERATIONS

When you create an input file, or when you create application processing functions (APFs) for GML processing, you should consider:

- How is the text formatted? Do you want to add spaces between lines or paragraphs? Indent lines? Create numbered or bulleted lists?
- What size paper are you using for output? How many lines of text should be on the page? How wide is it? Do you want special titles on the top or bottom of each page? Where, and in what format, do you want the page number to appear?
- Are you going to use a multiple column page layout?
- Do you want to generate a table of contents listing major headings, and the page numbers on which they occur?
- Do you want to generate an index?
- How long is the final document going to be? Can you organize it into several input files and let SCRIPT/VS combine them?
- Are you going to have illustrations? Are you going to create tables and boxes using SCRIPT/VS? Do you need to leave blank pages or blank space so that artwork can be included later? How are you going to number the illustrations?
- Are you using variable information? Can you use symbolic names throughout a document to represent information that changes frequently?
- Do you want the SCRIPT/VS processing to be interactive? Are there types of information you may want to enter during SCRIPT/VS processing?
- Are you using the same sequences of control words frequently? Can you define a macro so you don't have to rekey all the control words in sequence each time?

Selecting Control Words

This book describes many formatting techniques and shows many examples. No single example or technique is necessarily the best; there are usually several ways to do the same thing. As you become more experienced in using SCRIPT/VS, standard ways of doing things will evolve and may be accepted as installation standards where you work.

SCRIPT/VS FUNCTIONS

User-controlled SCRIPT/VS processing includes two general categories of functions: formatting functions, and general document handling.

Formatting Functions

Page Layout

You can control page dimensions, the number of columns per page, running headings and footings, and line spacing.

Page layout includes:

- Line Formatting. You can control concatenation, justification, centering, and left or right alignment. For details, see "SCRIPT/VS Text Formatting" on page 33.
- Line Spacing. You can control the amount of space left between output lines, including the reservation of space for drop-in art. For details, see "Vertical Space" on page 43.
- Paragraphing. You can control the style of paragraphing, spacing between paragraphs, and indention. For details, see "Breaks" on page 38.
- Fonts. You can control which font is used for different portions of text, both in the body and in running headings and footings. For details, see "Using Fonts with the IBM 3800 Printing Subsystem" on page 48.
- Columns. You can define the number of columns as well as the size of each and its placement on a page. For details, see "Chapter 5. Multicolumn Page Layout" on page 67.
- Margins. You can control the size of the top and bottom margins as well as the left and right margins. Title lines can be defined that will be put into the top or bottom margins. For details, see "Basic Page Dimensions" on page 55.
- Indention. You can control indention in a number of ways. For example, you can create hanging indents and left or right margin indention, and can control the vertical duration and extent of all indention. For details, see "Changing the Margins" on page 39.
- Headings and Footings. You can create running headings and footings with page numbers for all pages or different ones for odd and even pages. For details, see "Running Headings and Footings" on page 58.
- Reference Numbers. You can have line reference numbers placed on any page of a document. If reference numbers are requested, all nonblank lines in the body of the page will be serially numbered. For details, see "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87.

Head Levels

You can specify up to seven head levels for distinctive formatting of headings for different levels of topics. Distinctive formatting includes before and after spacing, font selection, overstriking, capitalization, underscoring, and right or left alignment. For details, see "Chapter 6. Head Levels and Table of Contents" on page 73.

Table of Contents

You can control whether or not a table of contents is automatically generated and where it is placed. SCRIPT/VS collects entries for a table of contents from the text of head levels and supplies the page number. You can also specify phrases other than the text of head levels to appear in the table of contents. For details, see "Chapter 6. Head Levels and Table of Contents" on page 73.

Indexing

You can include index entries in the body of your document at their points of reference. SCRIPT/VS will use these index entries to generate an index for your document that includes appropriate page numbers for all of the entries. For details, see "Chapter 7. Indexing" on page 79.

Highlighted Phrases

You can highlight phrases for emphasis. Font selection, overstriking, capitalization, and underscoring can be used to emphasize important phrases. For devices that support multiple fonts, you can change font for emphasis. For details, see "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87.

Footnotes

SCRIPT/VS saves text indicated as a footnote and places it at the bottom of the page.[3] Subsequent footnotes are placed below it. For details, see "Footnotes" on page 96.

Revision Codes

You can control the placement of up to nine distinct revision codes in the left margin to flag a line of particular interest, such as text that has been revised since a previous version of the document. For details, see "Marking Updated Material" on page 99.

Hyphenation and Spelling Verification

You can control whether words are hyphenated at the end of output lines and whether they are checked for correct spelling. SCRIPT/VS provides dictionaries of many common root words in nine languages. Algorithms for prefix and suffix variations, provided with each language, extend the basic root words. SCRIPT/VS determines hyphenation points and spelling validity based on these algorithms, and the basic root words. You can add words to addenda or user-created dictionaries as required for a particular document or use an algorithmic hyphenator[4] to further extend SCRIPT/VS's hyphenation and spelling verification capabilities. For details, see "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.

Printing Part of the Output Document

You can control whether every page of formatted text is put in the output document or only the range or ranges of pages specified. For details, see the PAGE option in "PAGE: Selectively Print Pages" on page 26.

Tabs

You can specify the values of tabs. When formatting output lines, SCRIPT/VS tabs to the right to the prescribed tab stop. For details, see "Using Tabs In SCRIPT/VS" on page 36.

---

[3] Like this.

[4] An algorithmic hyphenator for American English is provided with SCRIPT/VS.

Boxes

You can construct boxes around formatted text. You can also draw boxes within boxes, vertical lines to separate columns of text, and horizontal lines to separate rows. For details, see "Drawing Boxes" on page 101.

Keeping Text Together

SCRIPT/VS processing includes functions that keep text together to improve the appearance of output. For example, SCRIPT/VS always keeps the text of a head level together with the first few lines of text after the heading, so that they appear in the same column. SCRIPT/VS can also ensure that single lines at the beginning or end of a paragraph (widows) are not placed by themselves at the bottom or top of a column or page. For details, see "Ensuring That Blocks of Text Stay Together" on page 92.

Placing Text at the Top or Bottom of a Page or Column

You can indicate that blocks of text, called floats, are to be kept together and placed at the top or bottom of a column or page. For details, see "Ensuring That Blocks of Text Stay Together" on page 92.

General Document Handling Functions

Documents Marked up for Earlier Releases of SCRIPT

If you have documents prepared for SCRIPT/VS Release 1 or SCRIPT/370 Version 3, you can use SCRIPT/VS Release 2 to format them, with very few changes, if any, required. For details, see "Chapter 20. Compatibility with Earlier Releases of SCRIPT" on page 195.

Processing Generalized Markup Language (GML) Tags

SCRIPT/VS recognizes Generalized Markup Language (GML) tags as a form of text markup, and provides extensive facilities for mapping GML tags to APFs, attribute manipulation, and symbol processing. For details, see "Chapter 14. GML Support in SCRIPT/VS" on page 159.

Saving Input Lines for Subsequent Processing

You can control whether certain input lines will be written to a data set or a file. For details, see "Chapter 11. Combining SCRIPT/VS Files" on page 119.

Imbedding Separate Files

You can control how separate source files are brought together for processing as a single document. Any number of source files can be imbedded in the primary source file. A source file that has been imbedded can itself imbed another source file. For details, see "Imbedding and Appending Files" on page 119.

Processing Symbols and Macros

You can define symbols and macros for substitution during processing. Symbols have many uses: for example, in tests for conditional processing, for cross-references to pages or figure numbers, for entering characters unavailable on the entry keyboard, and as abbreviations for repetitive phrases. You can define what a particular macro will do. For example, you might redefine a particular head level to alter the SCRIPT/VS formatting style. Symbol and macro instruction facilities are used to support the Generalized Markup Language. For details, see "Chapter 12. Symbols in Your Document" on page 129 and "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147.

Processing Input Conditionally

You can cause SCRIPT/VS to alter input processing. For example, by setting symbol values, and comparing those values, you can control whether a block of input text is included in the output document. SCRIPT/VS uses condition testing as part of its normal processing. It checks the amount of space left in a column before processing certain blocks of text. Conditional processing can be controlled by defining macro instructions to supplement SCRIPT/VS control words. For details, see "Chapter 10. Conditional Processing" on page 111.

Specifying the Destination of Output

You can control the output destination of the formatted document. It can be stored as a file for later use, or printed on a variety of devices, including impact and nonimpact printers, and display and typewriter terminals. For details, see "Chapter 2. Using the SCRIPT Command" on page 13.

Processing Interactively During Formatting

In an interactive environment (CMS or TSO), you can affect SCRIPT/VS as it processes by entering text or markup from a terminal. In effect, the terminal can be treated as an input file. For example, you can interactively specify the values of symbolic variables specified in the document or enter those portions of text that vary from one processing time to the next. If you are using a typewriter terminal, you can also stop SCRIPT/VS output processing at any point on a line to change typing elements or enter text. For details, see "Interactive SCRIPT/VS Processing" on page 126.

Converting ATMS Documents

If the IBM Document Library Facility program product is installed with SCRIPT/VS, you can convert most ATMS-II and ATMS-III markup to similar or equivalent SCRIPT/VS markup. For details, see Document Library Facility Guide and "Chapter 21. ATMS Conversion" on page 211.

Debugging by Tracing Processing Actions

You can trace all control words and each step of symbol and macro substitution in input lines. In cases where unexpected results are observed, trace information can be an invaluable aid in pinpointing the problem area. For details, see "Chapter 18. Diagnostic Aids" on page 181.
Introduction

CHAPTER 2. USING THE SCRIPT COMMAND

Issue the SCRIPT command to process and format an input file. SCRIPT/VS formats the input file based on GML tags, macros, control words, and text that are included in the file, as well as the SCRIPT command options you specify.

SCRIPT can be issued as a CMS command, a TSO command, or an ATMS-III command. For details about using the SCRIPT command in a batch environment, see the Document Library Facility Guide. The format of the SCRIPT command is the same for each system, except that in TSO options must not be placed in parentheses:

In CMS,

```
SCRIPT   ┌                              ┐
         │ file-id [ ( options... ]     │
         │ ?                            │
         └                              ┘
```

In TSO,

```
SCRIPT   ┌                              ┐
         │ file-id [ options... ]       │
         │ ?                            │
         └                              ┘
```

In ATMS-III,

```
script   ┌                              ┐
         │ file-id [ [(] options...     │
         │ ?                            │
         └                              ┘
```

<u>where</u>:

? causes SCRIPT/VS to display a list of all the valid command options.

file-id is the name of the input file. When the input file contains imbedded or appended files, <u>file-id</u> names the primary or master file; the imbedded and appended files are named with control words in the master file. The format of the file-id depends on the environment from which SCRIPT/VS is called.

options specify how SCRIPT/VS is to process and format the input file and where the resulting output file is to go. You can specify as many options as you think appropriate. A detailed description of each option follows. The left parenthesis "(" before the option list is required in the CMS environment, is optional in the ATMS-III environment, and is not permitted in the TSO environment.

NAMING THE INPUT FILE

The format of the name you specify for <u>file-id</u> depends on the environment from which you call SCRIPT/VS. Except for TSO, the naming rules and conventions apply equally to the primary input file, the profile, and any imbedded or appended files.

CMS Naming Conventions

The file-id of a CMS file to be processed is given in the form:

```
filename [filetype [filemode] ]
```

If filetype is omitted, a filetype of "SCRIPT" is assumed. If filemode is omitted, the CMS search sequence is used to locate the file on an accessed CMS disk. If you want to specify the filemode, you must also give the filetype, since these parameters are positional.

TSO Naming Conventions

In TSO, you can use a fully or partially qualified data set name to refer to the primary input file or profile in the SCRIPT command. If the file-id given is not fully qualified (enclosed in single quotation marks), the userid is prefixed to the file-id as the leftmost qualifier, and "TEXT" is added (unless it already appears) as the right-most qualifier. For example,

Specified DSNAME	Actual DSNAME
A	userid.A.TEXT
A.TEXT	userid.A.TEXT
DOC(CHAP1)	userid.DOC.TEXT(CHAP1)
'DPJK1.X.Y'	DPJK1.X.Y
(CHAP2)	userid.TEXT(CHAP2)

Imbedded and appended files are qualified as follows:

- If the name of a partitioned data set is specified with the SEARCH option,

  ```
  'userid.searchname.TEXT'
  ```

  the library will be searched for a member of the given name.
- If a TEXTLIB has been allocated for the TSO session, the allocated libraries are searched for a member of the given name.
- If the SEARCH option is not specified, no TEXTLIB exists, and the primary input file is a partitioned data set, that library is searched for a member of the given name.
- If the SEARCH option is not specified, no TEXTLIB exists, and the primary input file is not a partitioned data set, the library 'userid.TEXT' is searched for a member of the given name.

ATMS-III Naming Conventions

Documents in an operator's working storage may be formatted with the command

```
script *
```

Documents that are to be formatted from permanent storage or imbedded or appended may be specified in a fully qualified way, such as:

```
'docname:opnum;getw'
```

This will result in a search for the document named "docname" with a getword of "getw" belonging to the user whose operator number is "opnum". A qualified name will always result in an explicit search without subdocument index search. A name can be qualified by the use of only the colon character (:) without any "opnum". This form of qualification signifies that the document is to be explicitly located and read from the operator's permanent storage.

Option	non-TSO Environments	TSO Environment
BIND	B	B
CHARS	C	CH
CONTINUE	CO	CO
CTF	CT	CT
DEST	DE	DES
DEVICE	D	DEV
DUMP	DU	DU
FILE	F	F
INDEX	I	I
LIB	L	L
MESSAGE	M	M
NOPROF	N	NOP
NOSPIE	NOS	NOS
NOWAIT	NOW	NOW
NUMBER	NU	NU
OPTIONS	O	O
PAGE	P	PA
PRINT	PR	PRI
PROFILE	PRO	PRO
QUIET	Q	Q
SEARCH	S	SE
SPELLCHK	SP	SP
STOP	ST	ST
SYSVAR	SY	SY
TERM	T	TE
TLIB	TL	TL
TWOPASS	TW	TW
UNFORMAT	U	UN
UPCASE	UP	UP

Figure 2. Minimum Abbreviations of SCRIPT Options: See Figure 3 on page 16 for those options that are valid for your environment.

If a getword is specified, it must match the document getword even though the document belongs to the requesting user. If a getword is not specified for a document that does not belong to the requestor it must have a getword of "any".

Documents in an operator's permanent storage may also be formatted by transmitting a request to an appropriate SCRIPT/VS peripheral queue:

```
XFO;qname;docname:opnum;getw;options
```

where "qname" is the name of a SCRIPT/VS output queue and "options" are any valid SCRIPT/VS command options.

Note: ATMS-III will always add either PRINT or CTF to the user's options.

SCRIPT COMMAND OPTIONS

SCRIPT command options control how SCRIPT/VS processes and formats your input file. Some of the options have parameters; each option's parameters are enclosed in parentheses. You do not have to enter a right parenthesis unless another option follows. Options and parameters are separated from each other by blanks. In TSO, a comma may also be used as a separator.

The options you can specify with the SCRIPT command are shown in Figure 3.

The name of each option can be shortened to its minimum unambiguous length. In TSO, ambiguous truncations are not accepted: you will be prompted to reenter the option. In other systems, ambiguous truncations are accepted and interpreted as shown in Figure 2.

Option	Parameters	Description
BIND	(bind) (obind ebind)	Shift the page image to the right.
CHARS	(font1 ... font4)	Specify up to four fonts.
CONTINUE		Continue processing after a nonsevere error occurs.
CTF		Prepare output in STAIRS/VS Condensed Text Format.
DEST	(station-id)	Specify a remote output station. (Valid only in TSO.)
DEVICE	(devtype)	Specify a logical output device.
DUMP		Enables the .ZZ [Diagnostic] control word.
FILE	[(fileid)]	Specify a disk file for output. (Not valid in ATMS-III.)
INDEX		Enable the .PI [Put Index] control word.
LIB	(libname ...) (opnum ...)	Specify symbol and macro libraries. (Only one in TSO; up to eight in CMS and ATMS-III.)
MESSAGE	([DELAY] [ID] [TRACE])	Control message printing.
NOPROF		Suppress the profile.
NOSPIE		Prevent entering SPIE exit routines. (Valid only in CMS and TSO.)
NOWAIT		Prevent prompting for paper adjustment. (Valid only for typewriter terminals in CMS and TSO.)
NUMBER		Print file name and line number.
OPTIONS	[(fileid)]	Specify a file that contains SCRIPT options. (Valid only in CMS and ATMS-III.)
PAGE	[([PROMPT] [[FROM] p [TO] q] [[FROM] p FOR n] [[FROM] p ONLY])]	Selectively print pages. (PROMPT is valid only in CMS and TSO.)
PRINT	[(copies,class, fcb,ucs)]	Produce printer output. (Suboptions are valid only in TSO.)
PROFILE	[(fileid)]	Specify a profile. (A file to be imbedded before the primary input file is processed.)
QUIET		Suppress formatter identifier message.
SEARCH	(libname) (opnum ...)	Specify a library. (Not valid in CMS.)
SPELLCHK		Enable the .SV [Spelling Verification] control word.
STOP		Print separate pages at the terminal. (Valid only for typewriter terminals in CMS and TSO.)

Figure 3. Summary of SCRIPT Options (Part 1 of 2)

Option	Parameters	Description
SYSVAR	(n value ...)	Set symbol values for &SYSVARn.
TERM		Display the output at a user's terminal.
TLIB	(libname ...)	Specify spelling checking and hyphenation libraries. (Valid only in CMS.)
TWOPASS		Prepare with two formatting passes, and produce output on the second pass.
UNFORMAT		Print all input lines without formatting.
UPCASE		Fold lowercase letters to uppercase before printing.
@user-option	[(sub-options...)]	User-defined options, which must begin with the character "@". (Valid only in CMS.)

Figure 3. Summary of SCRIPT Options (Part 2 of 2)

Defaults

When you specify the SCRIPT command with a file-id and no options, the defaults are:

For CMS,

```
TERM BIND (2) PROFILE (PROFILE) LIB (GML2)
```

For TSO,

```
TERM BIND (2) PROFILE (PROFILE)
```

For ATMS-III,

```
TERM BIND (2) PROFILE (PROFILE) MESSAGE (DELAY)
```

For batch,

```
PRINT BIND (2) PROFILE (PROFILE) MESSAGE (DELAY)
```

All other options must be explicitly specified when desired.

When you specify the PAGE option without parameters, SCRIPT/VS assumes you mean PAGE (PROMPT). All other suboptions must be explicitly specified.

Mutually Exclusive Options

Some of the SCRIPT command options are mutually exclusive from a logical standpoint. However, when two such command options are specified, no error results; but one option can cancel the effect of another previously specified option. Within the groups of options listed below, the last one processed by SCRIPT/VS takes effect, except in TSO. Because of the way TSO parses parameters before passing them to SCRIPT/VS, options are processed in alphabetical order regardless of the order of entry. In other systems, they are processed in the order in which they are specified.

- PROFILE and NOPROF. These options specify a file that is to be imbedded before the primary input file is processed, or that no profile is needed.

- CTF, FILE, PRINT, and TERM. These options specify the actual destination of the formatted output. If a logical output device is not also specified, SCRIPT/VS selects one, based on the destination. Figure 4 on page 19 lists the default logical device for each destination.

Logical and Physical Output Devices

When SCRIPT/VS formats a document it takes into consideration the characteristics of the _intended_ physical output device (called the "logical device"). The _actual_ destination of the formatted output may be one of these devices or a file on disk. If you specify, with the DEVICE option, an explicit logical device, SCRIPT/VS will assume an appropriate output destination. Conversely, if you specify an explicit output destination, SCRIPT/VS will assume an appropriate logical device. However, you may specify explicitly any combination of output destination and logical device. For example,

```
SCRIPT A1 ( FILE DEVICE(3800N8)
```

will format a document for the IBM 3800 Printer but save the output in a disk file for later demand printing on a physical printer.

Note: There is an exception to the general rule that output formatted for any logical device may be directed to any destination: The CTF destination is valid only for the STAIRS logical device; the CTF option will be ignored if any other logical device is specified.

Figure 4 on page 19 shows the logical output device and output destination for a document when various combinations of options are specified.

Examples

- In CMS, format and print the document named TEST for an IBM 1403 printer. Print the last part of the document, starting with page 10, and allow for a binding margin on the left side of each page of one inch:

  ```
  SCRIPT TEST ( PRINT PAGE (10) BIND (1i)
  ```

- In TSO, format and display at the terminal the document named 'userid.RESUME.TEXT'. Do not prompt for paper adjustment; begin typing immediately. Do not type the formatter identification message:

  ```
  SCRIPT RESUME NOWAIT QUIET
  ```

- In ATMS-III, format the document currently in working storage to check for possible spelling errors:

  ```
  SCRIPT * ( SPELL
  ```

BIND: Shift the Page Image to the Right

The BIND option causes SCRIPT/VS to shift the formatted output of each page to the right on the paper. The BIND option is specified as:

```
BIND (obind ebind)
 or
BIND (bind)
```

You can specify a binding for odd-numbered pages (_obind_) and a different binding for even-numbered pages (_ebind_). If _ebind_ is not specified, the value of _bind_ applies to both odd- and even-numbered pages. The actual (or potential) page number of the output page can be controlled by the .PA [Page Eject] and .PN

Options Specified	Logical Device	Physical Device	Output Destination
none [CMS, TSO]	TERM	2741 or 3270	Terminal
none [Background]	1403W6	1403	Printer[1]
CTF	STAIRS	1403	([2])
FILE	1403W6	1403	file
PRINT	1403W6	1403	Printer[1]
TERM	TERM	2741 or 3270	Terminal
DEVICE(devtype)	devtype	device	device
DEVICE(STAIRS)	STAIRS	1403	([2])
CTF DEVICE(devtype)	devtype	device	device
FILE DEVICE(devtype)	devtype	device	file
PRINT DEVICE(devtype)	devtype	device	Printer[1]
TERM DEVICE(devtype)	devtype	device	Terminal
CTF DEVICE(STAIRS)	STAIRS	1403	([2])
FILE DEVICE(STAIRS)	STAIRS	1403	file
PRINT DEVICE(STAIRS)	STAIRS	1403	Printer[1]
TERM DEVICE(STAIRS)	STAIRS	1403	Terminal

[1] In the background environment, printed output is written to the destination identified by DSMLIST.

[2] The destination of CTF output is a file named DSMUTCTF, except in DOS/VSE and ATMS-III, where the destination is a file named DSMUCTF and a CICS/VS partitioned data set, respectively.

Figure 4. Logical Output Device vs. Output Destination: It is the user's responsibility to ensure that the characteristics of the physical device to which the output is directed match the characteristics of the specified or implied logical device. Your installation's conventions for output class, FCB, and forms must be included in these considerations. Note that if the CTF and DEVICE options are both specified, and the DEVICE option does not specify STAIRS, the CTF option will be ignored.

[Page Numbering Mode] control words, which are used to specify even and odd page numbers. Consequently, you can have two or more even-numbered (or odd-numbered) pages in a row.

Bindings can be specified in numbers of character spaces or in space units.

If the BIND option is not specified, it defaults to two character spaces. This allows room for potential revision codes for the first column, but decreases the potential maximum line length. Revision codes for subsequent columns are placed in the gutter between columns. If sufficient room is not provided for revision codes, they are discarded.

CHARS: Specify Fonts

The CHARS option identifies the fonts to be used.

The CHARS option is specified as:

```
CHARS (font1 [ ... font4 ] )
```

When you specify the CHARS option, you must specify at least one font.

All of the fonts specified with the .DF [Define Font] control word must be identified with the CHARS option. If you do not specify the CHARS option, the default font specified for the logical device is used. In either case, the first font specified or implied becomes the initial font.

When formatting for the IBM 3800 Printer, you can specify up to four uppercase-only fonts, or two upper- and lowercase fonts.[5] The document must then be printed on a system that supports the IBM 3800 Printer. The CHARS JCL parameter must specify the corresponding character arrangement tables in the same sequence as the fonts specified in the CHARS option of the SCRIPT command.

Refer to "PRINT: Produce Printer Output" on page 27 for details on printing documents formatted for the 3800.

CONTINUE: Continue Processing After a Nonsevere Error Occurs

The CONTINUE option allows processing to continue after SCRIPT/VS detects an error condition and flags it with an error message. When SCRIPT/VS encounters an error that is too severe for processing to continue, it terminates processing even when CONTINUE is specified. "Severe" and "terminal" errors cause SCRIPT/VS to terminate processing.

For a description of error types and SCRIPT/VS error messages, see the Document Composition Facility: Messages manual distributed with SCRIPT/VS.

CTF: Prepare Output in STAIRS/VS Condensed Text Format

The CTF option specifies that the document be prepared as input for the STAIRS/VS program product. SCRIPT/VS output is placed in CTF blocks[6] and written to an appropriate destination:

- In CMS, a file named DSMUTCTF SCRIPT A
- In TSO, a file named DSMUTCTF.TEXT
- In ATMS-III, a CICS extra partitioned data set
- In a background environment, the file identified by the DSMUTCTF DD statement (in OS/VS2 MVS and OS/VS1) or the DSMUCTF DLBL statement (in DOS/VSE)

If the device option is not specified, the STAIRS logical device is assumed; if any other logical device is specified, the CTF option is ignored.

DEST: Name a Remote Output Station or Queue

The DEST option, available only in TSO, is used to specify a remote output station at which the output document is to be printed.

The DEST option is specified as:

```
DEST (destination)
```

where destination is a one- to eight-character station-id.

The DEST option is ignored if the output destination is not a printer (as shown in Figure 4 on page 19).

---

[5] Fonts provided with SCRIPT/VS are illustrated in "Appendix D. Fonts Supplied with SCRIPT/VS" on page 377.

[6] The format of STAIRS/VS Condensed Text Format blocks is described in Figure 13 on page 180.

Logical Device Type	Real Device Type	Lines per Inch	Page Size (inches)		Line Length[1]	Page Length[2]
			Width	Depth	(bytes)	(lines)
TERM	([3])	6	8-1/2	11	60/132	66/144
2741	2741	6	8-1/2	11	60/132	66/144
3270	3270	6	8-1/2	11	60/132	66/144
1403N6	1403	6	8-1/2	11	60/85	66/144
1403N8	1403	8	8-1/2	11	60/85	88/192
1403W6	1403	6	13-1/2	11	60/132	66/144
1403W8	1403	8	13-1/2	11	60/132	88/192
1403W6S	1403	6	13-1/2	8-1/2	60/132	51/144
1403W8S	1403	8	13-1/2	8-1/2	60/132	68/192
1403SW [4]	1403	6	8-1/2	11	72/90	66/102
STAIRS	1403	6	13-1/2	11	60/132	66/144
3800N6	3800	6	8-1/2	11	60/85	60
3800N8	3800	8	8-1/2	11	60/85	80
3800N12	3800	12	8-1/2	11	60/85	120
3800W6	3800	6	13-1/2	11	60/136	60
3800W8	3800	8	13-1/2	11	60/136	80
3800W12	3800	12	13-1/2	11	60/136	120
3800N6S	3800	6	11	8-1/2	60/110	45
3800N8S	3800	8	11	8-1/2	60/110	60
3800N12S	3800	12	11	8-1/2	60/110	90
3800W6S	3800	6	13-1/2	8-1/2	60/136	45
3800W8S	3800	8	13-1/2	8-1/2	60/136	60
3800W12S	3800	12	13-1/2	8-1/2	60/136	90

[1] Line lengths are given as "default/maximum" in 10-pitch characters. For the IBM 3800 Printer, 12-pitch and 15-pitch fonts have values 20% and 50% greater, respectively. The potential maximum line length includes binding. The text fonts (as shown in Figure 39 on page 364) contain 10, 12, and 15 pitch blank characters. If these fonts are used, you should allow for an output record length, in bytes, that is 15 times the length (in inches) of your longest print line (including the binding).

[2] Default and maximum page lengths are identical for 3800 devices.

[3] The physical device type corresponding to the TERM logical device may be either 2741 or 3270, depending upon the actual terminal type.

[4] This is a 12-pitch device, as opposed to the normal 10-pitch 1403.

Figure 5. SCRIPT/VS Logical Device Characteristics

DEVICE: Specify a Logical Output Device

The DEVICE option allows you to identify the type of output device for which you want your document formatted. The logical device description includes the default page layout, font to be used, and characteristics of the physical output device.

The DEVICE option is specified as:

```
    DEVICE (devtype)
```

devtype is the name of a logical output device that takes into account the physical characteristics of the device as well as the characteristics that can be changed by the operator or by program control: font, lines per inch, form size, and page image size. SCRIPT/VS logical device support allows a single physical device type to be defined as many different logical device types, each having different characteristics. The logical devices defined in SCRIPT/VS are summarized in Figure 5.

You can add a new logical device to the SCRIPT/VS logical device table. For details about this procedure, see "Appendix B. Device and Font Table Maintenance" on page 367.

When you issue the SCRIPT command to format and display your document at the terminal, DEVICE (TERM) is assumed. When you invoke SCRIPT/VS in a batch environment or use the PRINT option in a foreground environment and do not specify a device type, SCRIPT/VS assumes DEVICE (1403W6).

The formatted output for all 3800 logical devices contains table reference characters (TRCs). Consequently, the parameter DCB=OPTCD=J must be included in the output JCL. For DOS/VSE, a //SETPRT control card must also be included.

Refer to "PRINT: Produce Printer Output" on page 27 for details on printing documents formatted for the 3800.

If you specify DEVICE(STAIRS) or CTF, the document will be formatted as it would for device 1403W6, but instead of preparing the output for a line printer, SCRIPT/VS will convert the output to STAIRS/VS Condensed Text Format (CTF)[6] records and write them to an appropriate file, as described under the CTF option.

If you specify DEVICE(STAIRS) _and_ PRINT, FILE, or TERM, the document will be formatted as it would for CTF output, but the lines and paragraph codes will be written to the specified destination for proofreading.

In ATMS-III peripheral queue operations, you can not override the DEVICE type defined by ATMS-III.

DUMP: Enable the .ZZ Control Word

The DUMP option allows SCRIPT/VS to perform a specific diagnostic action when it encounters a .ZZ [Diagnostic] control word in an input file. Parameters of the .ZZ control word specify the type of diagnostic action to be taken. The DUMP option is intended for use by the system programmer maintaining SCRIPT/VS.

If the DUMP option is not specified, SCRIPT/VS ignores the .ZZ control word.

For details about the output produced when the .ZZ [Diagnostic] control word is processed, see ".ZZ [Diagnostic]" on page 339.

FILE: Name a Disk File for Output

The FILE option directs the formatted output document to a direct-access file. If the DEVICE option is not also specified, the 1403W6 logical device will be assumed.

The FILE option is specified as:

```
    FILE [ (file-id) ]
```

file-id names the direct-access file. If you do not specify a _file-id_, SCRIPT/VS sends the output document to a default _file-id_ based on the environment.

- In CMS, the _file-id_ is of the form:

  ```
      filename [ filetype [filemode] ]
  ```

  The default _filename_ is "$filename", where "filename" is the first seven characters of the input filename, preceded by a dollar sign ($). The default _filetype_ is "SCRIPT", and the default _filemode_ is "A1".

If a file with the name specified or implied already exists, SCRIPT/VS issues a message to allow you to let the replacement of the old file occur or to cancel the output.

The file can contain a maximum of 65,533 records.

- In a TSO environment, file-id is a fully or partially qualified data set name. The full name will be determined by the following rules:

  1. If a fully qualified dsname (placed within quotation marks) is given, the name is used as specified.

  2. If a partially qualified name is provided, it is fully qualified by prefixing it with "userid." (or other prefix given with the TSO PREFIX command), and suffixing it with ".LIST" (unless ".LIST" is already the right-most qualifier) or replacing a right-most qualifier of ".TEXT" with ".LIST".

  3. If a file-id is not given, the name of the input file is examined. If the right-most qualifier of that data set is ".TEXT", a name is generated by replacing ".TEXT" with ".LIST". If the right-most qualifier is not ".TEXT", an error results. In this case, a file-id must be specified.

  For example,

File Specification	Input DSNAME	Generated Output DSNAME
FILE('DOC.OUT')	N/A	DOC.OUT
FILE(DOC.OUT)	N/A	userid.DOC.OUT.LIST
FILE(DOC.LIST)	N/A	userid.DOC.LIST
FILE(DOC.TEXT)	N/A	userid.DOC.LIST
FILE((CHAP2))	N/A	userid.LIST(CHAP2)
FILE	'DOC.TEXT'	DOC.LIST
FILE	'DOC.OTHER'	*** error ***

  If an output data set of the generated name does not exist, SCRIPT/VS creates an output data set with the following characteristics:

  Organization: PS or PO
  Record format: VB or VBM
  Record length: 250
  Block size: 3156

  When a new member is created in an existing partitioned data set, the existing record format and length are used.

  If the output data set already exists, a check is made to ensure that the characteristics of that data set are compatible with the data to be produced. Specifically, if a printer-formatted document is directed to a data set which does not have the machine carriage control record format, or if a terminal-formatted document is directed to a data set which does, the command will be terminated with an error message.

- The FILE option is not supported in the ATMS-III environment.

If a document is formatted for a printer and is sent to a direct-access file, the output document has printer controls imbedded in it appropriate for the specified or implied logical output device type. The first record of this file indicates the logical device for which it was formatted and the fonts used. This initial record, which has a hexadecimal 03 (Control No Operation) carriage control character, has the following format:

```
SCRIPT/VS R2.0: DEVICE device CHARS font1 [... font4]
```

This information may be used to ensure that the document is printed on the same device for which it was formatted. The initial record is ignored when the document is printed.

In CMS, for example, you can use the CMS PRINT command to print the file. You should use the CC parameter, so that the carriage controls are correctly interpreted. For details on the PRINT command, see IBM VM/370: CMS Command and Macro Reference.

INDEX: Enable Back of Book Index

The INDEX option enables the .PI [Put Index] control word. "Chapter 7. Indexing" on page 79 describes how the .PI [Put Index] control word can be used to create an index.

LIB: Specify Symbol and Macro Libraries

The LIB option is valid in the CMS, TSO, and ATMS-III environments, and specifies that SCRIPT/VS is allowed to search the specified libraries for a definition of the symbols and macros not defined within the input file. In a batch environment, the SEARCH option provides a similar facility.

In CMS, the LIB option is specified as:

```
LIB (libname1 [ ... libname8 ] )
```

where libname is the filename of a CMS macro library. The filetype is MACLIB. The CMS search sequence is used to locate the library on any accessed disk.

In TSO, the LIB option is specified as:

```
LIB (libname)
```

If the libname given is not fully qualified (placed within quotation marks), the userid is prefixed to the libname as the leftmost qualifier, and "MACLIB" is added (unless it already appears) as the right-most qualifier.

In ATMS-III, the LIB option is specified as:

```
LIB (opnum1 [ ... opnum8 ] )
```

where opnum is an operator number. It must include the user's own number if the user's own permanent storage is to be searched.

The library is searched when a symbol or macro is not already known and SCRIPT/VS has encountered a .LY ON, a .LY SYM (for symbols only), or a .LY MAC (for macros only) control word. The library is also searched (without regard to the setting of the .LY control word) when a symbol or macro is defined with the LIB parameter. For example,

```
.se symbolname LIB

.dm macroname LIB
```

You can specify up to eight library names in CMS, one name in TSO (although multiple libraries may be concatenated by preallocating a ddname of SCRPTLIB), or eight operator numbers in ATMS-III. If the symbol name or macro name is not found in the symbol table (and the symbol or macro is defined as being in a library), SCRIPT/VS scans each library named in the LIB option (in the order given) until the symbol or macro is found. SCRIPT/VS then moves the symbol or macro definition into its symbol table, so that a second occurrence doesn't require a library search. If no library option is specified, the symbolname or macro is searched for in the default library (if it exists).

- In CMS, a symbol and macro library is a standard MACLIB file. Its file type is MACLIB, and the default library is GML2 MACLIB.

- In TSO, a symbol and macro library is a partitioned data set. The default library, unless changed by your installation, is SCRIPT.MACLIB, and is concatenated to the library you specify.

  If the LIB option is not specified, but instead a user allocates a partitioned data set with the DDname of SCRPTLIB, SCRIPT/VS uses whatever data sets are allocated to this DDname to resolve symbols and macros. Any number of data sets may be concatenated in this manner, and SCRIPT.MACLIB is not included in the concatenation.

  If the LIB option is not specified and a DDname of SCRPTLIB is not allocated, SCRIPT.MACLIB is used.

- In ATMS-III, the LIB search is used only if the requested source can not be located through the use of ATMS subdocument index build/connect facilities. Futhermore, the search is performed only against the permanent storage of the users whose operator numbers are specified in the LIB list. Also, if the LIB search is used and the located source does not belong to the requesting user, the document must have a getword of "ANY".

  If a LIB option is not specified, ATMS-III will use only its subdocument facilities to search for unresolved symbols and macros.

MESSAGE: Control Message Printing

The MESSAGE option controls the amount and timing of the information SCRIPT/VS provides with error messages. If the MESSAGE option is not specified, SCRIPT/VS provides a short message that includes the message text and, when appropriate, the line number and text of the input last read when the error was detected.

The MESSAGE option is specified as:

```
    MESSAGE ( [DELAY] [ID] [TRACE] )
```

You must specify at least one parameter with the MESSAGE option; you may specify two or all three parameters, separated by blanks. Each of the options may be abbreviated as a single letter.

<u>D</u>ELAY requests that SCRIPT/VS not display messages while a document is being displayed or printed. SCRIPT/VS accumulates messages in a utility file and appends them to the end of the formatted output. DELAY is always used in ATMS-III.

<u>I</u>D causes SCRIPT/VS to include the error message identifier along with the error message.

<u>T</u>RACE causes SCRIPT/VS to list, whenever appropriate, the sequence of imbedded files, from the file that includes the error input line backward to the primary input file. This is useful when a file is imbedded in many other files.

Note: CMS will truncate messages which exceed 130 characters in length.

NOPROF: Suppress the Profile

The NOPROF option requests that SCRIPT/VS not imbed a Profile document. For details about the Profile, see the PROFILE option's description below.

NOSPIE: Prevent Entering SPIE Exit Routines

The NOSPIE option requests that SCRIPT/VS not establish a program interrupt exit. The NOSPIE option is intended for use by the system programmer who is maintaining SCRIPT/VS.

The NOSPIE option is valid only in CMS and TSO.

NOWAIT: Prevent Prompting for Paper Adjustment

The NOWAIT option causes SCRIPT/VS to send output to your terminal without first prompting you to adjust the paper. NOWAIT option is the normal mode for output to other than a typewriter terminal.

The NOWAIT option is valid only in CMS and TSO.

NUMBER: Print the File Name and Line Number

The NUMBER option causes SCRIPT/VS to print the file-id and line number of the last line read when a formatted output line is printed. The file-id and line number are printed to the right of the formatted output line, and each is a maximum of eight characters in length.

OPTIONS: Name a File That Contains Options

The OPTIONS option allows you to specify a file that contains, in essence, an extension to the SCRIPT command options list. The options in the file are in addition to options you specify with the SCRIPT command and with other "options" files.

The OPTIONS option is specified as

```
    OPTIONS [ (file-id) ]
```

In CMS, if the _file-id_ is not specified, the default _file-id_ is SCRIPT OPTIONS; if only a filename is given, the default filetype is OPTIONS.

In ATMS-III, the user may specify OPTIONS('docname:opnum;getw'). If "opnum" and/or "getw" is not specified, ATMS-III will first attempt to locate the source name via its subdocument index build/connect facilities. If it cannot be located as a subdocument, ATMS-III will attempt to locate it as an explicit document in the permanent storage of the requesting user. If a qualified source name is used, ATMS-III will only use an explicit search for the document.

Each record in the options file can contain one or more options, in the same format as they would appear on the SCRIPT command line. They must, however, be in uppercase. An option need not be completed on a single line (suboptions may appear on following lines), but each word must be completed in a single record. A left parenthesis _must not_ precede the options in the file.

The options in the file are processed as though they replace the OPTIONS option. Consequently, the OPTIONS option in one option file can refer to another option file. Options files can be chained together in this manner. Alternatively, the OPTIONS option in the SCRIPT command line might refer to a file that contains a list of OPTIONS options, each of which points to a different options file.

The OPTIONS option is valid only in CMS and ATMS-III.

PAGE: Selectively Print Pages

The PAGE option allows you to print pages of formatted output selectively. The page number need not be an integer; you can use the .PN [Page Numbering Mode] control word to establish decimal, alphabetic, or Roman numeral page numbers, and attach a prefix to each page number. The first eight characters of the page number you specify with the PAGE option is the character string SCRIPT/VS compares to the current page number symbol.

The PAGE option has several formats, and any number of page range specifications may be included in the PAGE option. Note, however, that prompting mode replaces the remainder of the suboptions. Valid forms of page range specifications are:

```
PAGE [ (PROMPT) ]

PAGE [ ([FROM] frompage [TO] topage) ]

PAGE [ ([FROM] frompage FOR n) ]

PAGE [ ([FROM] page ONLY) ]
```

If no parameter is given with the PAGE option, PAGE (PROMPT) is assumed (except under ATMS-III, where it is ignored).

The following are examples of valid explicit page range specifications:

```
PAGE (FROM 10 TO 15)

PAGE (7 FOR 2)

PAGE (viii ONLY 93 TO *)

PAGE (FROM 95.1 FOR 3 99 ONLY)
```

An asterisk (*) specified as *frompage* is interpreted as the current page; an asterisk specified as *topage* means the last page in the document.

If you use the ONLY option, the page number you specify before ONLY will be the single page that is printed.

If you specify or imply the PROMPT option, SCRIPT/VS will ask you to enter page range specifications from your terminal. You may respond with any of the forms described above, and SCRIPT/VS will continue to ask for new page range specifications until the end of the document is reached or you indicate an end to prompting mode by entering a null line.

If there is a syntax error in your page range specification, SCRIPT/VS issues an error message and begins prompting.

The page numbers must be entered in the same order as they appear in the output document. For example, you can specify

```
PAGE (6 1)
```

but it will be meaningful only if there is, at some point following page 6, a .PN 1 or .PA 1 control word that resets the page counter to 1.

If there is no page with the number given or if SCRIPT/VS has passed the specified page, SCRIPT/VS will reach the end of the document without changing from not printing to printing, or vice versa.

PRINT: Produce Printer Output

The PRINT option causes SCRIPT/VS to send the output document to a printer. If the DEVICE option is not specified, SCRIPT/VS assumes the logical device 1403W6. If PRINT is specified for a device other than the 3800, the output will contain 1403 carriage control characters. Specifying the DEVICE option will cause a previously specified PRINT option to be ignored.

In CMS, the number of copies and the output class are controlled by the CP SPOOL command and the CP CHANGE command.

In TSO, you can control the disposition of the printed output by specifying the following positional parameters with the print option:

```
    PRINT (copies,class,fcb,ucs)
```

copies is the number of copies desired, and defaults to one. class is the SYSOUT class. Unless changed by your installation, class defaults to "A" when the UPCASE option is specified and "T" when it is not. fcb is the forms control buffer name. ucs is the universal character set name. Your installation determines the appropriate values for class, fcb, and ucs.

Note that under JES2, if the default LINECT value is not zero, JES2 may insert extra page ejects into your document when it is printed. You may circumvent this by directing your formatted output to a file and then submitting a job to print the contents of the file specifying LINECT=0. If SYSOUT parameters such as CHARS, FLASH, FORMS, OPTCD, etc., are desired, you should also direct your formatted output to a file and submit a job to print the contents of the file specifying the desired parameters.

For more information on fcb, ucs, and LINECT, see the OS/VS2 MVS JCL manual.

In ATMS-III, the number of copies and output class are controlled by the peripheral queue. When an ATMS-III peripheral queue is defined for SCRIPT/VS, a CICS/VS destination ID must be specified that is consistent with the queue device type.

SCRIPT/VS does not control the 3800 Printer in any environment. You are required to use the proper facilities and procedures established at your installation in order to properly route and obtain correct output results.

PROFILE: Specify a Profile

A profile is a SCRIPT input file that is imbedded before processing begins on the primary input file.

The PROFILE option names the file that SCRIPT/VS is to use as the profile for the document being formatted. A profile can contain frequently used symbol and macro definitions, GML application processing functions, and text appropriate for many documents (for example, top and bottom titles).

For details about creating a profile, see "Chapter 14. GML Support in SCRIPT/VS" on page 159.

The PROFILE option is specified as:

```
    PROFILE [ (file-id) ]
```

file-id names the profile. You can select different profiles to use when formatting the document for different applications.

If the PROFILE option is not specified or if file-id is not specified, SCRIPT/VS searches your files for a file named PROFILE.

In CMS, the default is:

```
    PROFILE SCRIPT
```

In TSO, the default is:

```
    'userid.PROFILE.TEXT'
```

In ATMS-III, the default is:

```
    PROFILE
```

The PROFILE option also provides an "epifile" facility through the use of the .EF [End of File] control word. If "End of File" is indicated with the .EF control word within the PROFILE file, SCRIPT/VS will proceed immediately to the primary input file; the remainder of the PROFILE file will be processed <u>after</u> the primary input file is completed.

The .EF control word is described in ".EF [End of File]" on page 255.

QUIET: Suppress the Formatter's Identifier Message

The QUIET option causes SCRIPT/VS to suppress the version identification message that is otherwise typed or displayed as a response to the SCRIPT command.

SEARCH: Specify a Library

The SEARCH option, in a TSO, ATMS-III, or batch environment, causes SCRIPT/VS to search the specified library or partitioned data set for imbedded files. In a batch environment, SCRIPT/VS also uses the library to locate symbols and macros not defined within the input file. For more details on libraries, see the LIB option.

In TSO, the SEARCH option is specified as:

```
SEARCH (libname)
```

The specified library name is used as described in "TSO Naming Conventions" on page 14.

In ATMS-III, the SEARCH option may be specified as:

```
SEARCH (opnum1 [ ... opnum8 ] )
```

Up to eight ATMS-III user operator numbers may be specified as an alternative source for imbedded documents.

If an imbedded document with an unqualified name can not be located as a subdocument or as an explicit document in the permanent storage of the requesting user, ATMS-III will search, in the specified order, the permanent storage of the operators specified. If the located documents are not in the operator's permanent storage, they must have a getword of "ANY".

The SEARCH option is invalid in a CMS environment.

SPELLCHK: Enable the .SV Control Word

The SPELLCHK option causes SCRIPT/VS to verify spelling. Each word is verified using the spelling and hyphenation dictionaries specified with the .DL [Dictionary List] control word, unless spelling verification has been turned off with the .SV [Spelling Verification] control word. Spelling errors are listed, using the .UW [Unverified Word] control word, with other errors found during formatting.

STOP: Print Separate Pages at the Terminal

The STOP option causes SCRIPT/VS to wait for you to press the return key before starting to type each page. Use this option when printing your output document on separate sheets of paper at a typewriter terminal.

When SCRIPT/VS stops after the first page, no message is issued. SCRIPT/VS unlocks your keyboard, and is ready to type the first line of the new page. Position the typewriter at the top edge of the paper, and press RETURN. SCRIPT/VS will resume typing with the next page.

The STOP option is valid only for typewriter-like terminals in CMS and TSO.

SYSVAR: Set System Variable Symbols

The SYSVAR option allows you to pass information to SCRIPT/VS as defined symbols when you issue the SCRIPT command.

The SYSVAR option is specified as:

```
SYSVAR (x value ... x value)
```

Each x value pair causes the symbol &SYSVARx to be set to value. x is any alphameric character identifying the token. Value is any alphameric string of up to eight characters, and cannot contain imbedded blanks or parentheses. Because both x and value are part of the SCRIPT statement, any lowercase characters you specify will be converted to uppercase.

The maximum number of x value pairs is limited only by the length of the SCRIPT command line.

For example, your input file might include the lines

```
.in &SYSVARA
.ll &SYSVARL
```

When you issue the SCRIPT command to format your document, you can specify values for indention and line width as:

```
SCRIPT ... SYSVAR (A 10 L 72)
```

The symbols on the input line are substituted with the values set by the SYSVAR option. The input lines shown above are processed by SCRIPT/VS as though they had been:

```
.in 10
.ll 72
```

Note: The symbols created by the SYSVAR option are always uppercase.

TERM: Display the Output at the User's Terminal

The TERM option causes SCRIPT/VS to send the output document to your terminal. If the DEVICE option is not specified, SCRIPT/VS assumes the logical device TERM and displays the document on your terminal.

TERM is the default destination in CMS and TSO; ATMS-III forces TERM when SCRIPT/VS is called from a terminal.

TLIB: Specify Spelling Checking and Hyphenation Libraries

The TLIB option specifies text libraries that contain IBM-supplied root word dictionaries, user-created root word dictionaries, and stem processing routines for use in spelling checking and hyphenation.

The TLIB option is specified as:

```
TLIB ( libname1 [ ... libname8 ] )
```

where libname is the name of a CMS text library. The filetype is TXTLIB. The CMS search sequence is used to locate the library on any accessed disk.

- The specified libraries are searched when a dictionary is named in the .DL [Dictionary List] control word which is not included as part of the SCRIPT/VS load module. Both the dictionary and stem processing routines are loaded from the libraries.

- If the TLIB option is not specified, the library searched is SVTEXT TXTLIB.
- If the dictionary or dictionaries used are included as part of the SCRIPT/VS load module when it is created, no library is needed.

The TLIB option is valid only in CMS.

TWOPASS: Prepare the Document With Two Formatting Passes

The TWOPASS option causes SCRIPT/VS to process the input file in two passes. Both passes process all control words, but output occurs only on the second pass. Unless you specify TWOPASS, SCRIPT/VS formats and outputs everything in one pass.

Two formatting passes are required when a symbolic value is needed earlier in the document than when it is set; for example, a page number in a table of contents or list of figures. The first formatting pass allows SCRIPT/VS to collect head-levels and corresponding page numbers. The second formatting pass, which produces output, includes accurate page numbers in the table of contents.

You can produce an accurate table of contents with a single formatting pass by having SCRIPT/VS prepare the table of contents at the end of the output document. Later, you can move the table of contents pages to the front of the document. If you do this, be sure to reset the page number before the table of contents.

You can also use the TWOPASS option to detect errors in an input file. If you process a document with TWOPASS and without CONTINUE, the second pass will not begin unless the first pass is completed.

If TWOPASS is used while processing a file that uses .TE [Terminal Input], text entered as a result of .TE on the first pass will be excluded from the formatted output. Text entered during the second pass, however, will be formatted. You can use the TWOPASS symbol, &$TWO (which is equal to 1 if TWOPASS was specified), and .IF [If] to skip the .TE on the first pass. See the discussion of the .IF [If] family of control words in "Chapter 10. Conditional Processing" on page 111.

Note: The SCRIPT/VS symbol and conditional processing functions might cause the input file to look entirely different on the second pass than it did on the first pass. As a result, page numbers might not be accurate in the table of contents or in other cross-references. If the table of contents is placed at the front of the document, it will reflect the page numbers on the first pass. If it is placed at the end of the document, it will reflect the page numbers on the last pass.

UNFORMAT: Print All Input Lines Without Formatting

The UNFORMAT option causes SCRIPT/VS to print all input lines as they appear in the input file. The lines that are produced in an unformatted listing represent all (and only those) lines that will be processed by SCRIPT/VS: For example, Input lines that are not processed as a result of a .GO [Goto] control word or are ignored because of a .CS [Conditional Section] control word are not shown in the unformatted listing.

Some lines not in the primary input file might be printed. When SCRIPT/VS encounters the .IM [Imbed] or .AP [Append], control word, the contents of the imbedded or appended file are included following the control word. In the unformatted listing, SCRIPT/VS puts the line

```
.*===> IMBED/APPEND FILE: file-id
```

at the beginning of each imbedded or appended file. The file-id is always listed. SCRIPT/VS puts the following line after the last line of an imbedded or appended file:

```
.*<=== END OF FILE: file-id
```

If the NUMBER option is used with UNFORMAT, the file-id and line number are printed on the left instead of the right.

UPCASE: Print Lowercase Letters as Uppercase

The UPCASE option causes SCRIPT/VS to convert, for the formatted output document only, all lowercase letters to uppercase. This option should be specified when the output is directed to a printer that cannot print lowercase letters.

@user-option: User-defined Options

In CMS, additional options can be defined as needed by prefixing the new option with the character "@". For example,

```
@duplex
```

User-options may also have suboptions. For example,

```
@columns (2)
```

User options are saved with their suboptions, if any, but without the "@" delimiter or the parentheses surrounding the suboptions, and can be executed at any time with the .GS [GML Services] control word. See "Chapter 14. GML Support in SCRIPT/VS" on page 159 for details.

GML MARKUP AND CONTROL WORDS

When you prepare a document for SCRIPT/VS to format, the document (called the input file) can contain two kinds of data:

- Text, the actual content of the document which SCRIPT/VS places on your output page, and
- Markup, which consists of
  - SCRIPT/VS control words that control processing of your document and the placement of the text on the output page.
  - GML markup that describes the characteristics of the document, but does not specify processing. When GML markup is used, the application processing functions (APFs) contain the control words that specify the processing.

A SCRIPT/VS input file might contain text data only. In this case, SCRIPT/VS formats the file using a set of defaults appropriate for the logical output device. Typical default values specify the output page as 8-1/2 by 11 inches, single-column format, with concatenation and justification.

Insert control words into the input file when you want to change any of the default assumptions and when you want to use the more advanced functions of SCRIPT/VS, such as footnotes, automatically generated table of contents, and interactive text input.

The _Document Composition Facility Generalized Markup Language: Starter Set Reference_ describes how to mark up a document with GML tags. This manual also discusses SCRIPT/VS control words.

SCRIPT/VS TEXT FORMATTING

SCRIPT/VS can format input text to build output lines. The output text appears in columns of uniform width. This formatting consists of two processes which SCRIPT/VS performs as it builds output lines:

- _Concatenation_: moving words from one line to another to put as many words as possible on each output line, and
- _Justification_: distributing space between words to align the right edges of output lines (right-justified).

Format Mode

Most writing that you do requires some kind of formatting. With format mode on, lines that are entered in a SCRIPT/VS file as:[7]

```
The quick brown
fox
came over to greet the lazy poodle.
The lazy poodle was
as indifferent
as the fox was quick.
```

result in the output lines:

[7] Many of the examples of SCRIPT/VS formatting in this book are shown, for convenience, with short lines.

```
The quick brown fox came over to
greet the lazy poodle. The lazy
poodle was as indifferent as the
fox was quick.
```

When SCRIPT/VS reads input, it "saves" words until it accumulates enough of them to fill an entire line. When the next word in the input would make the line too long, SCRIPT/VS justifies and prints the line, then begins formatting the next line. When two input lines are joined (that is, concatenated), SCRIPT/VS inserts blank space between the last word of one line and the first word of the next.

If you enter text in a SCRIPT/VS file with no markup, the defaults established by SCRIPT/VS cause the text to be formatted (concatenated and justified) as in the above example.

There may be occasions when you do not want SCRIPT/VS to concatenate and justify the input lines. You may want to present a simple list, such as:

```
Boston
Chicago
New York
Providence
```

If these lines are processed when SCRIPT/VS formatting is in effect, the four names are concatenated as follows:

```
Boston Chicago New York Providence
```

To prevent this, you can use the .BR [Break] control word between each entry to force a "break",[8] or you can use the .FO [Format Mode] control word to suspend SCRIPT/VS justification and concatenation:

```
.fo off
Boston
Chicago
New York
Providence
```

To restore normal formatting, use the control word:

```
.fo on
```

Since ON is the default, you can also specify:

```
.fo
```

If you use the .FO OFF control word when you create tables or charts, remember to turn formatting back on when you resume entering text.

Concatenation and Justification

You can suspend concatenation and justification by specifying the .FO [Format Mode] OFF control word:

```
.fo off
```

With concatenation suspended, each input line results in a new output line. It is not joined to the previous input line. If the input line is shorter than the output line width, it will not be padded with blank spaces.[9] If the input line is longer than the

---

[8] The .BR [Break] control word is discussed later in this chapter under "Breaks" on page 38.

output column width, the placement of excess words depends on the other parameters of the .FO [Format Mode] control word:

- EXTEND: the excess words are printed on the same output line; the line is allowed to extend beyond the column width.
- FOLD: the excess characters are printed on the next output line.
- TRUNC: the excess characters are truncated at column width and are not printed.

With .FO FOLD or .FO TRUNC, a word is divided at the last character to fit in the column.

To restore both concatenation and justification, specify

```
    .fo on   -or-   .fo
```

Ragged Right

The .FO [Format Mode] OFF control word suspends both concatenation and justification. When you want to produce SCRIPT/VS output that resembles typewriter output (that is, "ragged right" output), you want each line to contain as many words as can fit on the line, but you do not want extra space inserted between the words to pad the line to a specific length. To achieve this, use the .FO [Format Mode] LEFT control word:

```
    .fo left
```

When the .FO [Format Mode] LEFT control word is in effect, output is formatted as in the above paragraph. To resume justification of output lines, use the ON parameter of the .FO control word:

```
    .fo on
```

SCRIPT/VS IMPLICIT FORMATTING CONVENTIONS

Unless you specify otherwise, SCRIPT/VS formats your document based on default settings appropriate for the logical device you have specified.

When input lines are concatenated, SCRIPT/VS inserts a blank at the end of the output line before joining it to the next input line. This blank is the interword space between two input lines on the same output line.

If you follow the typing convention that requires sentences to be separated by two blanks, you must enter both blanks if you enter a full stop in the middle of an input line. SCRIPT/VS will automatically insert two blanks after a full stop if it occurs at the end of an input line.

A "full stop" is a period (.), a question mark (?), or an exclamation point (!). A line is also considered to end in a full stop if it ends with a double quotation mark (") or a right parenthesis ()), and the next-to-last character is a "full stop" character. You can use the .DC [Define Character] STOP control word to change the characters that are treated as "full stop" characters (see "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87 for details).

Input lines that start with a leading blank or leading tab cause "breaks." SCRIPT/VS generates a control word and executes it when it detects one of these situations. For leading blanks, the .LB [Leading Blank] control word is generated, and for leading tabs,

[9] Unless the .JU [Justify Mode] ON control word is also specified.

the .LT [Leading Tab] control word is generated. These control words do the same thing as the .BR [Break] control word.

SCRIPT/VS implements these implicit breaks as control words to allow you to alter the processing for these situations. You can define a .LB or .LT macro to provide whatever processing you require.[10]

USING TABS IN SCRIPT/VS

To generate the tab character (hexadecimal 05) in your input lines, you can use one of the following techniques:

- Choose a character that you would not normally use in your text and assign it hexadecimal 05 using the .TI [Translate Input] control word. For example, to set the "¬" character to a tab character, specify

  ```
  .ti ¬ 05
  ```

  This causes every "not sign" character to be translated to a tab in the input line, before formatting occurs. Using this technique, you can see your "tab" characters when you edit the input file.

- Use the SCRIPT/VS system symbol "&$TAB" anywhere in a text line to create the hexadecimal 05 character. Using this technique, you can see your "tab" characters when you edit the input file. Always delimit the symbol with a period (.).

- Using an editor, build the text lines with hexadecimal 05 characters as required. This technique has the disadvantage of making the tab characters "invisible" when editing the file in normal character display mode.

- Build the input file using an input device that can generate a hexadecimal 05 in response to pressing a key (for example, pressing the TAB key on an IBM 2741 Communications Terminal). This technique has the disadvantage of making the tab characters "invisible" when editing the file in normal character display mode.

Setting Tabs

When SCRIPT/VS processes an input line and encounters a tab character, it formats the line using the current tab settings, which are established using the .TB [Tab Setting] control word.

The default tab settings (the ones SCRIPT/VS uses if you don't specify any with the .TB control word) are at every fifth character position to position 80. The numbers correspond to unqualified horizontal space units that represent the end of a tab expansion (tab stop) through which SCRIPT/VS prints blanks on the output line.

For example, if you use the default SCRIPT/VS tab settings and enter a tab character in front of each input text line, then character positions 1 through 5 of each output line will contain blanks; the text begins in character position 6. If you enter two tabs, character positions 1 through 10 are filled with blanks, and so on.

To change the default tab setting values, specify the tab settings you want using the .TB [Tab Setting] control word. For example, specifying

---

[10] Note that input lines processed in literal mode, under the .LI [Literal] control word, do <u>not</u> invoke the .LB or .LT functions.

```
.ti ¬ 05
.tb 8m 18m 30m
¬This line starts with a tab.
```

results in the following format:

```
....v....v....v....v....v....v
       This line starts with a tab.
```

If you enter a word or number (for example, a list item), followed by a tab, SCRIPT/VS fills the remaining character positions (through the next tab setting position) with blanks, then continues formatting text.[11] For example, specifying the following tab setting

```
.ti ¬ 05
.tb 14m
This¬line has been
formatted with a tab.
```

results in:

```
....v....v....v....v....v....v
This          line has been
formatted with a tab.
```

SCRIPT/VS has inserted blanks through character position 14, which is the current tab setting for the first tab.

Neither the physical tab settings nor the appearance of input lines on the terminal has any effect on the SCRIPT/VS tab positions. Once a .TB control word has been processed, the tab settings remain in effect until explicitly reset by another .TB control word.

You can add one or more tab settings to the ones that already exist by including the ADD parameter when specifying the .TB [Tab Setting] control word. For example, if your current tab settings are at positions 15m, 30m, and 45m, to put an additional tab setting at position 25m, specify

```
.tb add 25m
```

This gives you tab settings at positions 15m, 25m, 30m, and 45m.

You can remove one or more of your tab settings without respecifying the ones you want to keep. Specify the .TB [Tab Setting] with the DEL parameter and the tab settings that you want removed. For example, if your current tab settings are at 15m, 25m, 30m, and 45m, specifying

```
.tb del 15m 25m
```

leaves you with tab settings at positions 30m and 45m.

If you want to respecify all of your tab setting positions, you can specify .TB SET followed by the new tab settings that you want to have in effect. For example, specifying

```
.tb set 10m 25m 30m 45m
```

leaves you with tab settings at positions 10m, 25m, 30m, and 45m regardless of the previous tab settings.

To restore the SCRIPT/VS default values, you can specify the .TB control word without any parameters:

---

[11] Space units here are specified in "ems." For details about other ways to specify an amount of space, see "Vertical and Horizontal Space Units" on page 6.

```
     .tb
```

You can specify up to 99 tab positions using the .TB control word.

Some Uses for Tabs

Tab characters at the beginning of an input line (called leading tabs) ordinarily cause it not to be concatenated with the previous line (that is, they cause a "break" in concatenation). Therefore, you can use tab characters to create simple lists. For example, the input lines:

```
     .ti ¬ 05
     .tb 5m
     We are planting:
     ¬Marigolds
     ¬Peonies
     ¬Cucumbers
```

are formatted as:

```
     ....v....v....v....v....v....v
     We are planting:
          Marigolds
          Peonies
          Cucumbers
```

Tab Fill Characters

Ordinarily, SCRIPT/VS uses blank space to pad a line to a tab position. Instead of blank space, you can specify a "fill" character to be used to pad the line. You issue a .TB control word with the tab setting parameters preceded by the fill-character parameter. The two parameters are separated by a slash (/).

When you enter a line that includes a tab, the character positions normally padded with blanks are padded with the fill character (periods, in the following example) instead:

```
     .ti ¬ 05
     .tb ./5m
     ¬This line begins with a tab.
```

is formatted as:

```
     .....This line begins with a tab.
```

You can specify a different fill character for each tab setting position you specify with the .TB control word. For example,

```
     .ti ¬ 05
     .tb ./5m */10m -/15m
     A¬B¬C¬D
     E¬¬F
```

results in:

```
     A....B****C----D
     E....*****F
```

BREAKS

When you want an input line to begin a new line of output, you must cause a break. The break causes SCRIPT/VS to "promote" the partial output line that is being built before it processes the next input line.

If you begin a line with a blank or a tab, the formatting process is interrupted, the text that has accumulated for the current output line is "promoted," and the next input line begins a new output line.

To create paragraphs in text, one method you can use is to enter spaces before each line that begins a new paragraph. For example,

```
The quick brown
fox
came over to greet the lazy poodle.
Notice that the above sentence
contains each letter of the
alphabet, except the letters J and S.
That's why the quick brown fox usually jumps.
 But the poodle was frightened
and ran away.
```

results in:

```
The quick brown fox came over to
greet  the  lazy  poodle. Notice
that    the    above    sentence
contains  each  letter  of  the
alphabet,  except the letters J
and  S.  That's  why  the  quick
brown fox usually jumps.
 But  the poodle was frightened
and ran away.
```

You can specify a break using the .BR [Break] control word.

```
The quick brown
.br
fox came over to greet ... but you know the rest.
```

results in:

```
The quick brown
fox came over to greet ... but
you know the rest.
```

Without the .BR [Break] control word between the two input lines, the above input lines format as:

```
The quick brown fox came over
to greet ... but you know the
rest.
```

Some SCRIPT/VS control words cause a break in addition to their explicit function. For a complete list of the control words that cause a break, see Figure 25 on page 354.

CHANGING THE MARGINS

SCRIPT/VS formats text into the defined column or columns on the page.

The "left margin" is the leftmost print position in the column. This is always character position one.

The "right margin" is the right edge of the rightmost print position in the column. The right margin is determined by the column width. For example, if the column were 38M wide, the "right margin" would be at character position 39.

When the left or right margin is modified, the new margin is called the "current" left or right margin respectively.

To improve readability or emphasize a block of text, it may be desirable to alter the left or right margin. Two SCRIPT/VS control words are provided for this purpose:

- .IN [Indent] - change the left margin for subsequent output lines.
- .IR [Indent Right] - change the right margin for subsequent output lines.

```
Current margins:
|<——                                                       ——>|
The current  left margin is  either character  position 1, or  the character
position  established  by the  combined  effect  of  the .IN  [Indent],  .OF
[Offset], .UN  [Undent] and  .IL [Indent  Line] control  words. The  current
right margin is determined by the combined  effect of the .CL [Column Width]
and .IR [Indent Right] control words.

.IL [Indent Line]:
———>|
    The first line following  the indent line control word is  moved to the
right of the current left margin. All  subsequent lines start at the current
left margin. (Changes affect the current left margin for one line.)

.IN [Indent]:
—————>|
         All lines following the indent control  word are moved to the right
         of the current left margin. (Changes affect the current left margin
         for all subsequent lines until respecified.)

.OF [Offset]:
         |——>|
         The first  line following the offset  control word is  not indented
              from  the  current  left margin;  all  subsequent  lines  are
              indented. The offset remains in effect until changed by anoth-
              er offset or indent control  word. (Changes affect the current
              left margin after one output line.)

.UN [Undent]:
    |<——|
    The line following the undent control word is shifted to the left of the
         current left margin; all subsequent lines start at the current left
         margin. (Changes affect the current left margin for one line.)

.IR [Indent Right]:
                                                                    |<———
All lines following  an indent right control word are  justified to the
column width  minus the  right indention.  (Changes affect  the current
right margin for all subsequent lines until respecified.)
```

Figure 6. How the Current Margins Are Established

All margin-modifying control words normally cause a break. The NOBREAK parameter of .IN and .IR inhibits this function.

Simple Indention

The most basic form of indention is simple modification of the left or right margin. When the "indent" is zero, all text output lines originate in the leftmost print position of the column. By increasing the indent, the left margin can be moved to the right. For example, by specifying

```
    .in 6m

——6m—>
      the left margin is set 6M to the right of column origin. The
      left margin may also be changed by specifying an incre-
      mental value to be applied to the current left margin. This
      is called "relative" indenting. For example, by specifying

          .in +5m

————11m———>
           the value 5M is added to the current left margin. In
           this example, 6M + 5M is 11M, so the current left mar-
           gin is now 11M to the right of the column origin. You
           can move the current left margin to the left by speci-
           fying a negative value. For example, by specifying
```

```
            .in -3m

——8m——>
        the value 3M is subtracted from the current left margin.
        In this example, 11M - 3M is 8M, so the current left mar-
        gin is now 8M to the right of its origin.
```

You can return the left margin to the column origin by specifying

```
     .in 0      -or-      .in
```

The right margin can be easily changed with the .IR [Indent Right] control word. With justification on, the last character in each line is flush with the right margin. By changing the "right indent" the right margin may be moved to the left.

For example, by specifying

```
     .ir 8m
```

```
                                                             <—8m——
the right margin is moved 8M to the left. As with .IN [In-
dent] you can modify the current right margin using rela-
tive values. For example, by specifying
```

```
     .ir +3m
```

```
                                                          <——11m———
the value 3M is added to the current right indent. In
this example 8M + 3M is 11M, so the current right mar-
gin is now 11M to the left of its origin.
```

You can return to the original right margin by specifying

```
     .ir 0      -or-      .ir
```

In practice it is more convenient to use relative indention rather than absolute indention. The advantage of relative indention is that you need not be sensitive to the actual value of the margin that you are changing. Relative indents will work "in context" with the surrounding text so that the document may be imbedded into another while maintaining the same relative appearance.

Temporary and Permanent Indention

| Ordinarily, indention set with the .IN [Indent] and .IR [Indent Right] control words is permanent until changed by a similar control word. However, if a vertical extent is specified with the FOR parameter, the change is temporary; the indention reverts to the permanent value when the specified amount of vertical space has been formatted.

For example, to indent just the first line of a paragraph, specify:

```
     .in .5i for 1
```

   The indention of one-half inch is temporary, and lasts for only one line. The second line reverts to the left margin.

To create a hanging indent, a negative temporary indention may be applied to a permanent indention. For example,

```
     .in 5cm
     .in -3cm for 1
```

```
           Subsequent  text  will  be  indented  five  centimeters,
                       except for the first line, which will be
                       indented only two centimeters.
```

The .IL [Indent Line] and .UN [Undent] control words provide functions similar to the FOR parameter of .IN [Indent]. Figure 7 on

```
The FOR and AFTER parameters of the .IN [Indent] and .IR [Indent Right] con-
trol words  determine the  duration and extent  of temporary  indention. For
example,

     .in +1i for 1i after .5i
     .ir +1i for 1i after 1i

The FOR  parameter indicates that  the margin  change is temporary  and will
only be  in effect for  the duration specified.  The current margin  for any
line is a  combination of the permanent and temporary  indention values that
have been  specified. If you specify  the temporary indention as  a negative
            value (-), the current margin will  be decreased; if you specify
            it  as  a  positive  value  (+),  the  current  margin  will  be
            increased. After the duration of  a temporary indention has been
            reached, the current  margin reverts to the  permanent indention
            that was in affect prior to the temporary indention.
            If another temporary indention  is encountered prior
            to the completion  of an existing one,  the existing
            one is immediately terminated and  the new margin is
the sum of the permanent indention  margin and the new temporary
indention. A temporary margin change can either start immediate-
ly (if the AFTER parameter is not specified) or after the verti-
cal distance specified with the AFTER parameter. Once the values
specified with the FOR and AFTER  parameters have been satisfied, the margin
reverts to the  permanent indention that was in effect  before the temporary
margin went into effect.
```

Figure 7. Permanent and Temporary Indention

page 42 illustrates a more general use of temporary indention with both .IN and .IR.

By default, the .IN and .IR control words cause a break, and take effect on the next output line. The NOBREAK parameter may be used to suppress the break function, and the AFTER parameter may be used to delay the indention until a specific amount of vertical space has been formatted.

For example, a hanging indent list may also be created by delaying indention for one line:

```
.in 1i after 1
```

Subsequent text will be indented one inch, except for the first
            line, which will have the indention of the preceding
            text.

Using Indention with Tabs

A definition list contains definition terms of varying length followed by the text which defines these terms. To ensure that all the text lines originate at the same point on the output line, it is necessary to make each definition term appear to have the same length. This is done by following each term with a tab which is set equal to the current indention. For example, if you specify

```
.in 12m
.tb 12m
.ti ¬ 05
.in -12m for 1
.uc term¬definition
.sk 1
.in -12m for 1
BEE¬any of a number of related four-winged, hairy
insects which feed on the nectar of flowers.
.sk 1
.in -12m for 1
BEEKEEPER¬person who keeps bees for producing
honey; apiarist.
```

```
.sk 1
.in -12m for 1
BEESWAX¬a tallow-like substance secreted by
honeybees and used by them in making their
honeycomb.
```

With justification on, the result will be

<u>TERM</u>	<u>DEFINITION</u>
BEE	any of a number of related four-winged, hairy insects which feed on the nectar of flowers.
BEEKEEPER	person who keeps bees for producing honey; apiarist.
BEESWAX	a tallow-like substance secreted by honeybees and used by them in making their honeycomb.

The tab ensures that the text portion of each initial line starts at the same point on the output line as the text that follows it. If you did not use the tab, you would have to manually space the number of blanks necessary to position the first word of the text to the appropriate point. There are some disadvantages to manually entering the blank space:

- The number of keystrokes and attendant potential for error is greater.
- The blank space may be increased in width if justification is on. This problem can be avoided by using required blanks.
- The space can not always be accurately filled with manually entered blanks if you are formatting the document for the 3800 Printer.

For details on the margin-modifying control words, see "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219.

<u>VERTICAL SPACE</u>

Three of the ways you can separate lines of text with vertical space are:

- Enter a blank line.
- Use the .SK [Skip] control word.
- Use the .SP [Space] control word.

For example,

```
The quick brown fox came over to
greet the lazy poodle.
.sp
But the poodle was frightened
and ran away.
.sk
The poodle ran over to her
friend the Saint Bernard.
```

are formatted as:

```
The quick brown fox came over to
greet the lazy poodle.

But  the  poodle  was  frightened
and ran away.

The  poodle  ran  over  to  her
friend the Saint Bernard.
```

If the space generated by the .SK [Skip] control word occurs at the top of a column (or page), no blank lines are printed.

However, if the space generated by the .SP [Space] control word occurs at the top of a column (or page), the blank lines <u>are</u> printed. For this reason, you may prefer to use the .SK [Skip] control word instead of the .SP [Space] control word whenever you need blank output lines.

The .SP [Space] and .SK [Skip] control words allow you to specify an amount of vertical space. They also accept a parameter indicating how much space you want to create in the text output. For example,

```
.sp 2i
```

indicates that you want to create two inches of space in the output.

You can use blank space to cause a heading or a title to stand out. For example, the lines:

```
A Love Story
.sk 3
The quick brown fox
was eager
to meet the pretty poodle.
```

results in:

```
A Love Story



The quick brown fox was eager to
meet the pretty poodle.
```

Line Spacing

SCRIPT ordinarily places formatted text on successive output lines. You can obtain additional space between output lines using the .SL [Set Line Space] control word, which specifies the vertical depth of each output line. For example,

```
.sl .5i
```

specifies that each output line occupies half an inch of space. If you are formatting for a 6-lines-per-inch logical device, two blank lines will be inserted between each line of text. If you are formatting for an 8-lines-per-inch logical device, three blank lines will be inserted.

The line-spacing value is applied to all vertical space dimensions that are given in terms of lines. For example,

```
.sl 3
.sp 2
```

results in six lines of space, since two lines of space are requested, and each line is "triple-spaced." On the other hand,

```
.sp .75i
```

causes SCRIPT/VS to space vertically as close to three-fourths of an inch as the resolution of the logical device allows, regardless of what has been previously specified using the .SL control word. The .DS [Double Space Mode] and .SS [Single Space Mode] control words are special cases of .SL.

POSITIONING LINES ON THE PAGE

Most line positioning is based on a displacement from the left margin -- a cumbersome way to format when you want text centered between the margins or aligned with the right margin (leaving a "ragged left edge"). SCRIPT/VS allows you to center text using the

.CE [Center] control word, and to align text with the right margin using the .RI [Right Adjust] control word.

When using the .CE [Center] and .RI [Right Adjust] control words, remember that the text lines affected by these control words are not concatenated or justified.

The .CE [Center] control word adjusts an output line to provide an equal amount of space on either side of the line. The line

```
    .ce Chapter 1
```

results in:

```
                                   Chapter 1
```

The .RI [Right Adjust] control word adjusts an output line to align it with the right margin. For example,

```
    .ri Chapter 1
```

results in:

```
                                                                Chapter 1
```

Both the .CE [Center] and .RI [Right Adjust] control words allow you to specify a numeric parameter, indicating how many input lines should be centered or aligned with the right margin. For example,

```
    .ce 4
    After this control word is processed,
    the next four lines from the input file
    are centered within the current
    margins.
    However, subsequent input lines are
    processed without centering,
    to produce formatted (that is,
    concatenated and justified)
    output lines.
```

results in:

```
            After this control word is processed,
           the next four lines from the input file
               are centered within the current
                           margins.
    However, subsequent input lines are processed without
    centering, to produce formatted (that is, concatenated
    and justified) output lines.
```

You can also use the ON and OFF parameters with the .CE [Center] and .RI [Right Adjust] control words. For example,

```
    .ri on
    These lines must
    be flush with the
    right margin.
    .ri off
```

results in:

```
                                                         These lines must
                                                        be flush with the
                                                            right margin.
```

All the output lines between the .RI [Right Adjust] ON and .RI [Right Adjust] OFF control words are aligned with the right margin. No concatenation or justification takes place.

The following paragraph is formatted using the .FO CENTER control word:

Do not confuse the .CE [Center] control word with the .FO [Format Mode] CENTER control word. The .FO CENTER control word allows you to format the input lines with concatenation, producing unjustified output lines that are centered between the column's margins (that is, with ragged left _and_ ragged right edges).

The following paragraph is formatted using the .FO RIGHT control word.

Also, do not confuse the .RI [Right Adjust] control word with the .FO RIGHT control word. The .FO RIGHT control word allows you to format input lines with concatenation, producing unjustified output lines that are aligned with the right margin (that is, ragged left edge).

Perhaps you want to align part of an output line with the left margin, and the other part with the right margin, all on the same line. You can do this by using the .SX [Split Text] control word, whose format is:

```
    .sx /Left-edge text//Right-edge text/
```

which results in:

```
Left-edge text                                  Right-edge text
```

The slash (/) is used in the example above as a delimiter to separate the control word's fields. SCRIPT/VS recognizes the first character after the blank (in this case, the slash) as the delimiter character for the control word. If you want to use a slash as part of the text, use some other character as a delimiter. For example,

```
    .sx ¢SCRIPT/VS User's Guide¢¢Control Words¢
```

is formatted as:

```
SCRIPT/VS User's Guide                             Control Words
```

The space between the parts of split text can be left blank. However, you can specify a fill string or leader that can either be centered or repeated as often as necessary to fill the space between the two parts of the split text.[12] For example,

```
    .sx /Left side/*-/Right side/
```

results in:

```
Left side  *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-  Right side
```

while specifying

```
    .sx c /Left side/middle/Right side/
```

results in:

```
Left side                    middle                    Right side
```

If the left-side text of the output line does not fit on a single line with the right-side text, SCRIPT/VS will, by default, truncate the portion of the left-side text that does not fit. To prevent this, specify the F parameter. This parameter causes SCRIPT/VS to fold the portion of the left-side text that does not fit on the current line onto the next line, along with the fill character and the right-side text. This parameter can be partic-

---

[12] A fill string or leader that is to be repeated can be up to eight characters in length. If it is to be centered and not repeated, it may be as long as the space remaining between the left-side text and the right-side text.

ularly useful when producing such things as a Table of Contents for documents containing long headings.

UNDERLINING AND CAPITALIZING

Because underlining requires backspacing and overstriking characters, the procedure can be particularly frustrating when you need to create a line that contains an underlined word or words. Instead of manually keying in the character/backspace/underline sequence, you can use the .US [Underscore] control word to tell SCRIPT/VS to underscore a word or phrase when it is printed.

For example,

```
    .us Do not destroy this letter.
```

prints as:[13]

    <u>Do not destroy this letter.</u>

Because the .US [Underscore] control word does not cause a break, you underscore a single word as:

```
    This sentence contains a very
    .us important
    word for contemplation.
```

which results in:

This sentence contains a very
<u>important</u> word for
contemplation.

The .UP [Uppercase] and .UC [Underscore and Capitalize] control words work in a similar manner. Instead of entering text to be capitalized all in uppercase letters, you can tell SCRIPT/VS to capitalize text for you. For example,

```
    .up Chapter 10
```

results in:

```
    CHAPTER 10
```

Use the .UC [Underscore and Capitalize] control word when you want a line both underscored and capitalized. The line:

```
    .uc preface
```

results in:

<u>PREFACE</u>

You can also affect a number of input lines with the .US [Underscore], .UP [Uppercase], and .UC [Underscore and Capitalize] control words. For example, to underscore three input lines:

```
    .us 3
    Do not
    destroy this letter
    until
    its expiration date,
    which is 12 September 1982.
```

results in:

---

[13] By default, SCRIPT/VS draws an uninterrupted rule beneath underscored text. The .UD [Underscore Definition] control word allows you to specify that blanks are not to be underscored.

```
    Do not destroy this letter until
    its expiration date, which is 12
    September 1982.
```

You can use the ON and OFF parameters of these control words to affect a group of text lines in a similar manner. Using the ON and OFF parameters might require less updating than using a numeric parameter when you add or delete lines to a group of underscored lines. For example,

```
    This is capitalized for
    .up on
    emphasis
    .up off
    and
    .uc on
    emotional
    .uc off
    impact.
```

results in:

```
    This is capitalized for
    EMPHASIS and EMOTIONAL impact.
```

USING FONTS WITH THE IBM 3800 PRINTING SUBSYSTEM

When formatting a document for the IBM 3800 Printing Subsystem, you can take advantage of that printer's dynamic font storage and use several different fonts in your document. Use the CHARS option of the SCRIPT command to specify the fonts to use when formatting the document.

SCRIPT/VS supports the fonts distributed by IBM with the IBM 3800 Printing Subsystem. However, most of the fonts distributed with the IBM 3800 are uppercase only and therefore inappropriate for text applications. (For more information about the IBM 3800 fonts, see the IBM 3800 Printing Subsystem Programmer's Guide.)

In addition to the IBM 3800 fonts, SCRIPT/VS provides sixteen complete upper- and lowercase fonts, which are listed and illustrated in "Appendix D. Fonts Supplied with SCRIPT/VS" on page 377. You can also create your own fonts to use with SCRIPT/VS as long as the font's characteristics are listed in a font table. (See "Appendix B. Device and Font Table Maintenance" on page 367 for details on how to add a new font's characteristics to a font table.)

The IBM 3800 can contain up to four uppercase-only fonts, or two complete upper- and lowercase fonts. To ensure proper output line justification, you should not specify IBM 3800 fonts of different pitches on a single line. However, each SCRIPT/VS font contains special blanks that allow the SCRIPT/VS fonts to be freely intermixed without regard to pitch.

When SCRIPT/VS begins formatting a document for the IBM 3800, the first font specified with the CHARS option of the SCRIPT command becomes the current font. If CHARS is not specified, the default font of the logical output device becomes the current font. Use the .BF [Begin Font] control word at any time to change the current font to any of those specified with the CHARS option. For example, in this manual

```
    This is a
    .bf GB12
    bold
    .pf
    word.
```

would produce the line:

    This is a **bold** word.

The .BF [Begin Font] control word saves the current font before beginning a new font; the .PF [Previous Font] control word restores the last font saved. Up to sixteen fonts can be saved.

To eliminate dependence in the file on specific font names, you can use the SCRIPT/VS symbols &$CHAR(n) instead of actual font names. The previous example could be revised as:

```
This is a
.bf &$CHAR(2)
bold
.pf
word printed in the &$CHAR(2) font.
```

which prints as:

This is a **bold** word printed in the GB12 font.

For more information on the &$CHAR system symbol, see ".BF [Begin Font]" on page 227.

All SCRIPT/VS fonts contain three special blanks that are used for justification: hexadecimal 11, 12, and 13 identify 10-, 12-, and 15-pitch blanks, respectively. These special blanks allow SCRIPT/VS to justify output lines and align columns regardless of font and pitch changes. Therefore, you should not use these hexadecimal codes with the .TI [Translate Input] and .TR [Translate Character] control words.[14]

SCRIPT/VS font capabilities include underscoring, capitalization, and, on impact printers, overstriking. "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87 describes how to define new fonts for these purposes.

FORCING A NEW PAGE

As SCRIPT/VS formats text, it keeps track of how many lines it has filled on a page. When it reaches the bottom of the output page, SCRIPT/VS performs a "page eject" and continues on a new output page. SCRIPT/VS keeps track of the current page number as it is processing.

You can force SCRIPT/VS to begin a new output page by using the .PA [Page Eject] or the .CP [Conditional Page Eject] control word:

```
.pa
```

The .PA [Page Eject] control word causes a break. SCRIPT/VS prints the output line being constructed, then leaves the remainder of the current page blank. The .CP [Conditional Page Eject] control word is described in "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87.

The .PA [Page Eject] control word also allows you to specify a numeric parameter, to assign a page number to the new page. When you specify a page number with the .PA [Page Eject] control word, the page number counter is reset to the new number and continues sequentially from that number.

For example, if you are creating a SCRIPT/VS file with a title page and you want the second output page to be numbered "1". you can enter:

```
Title page ...
.pa 1
This is page one ...
```

[14] Hexadecimal 27 also is used internally by SCRIPT/VS and therefore should not be translated with .TI or .TR.

to cause a page eject after the title page and number the following pages, beginning with 1.

For a method of suppressing the numbering of introductory pages, see the discussion of the .PN [Page Numbering Mode] control word in "Chapter 4. Defining a Page Layout" on page 55.

Starting an Odd or Even Page

You can force a new odd-numbered or even-numbered page when you specify the ODD or EVEN parameter of the .PA [Page Eject] control word. For example, if SCRIPT/VS is currently processing output page 3 and the next control word it encounters is

```
.pa odd
```

it ejects the current page, prints any titles, heading, and footing that might be in effect on the next page (page 4), ejects, and prints the next output text on page 5.

This is convenient when some of your document's pages must begin on even- or odd-numbered pages, such as the first page of a chapter, or the text that describes a figure on the facing page.

Specifying Page Eject Mode

When you want your document to be printed only on even-numbered pages (leaving the intervening odd-numbered pages blank) you can specify

```
.pa even on
```

This process is called "page eject mode." To specify page eject mode, you use the ON and OFF parameters of the .PA [Page Eject] control word, along with its EVEN or ODD parameters. You can similarly specify odd-numbered page eject mode with

```
.pa odd on
```

You can end page eject mode by issuing:

- Another page eject mode control word. For example, if the odd-page eject mode is in effect, you can change to even-page eject mode with

  ```
  .pa even on
  ```

- The OFF parameter. To turn off the odd-page eject mode, issue

  ```
  .pa odd off
  ```

- Page renumbering. You can also cancel page eject mode by specifying a page eject that resets the page number:

  ```
  .pa 12
  ```

GUIDELINES FOR ENTERING TEXT AND CONTROL WORDS IN SCRIPT/VS

You may find the following tips useful when entering input for SCRIPT/VS files.

Start All Input Lines In Column One

When you enter input into a SCRIPT/VS file, you should enter all the input lines (text lines as well as control words) beginning in column one. Occasionally, you may want to enter lines that begin with blank characters or tabs. Remember that blanks and tabs at the beginning of a line may cause breaks. When you want to manipulate the margins for output lines, use control words instead of blanks or tabs.

Avoid a Text Period In Column One

When SCRIPT/VS processes an input line, data that follows a period in column 1 is treated as a control word. If what follows the period is not a valid control word or macro, SCRIPT/VS issues an error message. If a valid control word follows the period in column 1 (even though you intended it to be text), SCRIPT/VS processes it as a control word. In this case, the results might be undesirable.

The .LI [Literal] control word tells SCRIPT/VS that you want the line interpreted as a text input line, even though it begins with a period, leading blank, or leading tab. For example,

```
.ti ¬ 05
.li ...and so it goes.
.li 2
 Leading blank lines
¬and leading tab lines
do not cause an implicit break
when preceded by the .LI
control word.
```

prints as:

```
...and  so  it  goes.  Leading
blank lines and  leading  tab
lines do not cause an implicit
break when preceded by the .LI
control word.
```

You can specify parameters with the .LI [Literal] control word. If there are many lines that begin with a period, for example, you can issue:

```
Study the following control words:
.li on
.DS,
.LI,
.PA, and
.IM.
.li off
This assignment is due on Monday.
```

which results in:

```
Study  the  following  control
words: .DS, .LI, .PA, and .IM.
This  assignment  is  due  on
Monday.
```

Note: When literal mode is in effect, the only SCRIPT/VS control word that is processed is .LI OFF. Other forms of the .LI control word, as well as other SCRIPT/VS control words, are treated as text.

Remember Which Control Words Cause Breaks

When you finish a block of text or a paragraph, you might want SCRIPT/VS to print the text that has accumulated, so that the next input line begins a new output line. You can use the .BR [Break] control word to do this. However, many other control words cause breaks as part of their normal function. In the sequence

```
text text text
.br
.in 5m
```

the .BR [Break] control word is unnecessary, since the .IN [Indent] control word causes a break.

Many control words that provide format functions do not cause breaks. (A list of those that cause a break is provided in

Figure 25 on page 354.) The underscoring and capitalizing control words are good examples:

```
This
.up sentence
.us has several control
.uc words in
.up it,
and its text is concatenated.
```

results in:

This SENTENCE <u>has several</u>
<u>control WORDS IN</u> IT, and its
text is concatenated.

THE CONTROL WORD SEPARATOR

You can enter more than one control word on a single input line. You can also enter control words and text on the same input line. To separate the control words, or the control words and text, use a semicolon (;). The semicolon is called the <u>control word separator.</u> Its effect is to allow SCRIPT/VS to separate an input line into two or more processable input lines. For example,

```
.sk;.ce on
```

is the same as the two lines:

```
.sk
.ce on
```

Grouping control words on a line is useful because you can quickly see the sequence and context of one control word within the group.

Redefining the Control Word Separator

Each time a control word line is processed, SCRIPT/VS divides it into <u>two</u> pieces: the part before the first control word separator, and the remainder (which is saved for later processing). For example, the input line:

```
.dc cw ?;.ce ;Centered;?.dc cw ;
```

is processed as follows:

Step	Active Part	Remainder
1)	.dc cw ?	.ce ;Centered;?.dc cw ;
2)	.ce ;Centered;	.dc cw ;
3)	.dc cw ;	---

In the first step, the .DC [Define Character] CW control word is separated from the remainder of the input line by the first semicolon, which is the current control word separator. The first .DC CW control word changes the control word separator to a question mark (?). When the next line is divided into the active piece and the remainder, the line is divided at the question mark. The semicolon is now an ordinary character with no special meaning.

In the second step, the .CE [Center] control word is processed, and the text ";Centered;" is centered on the output page.

In the third step, the control word separator is restored to its usual value. Semicolons that are part of a control word line now have the intended effect.

You must be careful when you use semicolons on text lines that are processed as control word lines. For example, the line

```
.us Be careful; semicolons end control word lines.
```

results, on output, in:

```
    Be careful
     semicolons end control word lines.
```

Notice that the second line caused a break because it begins with a leading blank.

To avoid this problem, use the control word modifier to suppress control word separator scanning for a single input line. For example, specifying

```
    .'us Be careful; semicolons end control word lines ...
```

results in:

```
    Be careful; semicolons end control word lines ...
```

In this case, the semicolon after "careful" is not a control word separator. The single quotation mark between the period and the two letter control word name causes all control word separator scanning to be suppressed for the rest of the input line.

Another way to avoid this problem is to use the .DC [Define Character] CW control word to indicate a character other than a semicolon as the separator for separating control words. When you specify .DC CW OFF, SCRIPT/VS does not recognize any character as a control word separator character. For example, you can enter the line above as follows:

```
    .dc cw off
    .us Be careful; semicolons end control word lines.
    .dc cw
```

The ".DC CW" line restores the initial control word separator.

The .DC [Define Character] CW control word is also useful when you define symbols. For details on symbol definition, see "Chapter 12. Symbols in Your Document" on page 129. The example above shows how to use it to solve a text input problem.

COMMENTS IN SCRIPT/VS DOCUMENTS

In addition to text and control words, SCRIPT/VS files can contain comments. Comments are useful for:

- Accounting notes: You can include comments that give your name and location, the date and reason you created a file, and a date when the file can be erased.
- Documenting formats: If you use a special format in a SCRIPT/VS file that may be accessed by other people, you can leave notes within the file explaining how to access it.
- Placeholders: If a file is only partially complete, you may want to insert comments at places where information should be added later.

To place comments in a SCRIPT/VS file, use the .CM [Comment] control word. SCRIPT/VS treats the .CM control word the same as any other control word. However, when it scans the input line that contains this control word, it will ignore the text of the comment. This means that any other control words that exist on the same input line as the .CM control word but are separated from the comment text by a control word separator will still be processed. The comments themselves will not be included in the final formatted output. For example, if you specified

```
    .cm Created:  11/3/78
    .cm Updated:  6/25/79 ;.im doc3
```

These two comments will only appear in your input file; they will not appear in the final output. SCRIPT/VS will recognize the control word separator (;) and will process the .IM control word that imbeds file DOC3.

If you do not want SCRIPT/VS to scan your comment lines for control word separators, enter them using .* instead of the .CM control word. The .* function, even though it begins with a period, is not considered a control word. Therefore, SCRIPT/VS ignores any input line that begins with .* including any other control words or control word separators that exist on that line. For example, specifying

```
    .* SCRIPT/VS ignores this line ;.im doc3
```

causes SCRIPT/VS to ignore this entire input line. Therefore, file DOC3 will never be imbedded.

CHAPTER 4. DEFINING A PAGE LAYOUT

The previous chapter showed you how to format your text to provide paragraphs, indention, formatting, and page ejects.

This chapter describes the SCRIPT/VS control words you can use to establish the page layout within which the text resides. It covers:

- Page Dimensions: The length and width, and the amount of space reserved for top and bottom margins.
- Running Top Titles: Descriptive information that is printed within the top margin, above the heading.
- Running Headings: Descriptive information that precedes the body of text on each page, printed below the top title.
- Running Footings: Descriptive information that follows the body of text on each page, printed after footnotes, if any, and above the bottom title.
- Running Bottom Titles: Descriptive information that is printed within the bottom margin, below the footing.
- Page numbering: SCRIPT/VS can automatically insert the current page number and its prefix, if any, on each page as it is formatted for printing.

Figure 8 on page 56 shows the layout of a SCRIPT/VS output page. Control words used to specify the size or contents of each area are shown in parentheses.

BASIC PAGE DIMENSIONS

The output pages that SCRIPT/VS formats are designed to fit the form size of the logical output device (for details, refer to "DEVICE: Specify a Logical Output Device" on page 21). The default logical devices are defined for a form size of 8-1/2 by 11 inches. When SCRIPT/VS formats output for logical devices that specify a form size of 8-1/2 by 11 inches, each SCRIPT/VS page has the default dimensions of:

- 11 inches long (66 lines at 6 lines per inch, 88 lines at 8 lines per inch, or 132 lines at 12 lines per inch). For 3800-type logical devices, the values are 60, 80, and 120 respectively, because one inch of the form is reserved by the 3800 Printer.
- 6 inches wide (60 characters at 10 pitch, 72 characters at 12 pitch, and 90 characters at 15 pitch).

Although the initial page length and line length values are based on the logical output device, you can change these values within your document by using the .PL [Page Length] and .LL [Line Length] control words.

In addition (if not otherwise specified), SCRIPT/VS provides space for top and bottom margins which is included in the page length. The amount of space is based on the logical output device type. Based on the logical output device, the maximum number of _text_ lines on a page is the number of lines per page less the number of lines for top and bottom margins. The .TM [Top Margin] and .BM [Bottom Margin] control words are used to respecify the top and bottom margin size.

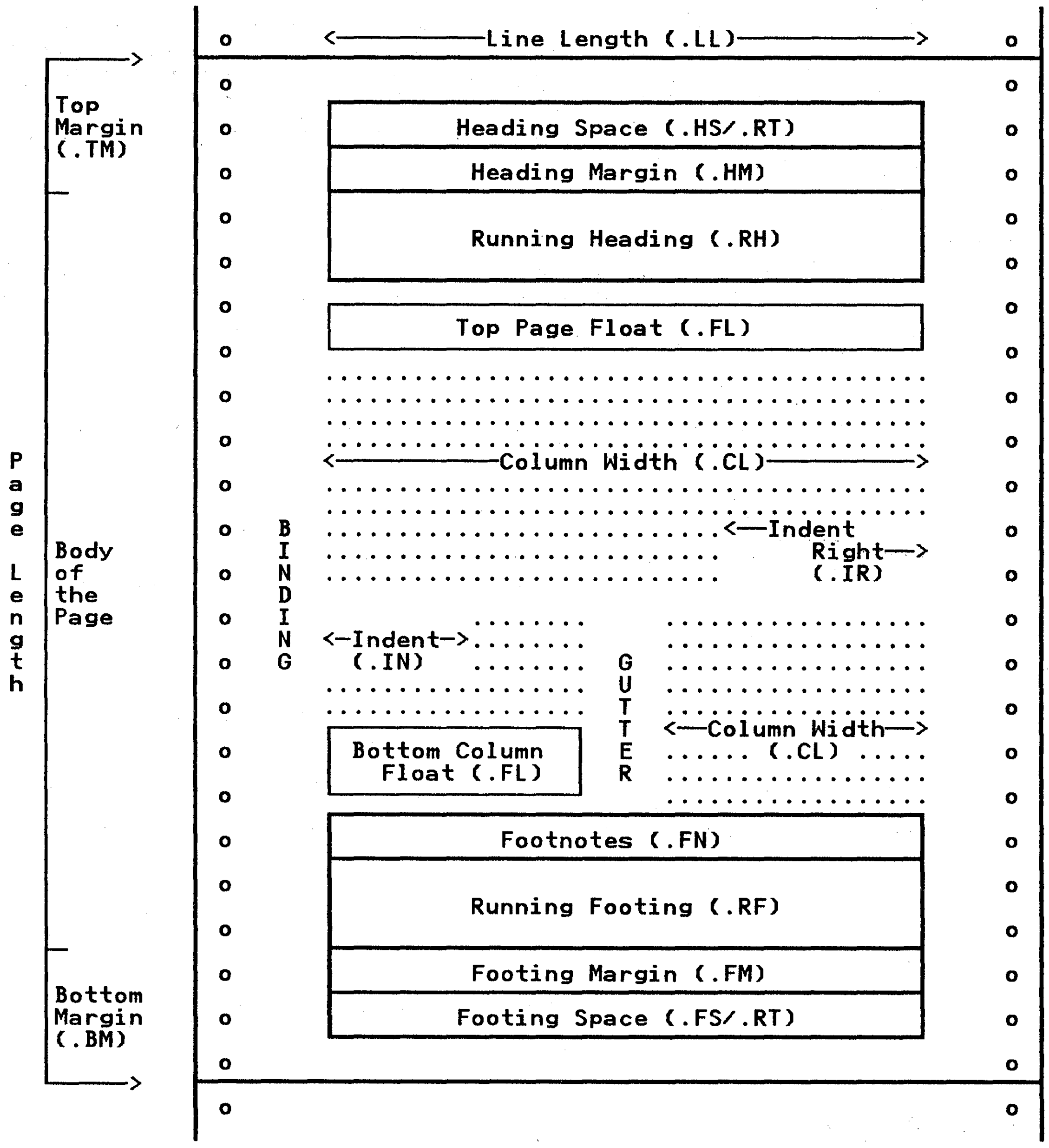


Figure 8. SCRIPT/VS Terms for Parts of the Page: Note that Top Margin and Bottom Margin include all the space on the paper that is accessible to SCRIPT/VS. For terminals and 1403-type printers, this includes the entire page. For 3800-type devices, Top and Bottom Margin do not include 1/2 inch on each side of the interpage perforation. This space is reserved by the 3800 Printer for accelerating and decelerating the paper when it is necessary to halt the paper.

By changing the values of these control words, you can adjust the dimensions of an output page. Three immediate considerations are:

- The physical size of the paper on which you are printing SCRIPT/VS output.
- The number of lines printed or typed per page on the output device.
- The 3800 Printer reserves one-half inch at the top and bottom of the page that is not included as part of the page length.

Page length includes all of the page that is accessible to SCRIPT/VS. For non-3800 devices, this is the entire form (the vertical distance between perforations for continuous forms). The 3800 Printer reserves 1/2 inch above and below the perforation, and makes it inaccessible for printing. Consequently, for 3800 logical devices, page length does not include 1/2 inch at the top and bottom of the page.

Changing the Page Margin

The .PM [Page Margins] control word causes SCRIPT/VS to shift the formatted output of each page to the right. You can use this control word to change the margins that were established using the BIND option of the SCRIPT command.[15] For example, specifying

```
    .pm 6
```

sets the page margin to six character spaces, whereas specifying

```
    .pm .5i
```

sets the page margin to one-half inch.

The current page margin can be increased or decreased by preceding the amount with a plus or a minus sign. For example, specifying

```
    .pm +9mm
```

increases the page margin by 9 millimeters.

If only one value is specified with the .PM [Page Margins] control word, it will be used for both odd- and even-numbered pages. You can set different margins for odd- and even-numbered pages by specifying two values; the first value will be used for odd-numbered pages and the second one will be used for even-numbered pages. For example, specifying

```
    .pm 6p 9p
```

causes the formatted output to be shifted 6 picas to the right for odd-numbered pages and 9 picas to the right for even-numbered pages.

If you specify the .PM [Page Margins] control word with no parameters, the value that was specified in the BIND option on the SCRIPT command will be used.

Changing the Page Length

Page length can be changed using the .PL [Page Length] control word. If you change the page length, the top and bottom margins do not change automatically.

Usually, you do not set the page length and line length for a document unless deviating from the values set for the logical device. Once set, the page length and line length values remain in effect until you explicitly reset them. You can put page layout control words into the profile. Whenever you format the document using that profile, the page layout appropriate for that document is used.

You may need to adjust a page dimension to handle a special situation in your document. Instead of recalculating the margin values, you can increase or decrease the amount of space reserved for margins. For example, if you want to reduce the number of text lines per page from 68 to 65, you can increase the amount of space for the top margin by specifying

---

[15] If the BIND option is not specified, the margin default is two character spaces.

```
    .tm +3
```

To restore the original margin, use the control word

```
    .tm -3
```

If you specify the .TM [Top Margin] control word with no parameter, the top margin is set to the default established for the logical output device.

Changing the Line Length

When you are changing the default dimensions of SCRIPT/VS output, you should consider the width of pages as well as the length. The SCRIPT/VS default is based on the logical output device, specified with the DEVICE option of the SCRIPT command. You can use the .LL [Line Length] control word to set the page width. The page width controls the right-hand margin of your output. For example, if you want a width of 8 inches, specify

```
    .ll 8i
```

The .LL [Line Length] control word controls the width of the top and bottom titles, running headings and footings, and footnotes. Column width, controlled by the .CL [Column Width] control word, defaults to the .LL value, and controls the width of each output text column. The starting position of the rightmost column plus the column width is the effective width of the page body. This can exceed the .LL value.

As with the .PL [Page Length], .BM [Bottom Margin], and .TM [Top Margin] control words, you can increase and decrease the value of the line length. For example, specifying

```
    .ll -2i
```

decreases the line length by 2 inches.

If you specify the .LL [Line Length] control word with no parameter, the line length is set to the default established for the logical output device.

When SCRIPT/VS is concatenating text, the column width (not the line length) limits the number of characters that can fit on an output line in that column.

If SCRIPT/VS is not concatenating text (.FO [Format Mode] OFF), lines that are longer than the column width print as they appear in the input file. They can extend into the right margin unless the FOLD or TRUNC parameters of the .FO [Format Mode] control word are specified. (The EXTEND parameter of the .FO [Format Mode] control word is the default.)

RUNNING HEADINGS AND FOOTINGS

The .RH [Running Heading] and .RF [Running Footing] control words provide a flexible mechanism for placing information at the top and bottom of each page. Running headings and footings appear inside of, and flush with, the body of the page. Running headings and footings can contain both text and control words, enabling you to format the information to fit your needs. For example, to center text at the top of each page, you can specify

```
    .rh on
    .ce Internal Use Only
    .sp 2
    .rh off
```

You can also emphasize the security classification of your document by specifying:[16]

```
.rh on
.bf GB12
.ce Confidential
.sp 2
.rh off
```

which places the running heading in a bold font.

Separate running headings and footings can be defined for odd- and even-numbered pages. For example, specifying

```
.rf even
.sp 2
.sx c /Page &/Introduction//
.rf off
.rf odd
.sp 2
.sx c //Introduction/Page &/
.rf off
```

centers the title "Introduction" at the bottom of each page and places the page number in the lower left corner on even-numbered pages and in the lower right corner on odd-numbered pages. The page number symbol, by default the ampersand (&), is replaced by the current page number whenever it appears in a running heading or footing definition. The .DC [Define Character] PS control word may be used to change the page number symbol.

Because running heading and footing definitions can contain both text and control words, sophisticated headings and footings can be created to fill special requirements. For example,[17]

```
.rh on
.bx 1 &$LL
.fo center
.bf GB12
Expiration Date:
.pf
.us 12 September 1982
.bx off
.sp 2
.rh off
```

will result in the following running heading being placed at the top of each page:

Expiration Date: 12 September 1982

Running headings and footings appear in the body of a page flush with the text. Ordinarily, some space is included at the end of a running heading and at the beginning of a running footing to separate the heading or footing from the body text. There may be times, however, when you want to merge a running heading or footing with the body text. For example, the heading of a multipage table might be defined as

[16] It is not necessary to restore the previous font after the .RH [Running Heading] definition because the active formatting environment is automatically saved when a running heading or footing definition is formatted and restored afterward. See "Chapter 9. The SCRIPT/VS Formatting Environment" on page 109 for details.

[17] The .BX [Box] control word is described in detail under "Drawing Boxes" on page 101.

```
.rh on
.bx 1 &$LL
.ce Parts List
.tb 3 26 53
.bx 1 14 50 &$LL
&$TAB.Part No. &$TAB.Description &$TAB.Quantity
.bx
.sp
.bx can
.rh off
```

which would produce this heading:

Parts List		
Part No.	Description	Quantity

The vertical rules of this heading can be made to line up and merge with the vertical rules in the body text on each page.

Running heading and footing definitions must be redefined in their entirety when changed. If a running heading or running footing is no longer needed, it can be completely removed by specifying

```
.rf cancel
```

If there is an occasion where you do not want to remove a running heading or footing, but you do not want it to be placed on a particular page or series of pages, you can temporarily suppress it by specifying

```
.rh sup
```

Then, when you are ready to restore it, all you have to specify is

```
.rh res
```

This automatically restores the running heading without having to redefine it.

Running headings and footings are processed in two distinct and separate phases:

- Definition phase, where the running heading or running footing definition is processed, symbol substitution (except page number symbol substitution) is performed, and some control words are executed.[18] All control words that are not executed and all text lines are saved in input form until SCRIPT/VS performs the formatting phase.

- Formatting phase, where the data saved during the definition phase is reprocessed: all remaining control words are executed, page number symbol substitution is performed, and all of the text lines are formatted. SCRIPT/VS saves the active environment before this processing begins and then restores it when this processing has been completed.

For example, if your document contains the running footing definition:

```
.rf on
.sp 2
.of 3
.sx f /&title.//Page &/
.rf off
```

[18] The control words that are executed during the definition phase are listed in Figure 28 on page 355.

the value of the symbol &title will be substituted during the definition phase;[19] the saved .SX [Split Text] control word will contain no symbols other than the page number symbol and the running footing will be the same on all pages. However, if the symbol &title is changed periodically throughout your document, such as on chapter boundaries, you may want to define the running footing as:

```
.su off
.rf on
.sp 2
.of 3
.ec .su .sx f /&title.//Page &/
.rf off
.su on
```

In this definition, the .SU [Substitute Symbol] OFF control word is immediately executed during the definition phase, causing the symbol &title to be saved as part of the data to be processed during the formatting phase.[20] You may change the value of the symbol &title, and thus the running footing, without having to constantly redefine the running footing.

Since the formatting phase is normally performed with substitution off, you must include the .SU control word as part of the definition to explicitly perform symbol substitution during the formatting phase. To prevent this control word from being executed during the definition phase, you must "execute" it with the .EC [Execute Control] control word, which delays the execution of this control word until the formatting phase is performed.

TOP AND BOTTOM RUNNING TITLES

In addition to running headings and footings, you can use the .RT [Running Title] control word to specify running titles.[21] Running titles can include such things as page numbers, chapter or section headings, document titles, or form numbers.

Running top titles appear in the heading space (part of the top margin); running bottom titles appear in the footing space (part of the bottom margin). Each title consists of three parts, separated by arbitrary delimiters:[22]

```
.rt top /left part/center part/right part/
```

- The left part is flush left (aligned with the left margin).
- The center part is centered between margins.
- The right part is flush right (aligned with the right margin of the page).

---

[19] The page number symbol will not, however, be substituted. Page number substitution is performed during the formatting phase.

[20] Substitution must be explicitly turned on again after the running footing definition; the active formatting environment is saved and restored around the formatting phase, not the definition phase.

[21] Unless you specify otherwise, SCRIPT/VS provides a default top title that prints the page number in the upper right corner of the output page.

[22] The delimiter can be any character that does not appear in the title. SCRIPT/VS assumes that the first nonblank, nonnumeric character that follows "bottom", "top", "even", or "odd" is the delimiter.

For example, to center the words "First Draft" at the top of each page, specify

```
    .rt top //First Draft//
```

SCRIPT/VS replaces the page number symbol, which by default is an ampersand (&), with the current page number whenever it appears in a running title. To include the page number in the upper righthand corner of each page, specify

```
    .rt top //First Draft/Page &/
```

When the character ampersand (&) appears as part of the title, the .DC [Define Character] control word can be used to designate another character as the page number symbol. For example, if

```
    .dc ps ¢
    .rt bottom /Reference & User's Guide//Page ¢/
```

is specified, SCRIPT/VS will automatically place the page number where the "¢" appears in the title.

You can specify different titles on odd- and even-numbered pages by using the ODD and EVEN parameters of the .RT [Running Title] control word. For example, specifying

```
    .rt odd bottom //-&-/First Draft/
    .rt even bottom /Window Operator's Manual/-&-//
```

results in the words "First Draft" appearing on the lower right of each odd-numbered output page and the words "Window Operator's Manual" appearing on the lower left of each even-numbered output page.

You can specify up to twelve running title lines at one time; six lines of titles for even-numbered pages and six lines of titles for odd-numbered pages.

The .RT [Running Title] control word allows you to specify the order in which you want the lines printed. For top titles, you can number the lines 1 through 6 as you want them to appear at the top of the page. For example, specifying

```
    .rt top 1 //Top title line 1//
    .rt top 2 //Top title line 2//
       ...
    .rt top 6 //Top title line 6//
```

results in top titles being saved for the top margin as follows:

```
                          Top title line 1
                          Top title line 2
                               ...
                          Top title line 6
```

For bottom titles, you number the lines 1 through 6 starting with the last line of the title. Thus specifying

```
    .rt bottom 1 //Bottom title line 1//
    .rt bottom 2 //Bottom title line 2//
       ...
    .rt bottom 6 //Bottom title line 6//
```

results in the following bottom titles being saved:

```
                         Bottom title line 6
                                ...
                         Bottom title line 2
                         Bottom title line 1
```

Note: All running titles are printed in the initial font.

Allocating Space for Running Titles

The default top and bottom margins, listed in Figure 31 on page 357, are defined to allow one line each for running top and bottom titles (heading and footing space), and two blank lines between the running titles and the body of the page (heading and footing margin). Therefore, when you want to use multiline titles, you must be sure to allocate enough space for them.

Use the the .HS [Heading Space] and .FS [Footing Space] control words to specify how much additional space you want for the titles. However, when specifying space for multiline titles, you should note that:

- The heading margin plus the heading space cannot exceed the amount of vertical space specified for the top margin.
- The footing margin plus the footing space cannot exceed the amount of vertical space specified for the bottom margin.
- You can specify from 1 to 6 lines for both the footing space and the heading space, but you can have no more than 6 unique title lines.

For example, if you specify the following top and heading margins,

```
.tm 8
.hm 4
```

your heading space can not be greater than 4. If you specify

```
.hs 3
```

you have allocated three lines for your top title. The extra line in the top margin will be left blank and will appear above the first line of the title.

Note: If you specify six top title lines, and you specify heading _and_ footing spaces of six, all six title lines will be printed at the top _and_ the bottom of the page.

WHERE TO DEFINE HEADINGS, FOOTINGS, AND RUNNING TITLES

SCRIPT/VS formats running headings, running footings, and running titles for each page before processing the body text for that page. Therefore, when you redefine a running heading, footing, or title, you should make sure that it is redefined _before_ a control word that causes a new page is encountered, since it will not take effect until the _next_ output page is processed. If it is necessary for SCRIPT/VS to finish processing the current page before a running heading, footing, or title is redefined, you can specify

```
.pa nostart
```

which prevents the next page from starting. You can then redefine the running heading, footing, or title for the next page. The next page will automatically be started when text for the body of that page is formatted.

Ordinarily, running headings, footings, and titles do not appear on the first page of a document. If you want them to, you must issue their definitions before any text for the body of the first page is formatted.

PAGE NUMBERS

The page number symbol is, by default, the ampersand (&), but can be changed using the .DC [Define Character] PS control word. The page number symbol is replaced, wherever it appears in a running heading, footing, or title, with the current page number of the document being processed. SCRIPT/VS uses an internal page counter to keep track of what the current page number should be. You can use the .PA [Page Eject] control word to reset this counter if you need to. For example, specifying

```
.pa 17
```

sets the internal page counter to 17 regardless of how many pages have been formatted. Subsequent pages will be incremented by one.

If you do not want page number substitution to occur, but you want SCRIPT/VS to continue counting the pages internally, you can specify

```
.pn off
```

If you do not want page number substitution or internal page counting to occur, you can specify

```
.pn offno
```

The OFF and OFFNO parameters of the .PN [Page Numbering Mode] control word can then be reset by specifying

```
.pn on
```

The .PN control word further allows you to specify the form that the current page number takes when it appears in a table of contents, index, or running heading, footing, or title. The numbers can be arabic (which is the default), roman numerals, decimals, or alphabetics.

Roman Numeral Page Numbers

When you want page numbers to be printed in lowercase roman numerals, you can specify

```
.pn roman
```

The ROMAN operand is useful for printing prefaces, forewords, front matter, and similar pages that might be numbered with roman numerals. To restore arabic numbering, you can specify

```
.pn arabic
```

Decimal Page Numbers

You can specify that you want decimal-point page numbering to begin after the next even-numbered page:

```
.pn frac
```

If this control word is encountered while SCRIPT/VS is processing page 46, then subsequent pages are numbered 46.1, 46.2, 46.3, and so on.

You can end decimal-point page numbering and resume normal page numbering when you specify

```
.pn norm
```

SCRIPT/VS ejects the page and numbers the next page 47.

Alphabetic Page Numbers

When you want page numbers to be printed as lowercase alphabetic characters, such as page a, page b, page c, and so on, you can specify

```
.pn alpha
```

To restore arabic page numbering, you specify

```
.pn arabic
```

Prefixes for Page Numbers

Large documents often use a compound page numbering scheme to facilitate the frequent replacement or addition of chapters or sections. You can use the PREF parameter of the .PN [Page Numbering Mode] control word to obtain this effect. For example, if you specify

```
.pn 1
.pn pref 1-
```

for the first chapter of a document, then its pages will be numbered 1-1, 1-2, 1-3, ... If you then specify

```
.pn 1
.pn pref 2-
```

for the second chapter, its pages will be numbered 2-1, 2-2, 2-3, and so on.

With SCRIPT/VS, you can produce single-column or multiple-column output pages, or a mixture of both.

DEFINING MULTICOLUMN LAYOUT

SCRIPT/VS can format your output page with up to nine columns of text. To define a multicolumn layout, you should decide how many columns you want, the width of each column, and the desired horizontal position on the page for the left margin of each column.

The space between columns (the gutter) is determined by the relationship of the column width to the column positions. Usually the column width will be a value that is less than the difference between the left margin positions of adjacent columns, ensuring that some space will be present between columns.

Once you have decided the dimensions and positions of your columns, the column definition can be specified using the following SCRIPT/VS control words:

- .CD [Column Definition], which provides for
  - Specifying the number of columns
  - Specifying the left margin position for each column
- .CL [Column Width] which provides for
  - Specifying the column width for all columns

To define a multicolumn layout for three columns that have widths of 18M, and have left margins at the page's left margin, at 24M, and at 52M respectively, the following control words would be used:

```
.cl 18m
.cd 3 0 24m 52m
```

and would produce this effect:

```
0                 18m   24m               42m       52m               70m
|                 |     |                 |         |                 |
v                 v     v                 v         v                 v
As  you  can  see,      width) is 18M. The          two  and  three is
the  column  defi-      space between col-          greater  than that
nition has changed      umn one and column          between    columns
and  we  are   now      two  is 6M  (24M -          one  and  two. All
formatting    with      18M). Column two's          columns  have  the
three columns. The      right  margin   is          same  width. It is
first     column's      42M (24M  +  18M).          not    necessarily
left  margin is at      The third column's          desirable  to vary
the left margin of      left  margin is at          the  gutter  space
the page (position      position  52M. The          but   this    does
0).    The  second      space between col-          illustrate     the
column's left mar-      umn two and column          flexibility of the
gin is at position      three  is 10M (52M          .CD  [Column Defi-
24M.  Column one's      -  42M). As can be          nition]  and   .CL
right  margin (po-      seen,   the  space          [Column     Width]
sition   +  column      between    columns          control words.

                  <—6m—>                  <——10m——>
```

The preceding example shows one multicolumn layout. There are many possible variations.

The .LL [Line Length] control word is used to specify the line length for running headings and footings, top and bottom titles, page floats, and footnotes. Normally this value is set equal to the right margin of the rightmost column to align all the components of the page. In the preceding example you would specify:

```
.ll 70m
```

The following control words specify text that is formatted using line length (.LL) instead of the column width (.CL):

- .RH [Running Heading]
- .RF [Running Footing]
- .RT [Running Title]
- .FL [Float] PAGE
- .FN [Footnote]

Notes:

- The .CD [Column Definition] and .CL [Column Width] control words take effect immediately on the next output line.
- "Appendix E. Formatting Considerations for the 3800 Printer" on page 385 contains additional considerations regarding the definition of multicolumn layout for the 3800 Printer.
- Columns may be specified in any order, and the gutter space (the space between columns) may vary, but if text from any column extends into the next column, the text in all subsequent columns will be displaced to the right.

Page Sections and Section Breaks

A page is divided into "sections" that may be thought of as independent components. These sections are:

Top Title

Running Heading

Top Page Float

Body Text

Bottom Page Float

Footnotes

Running Footing

Bottom Title

Once a page section is completely formatted and its columns balanced, it cannot be changed. This is called a "section break." When all page sections are complete, the page is written to the output destination.

See Figure 8 on page 56 for a pictorial representation of the page and its component parts.

The "column depth" for each column on the page is equal to the page length minus the space reserved for the top and bottom margins, running headings and footings, and footnotes, if any. See "Chapter 4. Defining a Page Layout" on page 55 for details on these component space values.

When formatting a page, completed output lines are placed in the current column until it is full. The lines formatted for the current column are saved and a new column is begun. This is called a "column eject."

If all columns on the page are full, a new page is begun. This is called a "page eject."

A section break occurs when:

- All columns on the page are full
- A page eject is requested by:

  .PA [Page Eject]

  .CP [Conditional Page Eject]

  .CB [Column Begin] in the last column

  .CC [Conditional Column Begin] in the last column
- The column definition is changed by:

  .CD [Column Definition]
- The column mode is changed by:

  .MC [Multicolumn Mode]

  .SC [Single Column Mode]
- A full page skip or space is requested by:

  .SK [Skip] with the "P" parameter

  .SP [Space] with the "P" parameter

When a section break occurs, the lines that have been formatted for this section are redistributed as equally as possible among the defined columns. This is called "column balancing." This process is not performed if there is only one column, or if column balancing has been disabled by the .BC [Balance Columns] control word.

If the column definition is changed in the middle of the page, all lines formatted to that point are processed and sent to the output destination. A new output section is started using the new column definition. The depth of the new columns is equal to the space remaining on the page above the running footing and bottom margin.

Column Positions

Column positions remain in effect until explicitly changed by a .CD [Column Definition] control word. For example, you can define a multicolumn layout and then format using one or more columns without changing the column positions.

This first section was produced by specifying

```
   .cd 1 0 22m 44m
   .cl 18m
```

to format using only the first column. Notice that the second and third columns are empty, even though their positions have been defined.

This second section was produced by specifying

```
   .cd 2
```

to format using the first two columns. The original column width is used for all columns. Notice that the formatted lines are distributed between columns one and two.

This third section was produced by specifying

```
   .cd 3
```

to format using all three columns. As can be seen from this example, the number of columns may be varied without changing the column position values. Notice that the formatted lines are distributed among all three columns. If the lines cannot be equally divided, some columns may be longer than others.

Column Width

Column width remains in effect until explicitly changed by a .CL [Column Width] control word. The formatter attempts to build each output line to fill the column width by concatenating short input lines or folding long input lines. A partially full output line is padded to full width by justification. Line concatenation and justification are controlled by the .FO [Format Mode], .CE [Center], and .RI [Right Adjust] control words. For details, see "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219.

If an input line contains a word that is longer than column width, the portion of the word that overflows the column is processed according to the TRUNC, FOLD, or EXTEND option of the .FO [Format Mode] control word. Likewise, if concatenation is off, and the input line is longer than column width, the excess portion of the line is processed based on that specification.

If there is more than one column, and the document is being formatted for the 3800 Printer, excess characters in column one will cause dislocations in all subsequent columns on an extended line. For this reason, the EXTEND option is not recommended when formatting multicolumn output for the 3800 Printer.

Column width is normally changed along with column positions to maximize use of the space on the page when the number of columns changes. Normally the column width value would be set to line length minus all gutter space, divided by the number of columns.

With a line length of 68M, and a gutter of 4m, two columns would be defined as:

```
  .cd 2 0 36m
  .cl 32m
```

This two-column data is formatted with a column width of 32M to maximize the use of the space on the page. As can be seen, there is little wasted. This example is meant to show typical usage. Normally columns will be laid out to be as dense as possible for economic page use. Readability is also a factor in column definition.

With the same line length and gutter size, three columns would be defined as:

```
  .cd 3 0 24m 48m
  .cl 20m
```

This three-column data is formatted with a column width of 20M to maximize the use of the space on the page. As can be seen, there is little wasted. This example is meant to show typical usage. Normally columns will be laid out to be as dense as possible for economic page use. Readability is also a factor in column definition. In this three-column example the columns are a little narrow.

Starting a New Column

The following SCRIPT/VS control words may be used to end a column before it is full:

- .CB [Column Begin] ends the column unconditionally.
- .CC [Conditional Column Begin] ends the column based on the space remaining in the column.
- .CP [Conditional Page Eject] ends the column and causes a page eject based on the space remaining in the column.

Use the .BC [Balance Columns] control word to enable or disable column balancing. If column balancing is OFF, no columns are balanced. If column balancing is ON, each set of columns is balanced whenever a section break occurs.

Blocks of text, such as figures or tables, can be kept together and balanced as a unit. Text lines in a block will not be split across columns. See "Chapter 8. Additional Formatting Features of SCRIPT/VS" on page 87 for details on use of the .KP [Keep] control word.

If the current column is forced to end before it is full, the new column is ineligible for column balancing.

SUSPENDING AND RESUMING MULTICOLUMN PROCESSING

The .SC [Single Column Mode] control word

- Saves the current column definition
  - Column width
  - Number of columns
  - Column positions
- Defines a single column with a column width equal to line length.

The .MC [Multicolumn Mode] control word

- Restores the last-saved column definition

The .SC [Single Column Mode] and .MC [Multicolumn Mode] control words are always paired; you must specify the .SC control word before you specify the .MC control word.

SCRIPT/VS provides an automatic table of contents facility which is based on the concept of "head levels." When you create a SCRIPT/VS file, you can enter topic headings[23] to designate changes in content, or to create titles.

The format of a topic heading indicates its relationship to the other topic headings in the document. In SCRIPT/VS, different levels of headings can be entered with the control words .H0, .H1, .H2, .H3, .H4, .H5 and .H6[24]. When SCRIPT/VS processes a .H0 - .H6 [Head Level 0 - 6] control word:

- The text portion of the heading is formatted according to characteristics associated with the head level. The formatting may include such things as spacing above and below the heading, capitalization, underscoring, and font.
- If the heading requires a table of contents entry, the heading's text and current page number are saved in a temporary file called DSMUTTOC.

For example, if you enter a topic heading as

```
.h3 Symptoms
```

SCRIPT/VS uses the characteristics for a level-three heading to format the heading's text on the page. SCRIPT/VS also creates an entry in the table of contents file for the topic "Symptoms" and the page number on which it appears. All the headings entered with the .H3 control word are formatted in the same way.

If you use SCRIPT/VS head-level control words exclusively, you need not create a table of contents manually. When you revise or reorganize your document, the table of contents is automatically updated.

CHARACTERISTICS OF HEAD LEVELS

Head levels are commonly associated with the following sections of a document:

.H0	Table of Contents entry only
.H1	Chapter
.H2	Major section
.H3	Minor section
.H4	Topic
.H5	Inline heading
.H6	Inline heading

The .DH [Define Head Level] control word allows you to redefine the characteristics of any head level to suit your needs. The characteristics are:

- Whether the heading in the text should begin on a new page or cause a break.
- Whether the heading should be placed in a separate section.
- Whether the heading should be numbered with a decimal number associated with the head level.

---

[23] The word "heading" is used in this section to mean a topic heading that is printed as part of the text.

[24] GML and EasySCRIPT provide tags with similar names and functions. This discussion is concerned only with the SCRIPT/VS control words.

	.H0	.H1	.H2	.H3	.H4	.H5	.H6
Page eject before heading		yes					
Section breaks around heading		yes					
Heading Out-justified [1]		yes					
Line skips before heading	0	0	3	3	3	1	1
Line spaces after heading	0	5	2	2	2	0	0
Heading underscored		yes	yes		yes	yes	yes
Heading capitalized		yes	yes	yes		yes	
Heading will cause a break		yes	yes	yes	yes		
Table of Contents entry	yes	yes	yes	yes			
Table of Contents entry only	yes						
Skip before T.O.C. entry		yes					
Table of Contents indention	0	0	0	2	4	6	8

[1] The heading will be right-justified on the page if the page number is odd.

Figure 9. Summary of Default Head Level Characteristics: This table lists the default characteristics of the .Hn [Head Level n] control words. The .DH [Define Head Level] control word allows you to redefine any of these characteristics to suit your needs. Note that, by default, all headings and table of contents entries are printed in the current font.

- Whether the heading should be right-aligned if it occurs on an odd page.
- Whether the heading should be capitalized or underscored, and the font it is to be printed in.
- The amount of vertical space which precedes and follows the heading.
- Whether or not a table of contents entry is to be created. If so, other characteristics for the table of contents entry which can be specified are:
  - The indention of the entry in the table of contents
  - The font to be used for the entry in the table of contents
  - Whether the entry is to be preceded by a skip in the table of contents
  - Whether to right-align the page number associated with the entry, separated from the text by a "dot-leader"
  - Whether _only_ a table of contents entry should be created, placing no heading at all in the text

Figure 9 lists the default characteristics of the .H0 - .H6 [Head Level 0 - 6] control words.

Spacing and Page Ejects

Headings are printed in the current column when there is enough room for the heading and at least two lines of text that follow it in the body of the document. If there is not enough room, the heading is placed at the top of the next column.

The line spaces that follow topic headings are conditional. If the heading is followed by more vertical space (whether caused by the .SP [Space] or .SK [Skip] control words, or another head level), only the larger of the two spaces is used, not the sum. If the heading causes a section break, then both spaces will be used.

Head levels that are defined to begin new pages cause page ejects only if SCRIPT/VS is not already at the top of a page. This can be useful:

- To assign a page number to the output page with a .PA [Page Eject] control word.
- To eject to a new even- or odd-numbered page with the .PA EVEN or .PA ODD control words.

Defining Head Levels

The .DH [Define Head Level] control word allows you to redefine the characteristics of any head level. The .DH control word accepts parameters that describe head level characteristics, such as SPAF (SPace AFter) to set the amount of vertical space to follow the heading, and TC to indicate that a table of contents entry is to be generated. For example,

```
.dh 3 skbf 1 us
```

will redefine the .H3 head level to provide only one line of space before the heading, and to underscore the heading. The .DH control word is described in ".DH [Define Head Level]" on page 247.

You can also redefine a .H0 - .H6 [Head Level 0 - 6] control word using macros to provide an entirely different function for an existing head level.[25] Use the .DM [Define Macro] control word to define a macro with the name of the head level control word.

THE TABLE OF CONTENTS

When SCRIPT/VS processes a head-level control word that requires a table of contents entry, it writes an entry in the DSMUTTOC file. The entry contains the following information:

- A fixed-length field containing information about the font, indention, current revision code, and so forth, to be used for formatting this table of contents entry.
- The text of the heading.
- The page number of the page on which the heading appears.

All entries in the table of contents file are inserted into DSMUTTOC by .PT [Put Table of Contents] control words.

The automatic underscoring and capitalization provided for topic headings do not apply to the associated table of contents entry.

---

[25] SCRIPT/VS Release 2 does not use macros to process the .H0 - .H6 [Head Level 0 - 6] control words. SCRIPT/VS Release 1 created macros named DSMSTDH0 through DSMSTDH6 to process the .H0 through .H6 control words, and macros named DSMEZSH0 through DSMEZSH6 to process the EasySCRIPT head levels. If you require these macros, you can write a .DH macro that will produce macros for all the head-level control words in the same way that they were implemented in Release 1 of SCRIPT/VS. The file DSMSTDH, which is provided with SCRIPT/VS, shows an example of how this can be done.

Therefore, enter the text of a topic heading as it should appear in the table of contents.

Adding Lines to the Table of Contents

You can place lines directly into the table of contents with the .PT [Put Table of Contents] control word.

The .PT [Put Table of Contents] control word causes the text line to be written into the file DSMUTTOC along with the current page number as a .SX [Split Text] control word. For example, the input line:

```
.pt Sail and Rudder
```

will cause the following control word to be written into DSMUTTOC:

```
.'SX /Sail and Rudder/ ./76/
```

When the DSMUTTOC file's input lines are processed, the line appears in the table of contents as:

```
Sail and Rudder  . . . . . . . . . . . . . . . . . . . . .  76
```

You can insert any SCRIPT/VS control word into the table of contents with the .PT control word. If the "text line" part of the .PT control word begins with a period (with only one blank between .PT and the text line), SCRIPT/VS inserts it directly into the DSMUTTOC as a control word, rather than as the text of a .SX [Split Text] control word. For example,

```
.pt .h3
```

inserts a .H3 control word into the table of contents.

If the line of text you want to enter into the table of contents begins with a period, begin the line with a leading blank so that SCRIPT/VS will not interpret the line as a control word, but will include the page number with the line in the table of contents. For example,

```
.pt    .h3
```

inserts the control word

```
.'SX F /.h3/ ./76/
```

into the table of contents.

Printing the Table of Contents

Use the .TC [Table of Contents] control word to imbed the DSMUTTOC file. When .TC is encountered, SCRIPT/VS:

- Ejects to a new page, if it is not already at the top of a page.
- Prints the word "CONTENTS", unless otherwise specified with the .TC control word.

  If you want a different title for the table of contents page, you can specify it as

  ```
  .tc Table of Contents
  ```

  If you don't want a title at all, specify

  ```
  .tc /
  ```

- Formats the DSMUTTOC file according to the SCRIPT/VS environment in effect when the .TC control word is processed, as modified by formatting controls inserted in the DSMUTTOC file.

The table of contents will contain all the entries made prior to the .TC control word during the current or previous pass.

The DSMUTTOC file is not deleted until the next time a new table of contents is started.

TWOPASS Considerations

If you place the .TC [Table of Contents] control word at the beginning of your input file, you must use the TWOPASS option of the SCRIPT command to produce a complete table of contents. Otherwise, the DSMUTTOC file will be empty when the .TC control word is encountered. For details, refer to "TWOPASS: Prepare the Document With Two Formatting Passes" on page 31.

In order to have correct page numbers in the table of contents, pages must be numbered the same way on both passes. On the first pass, the table of contents is empty. On the second pass, it can contain several pages of information. Because SCRIPT/VS doesn't know how many pages will be required for the table of contents, it numbers the pages following the table of contents the same way on both passes.

You can tell SCRIPT/VS the number of page numbers to reserve for the table of contents. For example, you can reserve six pages if the table of contents is to occupy pages 3 through 8. The page number range you reserve has nothing to do with how many actual pages the table of contents will occupy: it only establishes the page number of the page that follows the table of contents page.

For example, if the table of contents will require three pages, you can reserve the current page number and the next two page numbers by specifying:

```
.tc 3 Table of Contents
```

If the document is formatted with the TWOPASS option, SCRIPT/VS will allow page numbering to continue sequentially following the table of contents if the page number is explicitly reset with a .PA [Page Eject] or .PN [Page Numbering Mode] control word before any head level or .PT [Put Table of Contents] control word is encountered that requires knowledge of the page number.

You can precede the .TC [Table of Contents] control word with other SCRIPT/VS control words:

- To number table of contents pages with roman numerals, use the .PN [Page Numbering Mode] control word:

  ```
  .pn roman
  ```

- To put bottom titles on each table of contents page, use the .RT [Running Title] control word:

  ```
  .rt bottom even /Contents &///
  .rt bottom odd  ///Contents &/
  ```

- To ensure that the first page of the table of contents starts on an odd-numbered page, use the .PA [Page Eject] control word:

  ```
  .pa odd
  ```

Note: Since the .TC [Table of Contents] control word has a level-one heading built into it, you should never (whether you specify one or two formatting passes) put a .DH [Define Head Level] level one heading before a .TC [Table of Contents] entry.

CHAPTER 7. INDEXING

SCRIPT/VS enables you to automatically produce an index, such as the one contained in this publication. Include the INDEX option when you issue the SCRIPT command to indicate to SCRIPT/VS that an index is to be generated from the information provided by the .PI [Put Index] control words. The .PI [Put Index] control words are used to specify the index entries and are placed throughout a document wherever the index entry topics are described.

This automatically generated index can contain multilevel entries and cross references. SCRIPT/VS will automatically generate the page numbers for the index entries based on the location of .PI control words within the document. For example, specifying

```
.pi /weasels/
```

indicates that the term "weasels" should be placed in the index along with a reference to the current page number.

POSITIONING THE INDEX IN A DOCUMENT

Use the .IX [Index] control word to indicate where you want the index to be placed in the document. When SCRIPT/VS encounters this control word, it starts a new page, if it is not already at the top of a page, and prints the word "INDEX" as a heading. If you want a different title, you can include one when you specify the .IX control word. For example, you might specify

```
.ix Subject Index
```

which indicates that "Subject Index" is to be used for the title instead of "INDEX."

If you do not want any title at all, specify a single slash (/). For example, specifying

```
.ix /
```

causes SCRIPT/VS to generate the index without a title.

CREATING INDEX ENTRIES

The first nonblank character that follows the .PI control word delimits the beginning of an index term; the second occurrence of that character delimits the end of that term. Any nonblank character that does not appear in the index term can be used as a delimiter. The trailing delimiter does not have to be specified if no other text follows the index term on the input line. For example, specifying

```
.pi ?SCRIPT/VS
```

will place the term "SCRIPT/VS" in the index along with a reference to the current page number.

Regardless of the order in which they are specified within a document, all index entries are placed in alphabetical order before the index is formatted. For example, if you specify

```
.pi /martens
.pi /marsupials
```

"marsupials" will appear in the index before "martens."

Page References

If the same index term is specified several times within a document, that term will only be included in the index once. The page numbers for each occurrence of that term will be listed after it, separated by commas. For example, several occurrences of the term "melodrama" will be formatted in the index as

```
melodrama  9, 14, 37
```

If one particular instance of an index term is more important than the others, you can use the ORDER parameter of the .PI [Put Index] control word to indicate that that page reference is to be listed first, regardless of where in the document other references occur. For example, if the second reference to "melodrama" refers to the principal discussion of the subject, and the others are just passing references, the second instance can be specified as

```
.pi order /melodrama
```

The entry will be formatted as

```
melodrama  14, 9, 37
```

If the discussion of a topic is lengthy, you can indicate the range of pages the discussion spans. The START and END parameters of the .PI control word can be used to mark the beginning and end of the topic. For example, the principal discussion of "melodrama" can be preceded by

```
.pi start /melodrama
```

and succeeded by

```
.pi end /melodrama
```

The entry will be formatted as

```
melodrama  9, 14-16, 37
```

Multilevel Entries

When there are many references to a particular index term, you may want to further qualify the term with another level of indexing. Up to three levels of terms can be specified for an index entry using the .PI control word. SCRIPT/VS collects all index entries with the same first-level term and sorts the second-level terms alphabetically. These second-level terms are then placed in the index immediately following the first-level term to which they apply. Similarly, all index entries with the same first- and second-level terms are collected together and formatted alphabetically. These third-level terms are then placed in the index immediately following the second level-term to which they apply. For example, a description of weasels may contain the following:

```
.pi /weasels/training
   ...
.pi /weasels/care and feeding
   ...
.pi /weasels/breeding
```

The index entry for "weasel" would look like this:

```
weasels
   breeding  22
   care and feeding  15
   training  14
```

Similarly, the section concerned with the care and feeding of weasels might contain these entries:

```
.pi /weasels/care and feeding/exercise
   ...
.pi /weasels/care and feeding/nutrition
   ...
.pi /weasels/care and feeding/dental hygiene
```

The index entry will then appear as

```
weasels
   breeding  22
   care and feeding  15
      dental hygiene  19
      exercise  15
      nutrition  16
   training  14
```

Explicitly Specified Page Numbers

When four index terms are specified with a .PI [Put Index] control word, the fourth term is not interpreted as a fourth-level term; it is used in place of the current page number in formatting the index entry. For example, to indicate that a Japanese Black Pine is illustrated in the fourth folio that is being attached, you can specify

```
.pi /Pines/Japanese Black//Folio 4
```

The text "Folio 4" will be formatted in the index along with the page numbers of the other occurrences of these terms. This index entry will be formatted as

```
Pines
   Japanese Black  19, 22, Folio 4
```

Cross-References

When a document discusses a great variety of topics, you may want to include cross-references to related topics. The REF parameter of the .PI [Put Index] control word can be used for this purpose. This parameter indicates that the last term specified is a cross-reference to another index entry.[26] Rather than suffixing the term with the current page number, SCRIPT/VS will prefix it with "See" or "See also" depending on whether or not there are any non-reference terms of the same level. For example, if the document describing weasels also contained a general discussion of burrowing mammals, you might want to specify

```
.pi ref /weasels/martens
```

which would make the index entry appear as

```
weasels
   See also martens
   breeding  22
   care and feeding  15
      dental hygiene  19
      exercise  15
      nutrition  16
   training  14
```

Index cross references can also be used to direct readers from variations on a term to the principal index entry for that term. For example, specifying

```
.pi ref /circular definition/definition, circular
```

will create the following index entry:

```
circular definition
   See definition, circular
```

Since there are no nonreference terms under the entry "circular definition," the cross reference is prefixed only with "See".

[26] The REF parameter is valid only if at least two terms are specified.

SORTING INDEX ENTRIES

SCRIPT/VS collects index entries as they are specified throughout the document and sorts them alphabetically. Included for each index term are the text of that term, and a sort key. The sort key is used to determine where each entry is placed in the index, and to group multiple occurrences of the same entry.

The sort key for an index term is created by folding the text of the index term to uppercase. Therefore, many different index terms can result in the same sort key. SCRIPT/VS considers index terms with the same sort key to be multiple occurrences of the same term, and formats them as one term with multiple page references. Thus the index terms

```
.pi /Walrus
   ...
.pi /walrus
   ...
.pi /WALRUS
```

all have the same sort key, WALRUS, and will be recognized as three occurrences of the same index term.

The text of the term printed in the index is that of the first occurrence specified with the .PI [Put Index] control word. Subsequent occurrences contribute only additional page references. Therefore, the index entry for the preceding three terms will be formatted as

```
Walrus  4, 7, 22
```

The .PI control word also enables you to specify the text of an index term independently of how it appears in the document. When the page number for an index term is explicitly specified, but null, the term becomes part of the index without any page number associated with it. Subsequent occurrences of that term contribute page number references, but the text of the term is that of the first occurrence. For example, if the profile for the document containing the preceding terms contains

```
.pi /walrus////
```

The index entry will appear as

```
walrus  4, 7, 22
```

The text of the index entry is taken from the first occurrence of the term and the page numbers of the three subsequent occurrences of that term.

Handling Special Characters

Index terms often contain special characters which, even though they are part of the term, should not be considered when the term is being alphabetized. For example,

```
.pi /"The Walrus & the Carpenter"
```

will, by default, be placed at the beginning of the index since the double quotation mark (") appears near the beginning of the alphabetizing sequence. You can use the IXI parameter of the .DC [Define Character] control word to indicate to SCRIPT/VS that certain characters are to be ignored when they appear in an index term. For example, if you specify

```
.dc ixi "
```

the preceding index term will be placed in the "T" section of the index, rather than at the beginning, since the double quotation marks will be ignored when the index terms are sorted. Similarly, the terms

```
.pi /Olduvai Gorge
.pi /O'Leary/
```

will, by default, be formatted in the index as

```
O'Leary  39
Olduvai Gorge  22
```

However, if you specify

```
.dc ixi '
```

before specifying these terms, the apostrophe in "O'Leary" will be ignored during sorting and the index entries will be formatted as

```
Olduvai Gorge  22
O'Leary  39
```

It is even possible to have SCRIPT/VS ignore blanks in index terms when sorting them. For example, the terms

```
.pi /Waterford
.pi /water wheel
```

will, by default, be formatted as

```
water wheel  12
Waterford  7
```

since the blank preceeds "f" in the alphabetizing sequence.

If you specify

```
.dc ixi 40
```

blanks will be ignored during sorting and the entries will be formatted as

```
Waterford  7
water wheel  12
```

There may also be occasions when you want SCRIPT/VS to treat some special characters as though they were blanks when they appear in an index term. You can use the IXB parameter of the .DC [Define Character] control word to do this. For example, the terms

```
.pi /second-class mail
.pi /second division
```

will, by default, be formatted as

```
second division  32
second-class mail  29
```

since the blank character precedes the hyphen in the alphabetizing sequence. However, if you specify

```
.dc ixb -
```

before specifying the preceding index terms, the hyphens will be treated as blanks during the sorting process, and the entries will be formatted as

```
second-class mail  29
second division  32
```

Explicitly Specifying Sort Keys

Occasionally, you may want to place an index entry in some section of the index independent of the entry's normal sort key. You can use the KEY parameter of the .PI [Put Index] control word to explicitly specify the sort key that is to be used for an index term. The KEY parameter is specified as

```
    .pi key /key1/key2/key3/ /term1/term2/term3
```

where key1, key2, and key3 are the new sort keys that are to be used and term1, term2, and term3 are the index terms that they apply to. The keys are separated by a delimiter which can be any nonblank character that does not appear in any of the keys. All four delimiters must be specified even if only one key is being specified and the other keys are null. For example, to place the term "IBM 3800" at the end of the index in the "3" section, specify

```
    .pi key /3800/// /IBM 3800
```

For a multilevel index entry, explicitly specified keys can be specified separately for each level. When the specified key is null, the sort key is developed according to normal SCRIPT/VS sort key processing, as described earlier in this section. For example, if you specify

```
    .pi key /HUNGARY/// /Austria-Hungary/Domestic Policy
```

SCRIPT/VS will place the term "Austria-Hungary" in the "H" section of the index, using "HUNGARY" as the sort key. The key for the second-level term, "Domestic Policy", is developed using normal SCRIPT/VS key processing, since the explicit sort key specified is null.

Using the KEY parameter of the .PI control word, you can make the text of an index term completely unrelated to the actual term. For example, if you specified

```
    .pi key /WALRUS/// /"The Walrus and The Carpenter"
```

all entries specified for walrus would be formatted in the "W" section as

```
    "The Walrus and The Carpenter"  4, 7, 22
```

instead of as

```
    walrus  4, 7, 22
```

provided that the entry with the KEY parameter was the first entry.

<u>CREATING THE INDEX</u>

When you specify the .IX [Index] control word, SCRIPT/VS formats the index by creating and executing .IE [Index Entry] control words for each index entry. The first parameter of the .IE control word indicates the index entry level. For example, the control words

```
    .pi /Pines/Lodgepole
    .pi /Pines/Japanese Black
```

will generate the first-level term, "Pines", with two second-level terms, "Japanese Black" and "Lodgepole".

When you specify .IX, SCRIPT/VS will format these terms by creating and executing the following control words:[27]

---

[27] The blank between the control word name and the first parameter does not have to be specified, as described in "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219. It is omitted in the .IE [Index Entry] control words generated by .IX [Index].

```
.IE1 Pines
.IE2 Japanese Black  9
.IE2 Lodge pole  5
```

The .IE [Index Entry] control word causes a break and sets an indention based on the first parameter. This makes the formatted entry appear as

```
Pines
   Japanese Black  9
   Lodge pole  5
```

The .IX [Index] control word precedes each section of the index with an .IE Header control word. For example, the "P" header that precedes the section containing terms beginning with "P" is generated using

```
.IEH P
```

This header control word causes SCRIPT/VS to skip two lines, print the specified section letter, and then skip another line before formatting the first index entry for that section.[28] The .IX control word generates and executes a header for each section of the index for which there are entries.

[28] Since SCRIPT/VS omits the optional blank between the control word name and the first parameter, you can write macros to provide more elaborate formatting for some index entries without interfering with other terms. For example, the following macro will draw a box around each index header:

```
.dm ieh /.sk 2 /.bx 1 5 /  &*/.bx off /.sk
```

The formatting of first-, second-, and third-level entries is not affected by this macro.

In addition to formatting your document as described in previous chapters, SCRIPT/VS allows you to:

- Define character mappings, or translations, to be performed at various times during the processing of input and output lines
- Establish characters with special meaning for SCRIPT/VS
- Keep blocks of text together so that, on output, the kept-together material appears entirely in one column
- Define footnotes to be placed at the bottom of the page
- Mark all updated material in a review draft, so readers can identify the modified material
- Draw rectangular boxes with horizontal and vertical interior lines
- Treat underscoring and capitalization as fonts, and overstrike text (on impact printers) to produce bold and strikeout text

CHARACTER MANIPULATION

SCRIPT/VS performs several character translations on input and output lines as part of its normal processing. You can define the specific character mappings each of these translations peforms for such purposes as:

- Printing characters that are available on your output device but not on your terminal
- Simulating control characters not available on your terminal
- Pairing the upper- and lowercase letters of various national languages, such as German
- Expanding individual input characters into character strings

Output Character Translation

If you are using a terminal with a standard keyboard, you may not have an immediate way to enter special characters in a SCRIPT/VS file. You may not, for example, be able to directly enter a bullet (•) from the keyboard. When you print SCRIPT/VS output, you may want to use a bullet and other special characters as well.

One way to enter special characters into a file is to use appropriate commands while editing.

SCRIPT/VS provides another method for printing special characters. You can specify that one of your keyboard characters be translated to the special character, using the .TR [Translate Character] control word.[29] For example,

```
.tr * af
```

Each occurrence of an asterisk in your file is translated, on output, to the bullet character (•) which has the hexadecimal code AF. For example, the input line

[29] Hexadecimal 27 is reserved for internal use by SCRIPT/VS, and should not be used with the .TR [Translate Character] control word.

```
    * Pay attention to this point.
```

results in:

    • Pay attention to this point.

"Appendix D. Fonts Supplied with SCRIPT/VS" on page 377 and Figure 38 on page 364 illustrate the various character sets available and their hexadecimal codes. You can use these charts when you want to translate characters such as:

- Brackets: [ ]
- Braces: { }
- Algebraic and logical symbols: < ≤ ≠ = ≥ > | ¬ ±
- Superscript numerals: ^{0} ^{1} ^{2} ^{3} ^{4} ^{5} ^{6} ^{7} ^{8} ^{9}
- Bullets: • ▪
- Box characters: ┌ ┐ └ ┘ ─ │ ┬ ┴ ┼ ├ ┤

You can specify as many translation pairs with one .TR [Translate Character] control word as your input line allows. For example,

```
    .tr  a AC  b BC  c BB  d AB  - BF  | FA
```

specifies the corners, vertical bar, and dash[30] used for drawing a box. Each special character is specified as its character code: hexadecimal AC is the code for "┌", hexadecimal BC is the code for "┐", and so on. When the characters "a", "b", "c", "d", "|", and "-" appear in a subsequent output line, they are replaced with the special characters' hexadecimal codes. For example, the input lines

```
    a----------b
    |          |
    |          |
    |          |
    d----------c
```

result in

```
    ┌──────────┐
    │          │
    │          │
    │          │
    └──────────┘
```

To cancel translation of all previously specified character mappings, use the .TR [Translate Character] control word with no parameters:

```
    .tr
```

When you have many character mappings specified, you can reassign or cancel some of them without affecting the others. For example,

```
    .tr ( (
```

cancels translation of the left parenthesis to any character established for it. Actually, this is equivalent to setting up a new mapping for "(": the character is to be translated to itself.

---

[30] When you use hyphens to draw boxes, you can translate them to the extended dash (hexadecimal BF), which aligns with the corner symbols and extends further than a hyphen to create an uninterrupted line. Similarly, the character code FA provides a vertical bar that is longer than the one available on most keyboards which abutts with the corner characters.

Note: While an output character mapping is in effect, every occurrence of the affected character is translated to the designated output character. You should therefore take care to translate only characters that will not be needed during that time.

Output translation is performed during formatting just before the characters' widths are measured for justification.

If you have used the .TR [Translate Character] control word and direct the SCRIPT/VS output to your terminal, some of the special characters may not be displayed in the output. The positions occupied by the translated characters may appear as blanks, because there are no equivalent characters on the terminal. You can use the .IF [If] control word to make character translations conditional based on the output device:

```
.if SYSOUT eq PRINT .tr * af
```

This control word line results in output translation of asterisks (*) only if output is going to the printer. The .IF [If] control word is discussed in detail in "Chapter 10. Conditional Processing" on page 111.

Input Character Translation

SCRIPT/VS also performs character translation on input lines. The .TI [Translate Input] control word allows you to make characters that are unavailable on your terminal effectively part of your input file.[31] For example, the IBM 3270 terminal does not have a tab key. However, an available character, such as the not-sign ("¬"), can be translated to hexadecimal 05, the tab character code:

```
.ti ¬ 05
```

While the translation is in effect, any not-sign (¬) on an input line is processed as though it were a tab. Because the translation occurs first, before any other processing, you should take care when using the .TI control word:

- Use hexadecimal codes for the special character rather than the character itself. For example,

  ```
  .ti % $
  ```

  translates all occurrences of % to $. However, you cannot restore the percent-sign character by subsequently issuing

  ```
  .ti % %
  ```

  because that input line is translated to ".ti $ $" before being processed. However, you can restore % to itself with

  ```
  .ti 6C 6C
  ```

  Be careful, though. Remember that <u>each</u> character on the input line is translated (if a translation for it exists) before processing the input line. If you translate 0 (hexadecimal F0) to @ (hexadecimal 7C), for example, with

  ```
  .ti F0 7C
  ```

  you cannot restore the 0 to its original definition by issuing

  ```
  .ti F0 F0
  ```

  because each "F0" in the above control word would be translated to "F@" before the control word is processed.

[31] Hexadecimal 27 is reserved for use by SCRIPT/VS, and should not be used with the .TI [Translate Input] control word.

- Be careful when you translate a symbol that has special meaning for SCRIPT/VS, specifically the period (. or hexadecimal 4B) and the blank (hexadecimal 40). For example,

  ```
  .ti . %
  ```

  translates the period (.) to the percent sign (%). All subsequent SCRIPT/VS control words are ignored because the input characters are translated first, before the line is processed. Control words and macros would be regarded as text because they begin with a percent sign instead of a period.

- To restore all characters to normal, use the .TI [Translate Input] control word with no parameters:

  ```
  .ti
  ```

Uppercase Translation

SCRIPT/VS provides several means of capitalizing text. They are:

- The UPCASE option of the SCRIPT command, described in "UPCASE: Print Lowercase Letters as Uppercase" on page 32
- The .UP [Uppercase] and .UC [Underscore and Capitalize] control words, described in "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219
- The .H0 - .H6 [Head Level 0 - 6] control words, if capitalization is specified with the .DH [Define Head Level] control word
- The .DF [Define Font] and .BF [Begin Font] control words, described later in this chapter
- The &U' symbol attribute, described in "Chapter 12. Symbols in Your Document" on page 129

By default, SCRIPT/VS capitalizes text by translating the letters a through z to A through Z. This translation can be extended for languages other than English with the .TU [Translate Uppercase] control word. For example,

```
.tu  8a ca  9a da  aa ea
```

would add capitalization pairs appropriate for German.

Uppercase translation may be reset to its default by entering

```
.tu
```

without any parameters. Note, however, that unlike .TR and .TI, the default for .TU is the mapping of a through z to A through Z.

String Translation

All of the forms of translation discussed above provide "one-to-one" character pairings: Each character is mapped by the translation into another single character. Occasionally, it may be convenient to translate a single character into a string of characters. For example, single asterisks may be expanded into arrows:

```
.ts * /===> /
```

With this translation in effect, the input line

```
*Pay Attention
```

will be formatted as

```
===> Pay Attention
```

The character string that replaces a character may contain both text and control words. For example,

```
.dc cw off
.ts < /;.bf GB12;"/
.ts > /?";.pf;/
.dc cw ;
```

will cause the input line

```
<What, four> bellowed the Mathemagician.
```

to be formatted as

"What, four?" bellowed the Mathemagician.

String translation is actually a form of symbol substitution, and therefore:

- Is only performed when symbol substitution is on. (You can inhibit string translation with the .SU [Substitute Symbol] OFF control word.)
- Is performed at the same time as symbol substitution, just after input translation, but before any other processing.
- Is not subject to further symbol substitution.

String translations are reset somewhat differently from other forms of translation, and special care must be taken to prevent string translation when resetting a character. The first example above may be safely reset by specifying

```
.ts 5c off
```

or

```
.su off
.ts * off
.su on
```

Remember when using .TS that, like .TI, string translations affect <u>all</u> occurrences of the character, and are performed before any other processing of the line.

DEFINING SPECIAL CHARACTERS THAT AFFECT SCRIPT/VS PROCESSING

You can define characters with special meaning to SCRIPT/VS using the .DC [Define Character] control word. The special characters are:

The array element separators, which are placed between elements of a symbol array. See "Chapter 12. Symbols in Your Document" on page 129 for details.

The continuation character, which allows single words to span input lines. The continuation character is described below.

The control word separator, which allows several SCRIPT/VS control words to be "stacked" on a single input line. See "Chapter 3. Basic Text Processing" on page 33 for details.

The GML delimiter and markup content separator, which are used to delimit GML tags and attributes. See "Chapter 14. GML Support in SCRIPT/VS" on page 159 for details.

Indexing characters, which are to be ignored or treated as blanks when preparing index entries. See "Chapter 7. Indexing" on page 79 for details.

The page number symbol, which is replaced with the current page number wherever it appears in running titles, running headings, and running footings. See "Chapter 4. Defining a Page Layout" on page 55 for details.

Punctuation characters, which are recognized during spelling verification. See "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171 for details.

The required blank, which is not recognized as an interword space during justification, but is translated to an ordinary blank on output.

Full stop characters, which indicate the end of a sentence. See "Chapter 3. Basic Text Processing" on page 33 for details.

Word delimiters, which delimit words for purposes of spelling verification. See "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171 for details.

The parameters of the .DC control word are described in detail in ".DC [Define Character]" on page 241.

The Continuation Character

SCRIPT/VS ordinarily appends an interword space to the last word on a text input line. However, if the last character on a text input line is the continuation character, it is removed and the interword space is not appended. The continuation character is defined with the .DC [Define Character] control word:

```
.dc cont +
```

This allows a single word to span text input lines and control words. For example, the input lines

```
A few high+
.bf GB12
light+
.pf
ed characters.
```

will produce this output:

A few high**light**ed characters.

If the formatter control or text which follows the continued word causes a break, continuation is cancelled for that line. The control words that cause breaks are listed in Figure 25 on page 354.

There is no default continuation character; it must be explicitly set before it can be used.

ENSURING THAT BLOCKS OF TEXT STAY TOGETHER

SCRIPT/VS provides several means of keeping lines of text together for such purposes as:

- Ensuring that an example or list of items is not split across a column or page
- Keeping a heading and the first few lines of text below it together
- Preventing widows (single lines at the beginning or end of a paragraph that appear by themselves at the bottom or top of a column or page)
- Placing a figure or diagram at the top or bottom of a column or page

Keeps

When you wish to keep a specific group of lines, such as a figure or example, together, consider using:

- Regular keeps, started with .KP ON, place the kept text in the current column if it will fit. Otherwise, a column eject is performed and the text is placed in the next column.
- Floating keeps, started with .KP FLOAT, save the kept text for the next column if it does not fit in the current column, formats the text that follows the keep in the input file, and places it in the current column.
- Delayed keeps, started with .KP DELAY, are always placed in the next column, whether or not they fit in the current column. As with floating keeps, text following the keep in the input file may be moved ahead of it in the output to fill the current column.

Each of these keeps must be explicitly ended with .KP OFF, and each saves the current formatting environment, including any partially processed output line. The formatting environment is restored when the keep ends. See Figure 34 on page 360 for a list of the formatting parameters saved and restored around keeps.

For example,

```
.kp on
.fo center
These lines will be kept together in the
column, regardless of page ejects and column balancing,
and
the formatted lines will be centered.
.kp off
These lines will not, however, necessarily appear in the
same column
as the lines above, nor will they be centered,
since the formatting mode was restored when the
keep was ended.
```

will be formatted as

```
    These lines will be kept together in the column,
 regardless of page ejects and column balancing, and the
           formatted lines will be centered.
These lines will not, however, necessarily appear in the
same column as the lines above, nor will they be centered,
since the formatting mode was restored when the keep was
ended.
```

If you place a large figure in a regular keep, and it does not fit in the current column, it will be placed in the next column. This may leave a large blank space at the bottom of the current column. If the figure does not have a specific relationship to the text around it, you can avoid the blank space by placing the figure in a floating keep. For example,

```
This paragraph contains a reference
to the figure that follows it.
This text will appear above the figure,
.br
.kp float
   ....
   (drop in figure here)
   ....
.kp off
but this text may appear above or
below the figure, depending upon whether
the figure is moved to the next column.
```

There is an order of precedence among keeps, with regular, floating, and delayed keeps taking precedence over inline keeps. If an inline keep is encountered within a floating keep, it is ignored.

But if a regular keep is encountered within an inline keep, the inline keep is ended and the regular keep begun. Keeps of the same level of precedence end each other. For example,

```
    .kp on
    These lines will be
    kept together in
    one column.
    .kp on
    So will these lines,
    but not necessarily in
    the same column with the
    previous few lines.
    .kp off
```

Note: Some control words are not allowed within keeps, and will force termination of the keep before being processed. This is true regardless of whether the control word is found in the input file, in a tag, or within a macro. In general, these control words alter the page or column definitions; they are listed in Figure 27 on page 355.

Inline Keeps

When you wish to ensure that a certain amount of text is kept together without otherwise disturbing the formatting of that text, use an inline keep. Inline keeps are started with:

- .kp inline
- .kp v
- .kp v + v

where "v" is an amount of vertical space. For example, to ensure that the heading of a table is kept together with the first few items in the table, specify

```
    .fo off
    .kp li
    .ce AMERICAN INVENTORS
    .sp
    Name                      Born   Died
    .sp
    Armstrong, Edwin          1891   1954
    Bell, Alexander           1847   1922
    Bell, Herbert             1890   1970
    Carlson, Chester          1906   1968
    De Forrest, Lee           1874   1961
    ...                       ...    ...
```

Inline keeps that specify an amount of vertical space are automatically ended when that amount of text has been formatted. They may also be ended prematurely with .KP OFF. In either case, no break is performed; the formatting of lines is not affected by the inline keep.

Inline keeps are preferable to conditional column ejects, discussed below, especially when your page layout contains more than one column, because columns that are explicitly started with .CB [Column Begin] or .CC [Conditional Column Begin] are ineligible for balancing. Inline keeps ensure that text is moved to the next column if necessary to keep the text together, yet allow preceding text to be moved into the next column as needed to balance the columns if the page is not filled. See "Chapter 5. Multicolumn Page Layout" on page 67 for more information on column balancing.

Widows

When SCRIPT/VS is concatenating input text,[32] you can request that single output lines at the beginning or end of a paragraph not be left alone at the bottom or top of a column or page. If you specify

```
.wz on
```

subsequent paragraphs will be subject to widow control; if the paragraph spans columns, at least two lines of the paragraph will appear in each column.[33] Widow control can be turned off by specifying

```
.wz off
```

Note: For purposes of widow control, SCRIPT/VS considers paragraphs to be delimited by breaks. The control words which cause breaks are listed in Figure 25 on page 354.

Floats

Figures and tables often do not bear any specific relationship to the text immediately surrounding them. SCRIPT/VS provides a way of setting such text apart from the body of the page by placing it at the top or bottom of a column or page, independent of the body text.

Use the .FL [Float] control word to delimit the lines to be set apart, and to indicate where they should be placed. For example, the input lines

```
.fl on page
.im spunits
.sx //-//
.fl off
```

will place the contents of the file SPUNITS at the top of a subsequent page, separated from the text in the page by a row of dashes. (Figure 1 on page 6 illustrates such a float.)

Floats may be specifically designated for odd- or even-numbered pages. For example,

```
.fl on page even
.im tblleft
.sp 2
.fl on page odd
.im tblright
.sp 2
.fl off
```

will place the contents of the file TBLLEFT and TBLRIGHT at the tops of two subsequent pages.

The intent of the previous example is to produce a double-page-width table on facing pages of a duplexed document. However, if the next page is odd, the right-hand float will be placed first, on the front of a sheet, and the left-hand float will be placed later, on the back of the sheet. When floats bear such a relationship to each other, the ORDER option should be included in the .FL [Float] control word. Ordered floats will be placed in the same order in which they are defined. (Figure 35 on page 361 illustrates such a pair of floats.)

---

[32] Concatenation is controlled by the .FO [Format Mode] control word, as described in "Chapter 3. Basic Text Processing" on page 33.

[33] When widow control is in effect, paragraphs of fewer than four lines will not be split between columns.

When a single chapter of a document does not contain enough pages of text to accommodate all the floats defined within the chapter, some of those floats may appear on the first pages of the next chapter. This can be prevented by specifying

```
    .fl dump
```

before beginning the new chapter. As many extra pages will be added as needed to place all the queued floats within the current chapter.

Note: The same control words that are disallowed within a keep are also disallowed within a float. In general, these control words alter the page or column definitions; they are listed in Figure 27 on page 355.

FOOTNOTES

The .FN [Footnote] control word provides an automatic way to format text so it appears at the bottom of a page as a footnote. SCRIPT/VS determines how many lines currently remain on the page and reserves the space needed for the footnote. The .FN [Footnote] control word is specified as:

```
    .fn on
    ** This line is going to
    appear as a footnote
    on this page.
    .fn off
```

SCRIPT/VS prints a 16-dash line, called a "leader," to separate the body of the page from the footnote. To change the footnote leader, redefine it before the page on which the footnote appears is started:

```
    .fn leader
    .sp
    .tr - BF
    .us ----------------
    .sk
    .fn off
```

Normal Footnote Placement

Since there is no maximum depth for a footnote, once a footnote is started, all text until the next .FN (Footnote) OFF command is included in the footnote unless the footnote is prematurely ended by a disallowed control word.

To keep the footnote and its callout on the same page, you should enter the .FN (Footnote) control word and the footnote input lines immediately after the word or phrase that refers to the footnote (known as the "footnote callout"). If the footnote does not immediately follow a text line (without an intervening break), it will be placed as soon as possible without any attempt being made to keep it associated with a callout line or widow.

A line or widow containing a footnote callout(s) will be placed on the page providing that there is sufficient room for:

- the line or widow, and
- the footnote leader, and
- at least two lines, counting skips and spaces, of the last footnote. If the footnote is only three lines or less in depth, then the entire footnote must fit.

** This line is going to appear as a footnote on this page.

If there is insufficient room on the page for the line or widow, then it and its associated footnote(s) will be moved to the next page. However, if the line or widow is already at the top of a page it will not be moved. In such a case, the line or widow will be placed on the page with as much of the footnote(s) as will fit. The remainder of the footnote(s) will be placed on a subsequent output page(s).

In placing footnotes, SCRIPT/VS will, if necessary, attempt to split footnotes that are four or more lines (including skips and spaces) in depth. If a footnote is split, SCRIPT/VS will keep at least the first two lines of the footnote on one page, and it will keep at least the last two lines of the footnote on another page. For the purposes of splitting, a double spaced footnote line and vertical space generated by a single control word (for example, .SP 3) are considered to be single lines.

When a footnote is split, or can't be placed on a page (for example, the first footnote of two called out by a line is greater than a page), the remainder will be allowed to "float" to the next available page(s).

Whenever a new page is started, footnotes that were allowed to float from previous pages are placed on the new page. Again, any unplaced footnote(s) will be allowed to "float". The .FL (Float) DUMP control word causes SCRIPT/VS to place all floats, including footnotes, before resuming input text processing.

In placing footnotes that were "floated" from previous pages, SCRIPT/VS will attempt to reserve room on the page for any pending output line or widow that has not yet been placed. If that pending line or widow also contains footnote callouts, the line or widow may be further deferred, as necessary, in order to keep footnotes and their callouts on the same page.

Unusual Footnote Placement Conditions

There are certain conditions under which SCRIPT/VS will be unable to satisfy the general guideline of keeping footnote callouts and at least two lines of the last footnote on the same page. Some of these conditions could result if:

- the page depth was very small,
- the footnote leader was very large, or
- one or more footnotes are very large.

The conditions and the actions that will be taken are as follows:

- If the footnote leader is as large or larger than the body depth plus the first line of the first footnote, the footnote will be placed on the page but not the footnote leader.
- If the callout line or widow is at the top of the page and all of the footnotes will not fit, then SCRIPT/VS will cause as many of the footnotes as necessary to "float" to subsequent output pages.
- If the callout line or widow is at the top of the page, then SCRIPT/VS will, if necessary, place only one line of the first footnote on the page.
- If the callout line or widow is at the top of the page, then SCRIPT/VS will, if necessary, split the first footnote even if it is a two or three line footnote (this will cause the first line and/or the last line to be placed by itself on an output page.)

Note: The splitting of small footnotes or the placement of only one line of a footnote will not occur unless the footnote is the first one to be placed on the page. If at least one complete footnote is placed on the page, then SCRIPT/VS will only attempt to split the other footnotes if they are four or more lines deep and the first two lines and the last two lines will be kept together.

Other Footnote Considerations

You can mark up the footnote with GML tags, control words, macros, and text just as you can the material within a keep. For example, to provide special formatting within a footnote you could enter:

```
.fn on
.tr 2 B2
.in 2 after 1
2 This is the next footnote
in this section.
.fn off
```

Since footnotes do not cause breaks, you can interrupt a sentence to place the footnote on the line above the word it refers to, even if the word is in the middle of a sentence.

Because the environment is saved during a footnote definition and restored after it, any formatting changes within the footnote (such as indention, font changes, revision codes, and so on) are automatically restored to their previous values when the footnote is ended. In the example above, therefore, it was not necessary to reset the indention. See "Chapter 9. The SCRIPT/VS Formatting Environment" on page 109 for details about saving and restoring the formatting environment.

Note: The control words that are disallowed within a keep are also disallowed within a footnote.

STARTING A NEW PAGE OR COLUMN

When you want to place text on a new page, you can:

- Precede it with a .H1 [Head Level 1] control word.
- Precede it with a .PA [Page Eject] control word to force a new page.
- Precede it with a .CP [Conditional Page Eject] control word to force a new page if not enough space remains on the current page.
- Use the .DI [Delay Imbed] control word to save the input text until the next page eject occurs, then to process it. (The .DI [Delay Imbed] control word is described in "Chapter 11. Combining SCRIPT/VS Files" on page 119.)

When you want to place text in a new column:

- Precede it with a .CB [Column Begin] control word to force a new column.
- Precede it with a .CC [Conditional Column Begin] control word to force a new column if not enough space remains in the current column.
- Use a .KP [Keep] DELAY control word to keep a block of text together and print it in the next column.
- Use a .KP [Keep] control word to keep a block of text together and print it in the next column if it won't fit in the current column.

[2] This is the next footnote in this section.

Many of these control words are discussed in other parts of the book. The following section describes the .CP [Conditional Page Eject] and .CC [Conditional Column Begin] words.

Conditional Column and Page Ejects

The .CP [Conditional Page Eject] and the .CC [Conditional Column Begin] control words allow you to specify how much space must remain in the column for SCRIPT/VS to continue formatting lines in that column. If there is not enough space remaining, SCRIPT/VS performs the page (or column) eject. For example:

```
This list includes
.sk
.cp 3
GML Tags
Symbols
Macros
```

When the .CP [Conditional Page Eject] control word is encountered, SCRIPT/VS determines the number of lines left in the column. For the example above, if there are at least three lines, processing continues and the lines are printed on the current page. If there are fewer than three lines, however, SCRIPT/VS performs a page eject; the lines following the .CP control word are printed on the next page.

When you use the .CP [Conditional Page Eject] control word, SCRIPT/VS always ejects to the next page when less than the required amount of space remains in the column.

The .CC [Conditional Column Begin] control word works in an analogous manner. A column eject (which might result in a page eject if it occurs when the page's last column is processed) is performed when there are fewer than the required number of lines left in the column.

MARKING UPDATED MATERIAL

If you process documents that are frequently revised, you can identify revised text with a "change bar" (or other symbol) in the left margin.[34] Use the .RC [Revision Code] control word to identify changed material. You can establish up to nine different revision code characters, which are printed to the left of your text output.

For example, the lines

```
.rc 1 *
.rc 2 ?
```

define two different revision codes. Within the body of your document, you can bracket revised material with pairs of .RC [Revision Code] control words. The control word

```
.rc 1 on
```

* indicates the beginning of a revised piece of text. (If a piece of
* text is flagged as revised, and the revision code has not been
* defined, the default code is blank, or no revision code at all.)
* The control word

```
.rc 1 off
```

indicates the end of the revised piece of text.

---

[34] See "Appendix E. Formatting Considerations for the 3800 Printer" on page 385 for special considerations regarding the use of .RC within documents that are printed on the IBM 3800 Printing Subsystem.

Differently marked pieces of revised text may overlap, and their revision codes may be nested. For example, if you have specified

```
     .rc 1 on
```

* and then, while revision code 1 is on, specify

```
     .rc 2 on
```

? revision code 1 is suspended, and revision code 2 is turned on.
? When you turn revision code 2 off,

```
     .rc 2 off
```

* revision code 1 is restored to its former "on" status.

When you have changed only a single line, you can indicate that that line be flagged with a revision code by specifying

```
     .rc 1 on/off
```

* rather than bracketing the line with ".rc 1 on" and ".rc 1 off".

You may also flag a single line by specifying

```
     .rc * $
```

$ when you have not defined an appropriate revision code.

The revision code is placed to the left of the column of text to which it applies. For the leftmost column, the revision code is placed in the binding area provided with the BIND option of the SCRIPT command. For other columns, it is placed in the intercolumn gutter. If the space for the revision code is insufficient, the revision code is omitted.

When you do not want a revision code character to be printed, you can respecify the character to a blank character with the .RC control word. For example,

```
     .rc 1
```

Revision code 1 now prints as a blank.

Ordinarily, revision codes are placed in the gutter two spaces to the left of the column, so that a single blank separates the revision code from the column text. You can change this separation with the ADJUST parameter of the .RC [Revision Code] control word. For example,

```
     .rc adjust 1
     .rc 1 on
```

*specifies that the revision code be placed immediately adjacent
*to the column text, and

```
     .rc adjust 1i
```

* specifies that the revision code is to be placed one inch from the
* edge of the column. If a value is specified which exceeds the
* available gutter space,

```
     .rc adjust 30cm
```

the revision code is not printed.

DRAWING BOXES

SCRIPT/VS can draw boxes around illustrations or text, and can format charts with horizontal and vertical lines.[35] The control word that draws boxes and lines within boxes is the .BX [Box] control word. You use the .BX control word in three different ways to create a box:

1. Define the left- and right-hand edges of the box, and the character positions you want to contain vertical lines. For example, to create a box 30 spaces wide, starting in character position 1, with vertical lines at character positions 10 and 20, specify

   ```
   .bx 1m 10m 20m 30m
   ```

   This formats and prints a box top, with upper corners and descenders:

   ```
   ┌────────┬─────────┬─────────┐
   ```

2. Each time you want a horizontal line within the box, specify the .BX [Box] control word with no other parameters:

   ```
   .bx
   ```

   results in

   ```
   ├────────┼─────────┼─────────┤
   ```

   The lines are drawn with intersections at the vertical rule character positions.

3. When you want to complete the box, use the OFF parameter of the .BX [Box] control word. For example,

   ```
   .bx off
   ```

   This terminates the box definition and draws a bottom line with lower corners and ascenders.

   ```
   └────────┴─────────┴─────────┘
   ```

After a box is started, SCRIPT/VS processes and formats output lines as usual. When each line is formatted and ready to print, SCRIPT/VS inserts box vertical rule characters wherever appropriate to continue the box's vertical lines[36] on the output line.

You can use the .BX [Box] control word to build a three-column table, and use tabs to align text within the rules:

---

[35] See "Appendix E. Formatting Considerations for the 3800 Printer" on page 385 for special considerations regarding the use of the .BX [Box] control word within documents that are printed on the IBM 3800 Printing Subsystem.

[36] The box may be considered to be overlaid on the formatted text; vertical rules will "cover up" text characters which fall "beneath" them.

```
    .ti ¬ 05
    .tb 11m 21m
    .*
    .bx 1m 10m 20m 55m
    .cl 53m
    .in 21m
    .un 19m
    Item 1 ¬Part 1 ¬The first part
    of item 1 is described here.
    .sk
    .un 10m
    Part 2 ¬The second part of item 1 is
    described here. It is a rather long description.
    .bx
    .un 19m
    Item 2 ¬Part 1 ¬The second and
    subsequent items are entered in a similar fashion.
    .bx
    ---
    .bx off
```

The above example results in

Item 1	Part 1	The first part of item 1 is described here.
	Part 2	The second part of item 1 is described here. It is a rather long description.
Item 2	Part 1	The second and subsequent items are entered in a similar fashion.

Note: The character positions defined with the .BX control word are the positions at which the vertical lines are drawn. Contrast this with the displacement setting of the .TB [Tab Setting] control word (.TB 12m results in spaces through character position 12; the text begins in character position 13). Therefore, you can use the same numbers for the .BX control word and for the .TB control word, and use the tab to position to the character position immediately after the vertical bar.

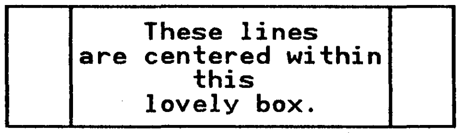
SCRIPT/VS constructs the corners and rules of boxes from the most appropriate characters available, based on the logical output device and current font. For example, the input lines

```
    .bx 1m 5m 25m 29m
    .cl 30m
    .ce on
    These lines
    are centered within
    this
    lovely box.
    .ce off
    .bx off
```

when formatted for a terminal appear as:

```
     +---+-------------------+---+
     |   |    These lines    |   |
     |   |are centered within|   |
     |   |       this        |   |
     |   |    lovely box.    |   |
     +---+-------------------+---+
```

However, when the same input lines are formatted for the IBM 3800 and one of the fonts provided with SCRIPT/VS, they appear as:


These lines
are centered within
this
lovely box.

SCRIPT/VS chooses the appropriate box character set for the logical output device. However, you can force SCRIPT/VS to use any of the box character sets. (See "Defining Internal Fonts" on page 107.) The ability to force SCRIPT/VS to use a specific box character set is important, because some box character sets, such as 3270 text and APL, are never automatically selected.

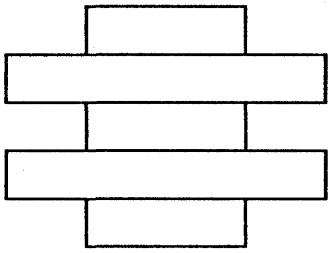
You can use SCRIPT/VS to produce many different box configurations, horizontal lines, and graphic structures. Some of the ways you can use the .BX [Box] control word are described below.

Stacking one box on another

You can define a box and then define a larger or smaller box, without first ending the first box's definition. The top of the second box is printed on the same line as the bottom of the first box. For example, the lines:

```
.bx 10m 20m
.sp
.bx 5m 25m
.sp
.bx 10m 20m
.sp
.bx 5m 25m
.sp
.bx 10m 20m
.sp
.bx off
```

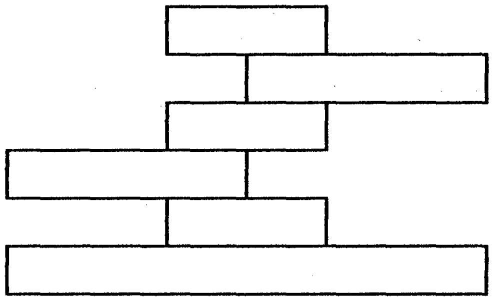
result in:



By using this form of the .BX [Box] control word, you can create a complex structure of boxes. For example, the lines

```
.bx 10m 20m
.sp
.bx 15m 30m
.sp
.bx 10m 20m
.sp
.bx 1m 15m
.sp
.bx 10m 20m
.sp
.bx 1m 30m
.sp
.bx off
```

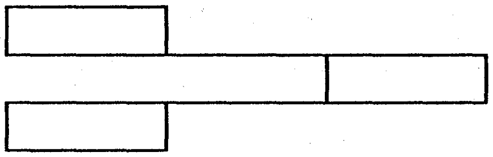
result in



When the upper box bottom line does not touch the lower box top line, SCRIPT/VS joins the two lines together. For example, the input lines:

```
.bx 10m 20m
.sp
.bx 30m 40m
.sp
.bx 10m 20m
.sp
.bx off
```

result in



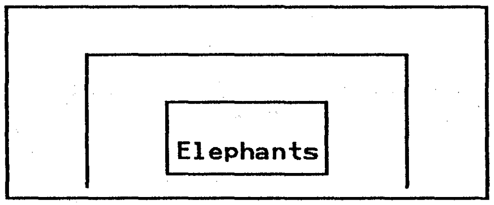
Drawing a box within a box

You can draw a box within a box, using the NEW parameter of the .BX [Box] control word.

Each box is ended with a .BX CAN or .BX OFF control word. Note the different results of each type of ending. For example,

```
.cl 30m
.bx 1m 30m
.sp
.bx new 5m 25m
.sp
.bx new 10m 20m
.sp
.ce Elephants
.bx off
.bx can
.bx off
```

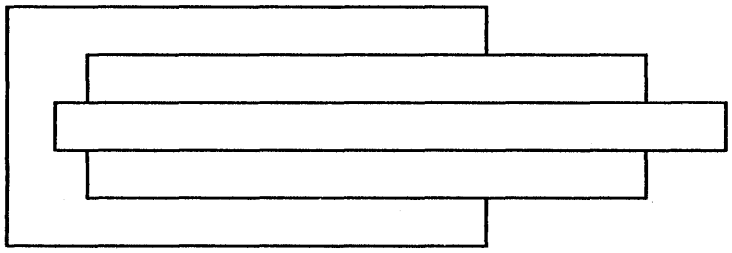
results in



When nesting boxes, the new box does not have to be completely within the previous box. For example,

```
    .bx 1m 30m
    .sp
    .bx new 5m 40m
    .sp
    .bx new 3m 45m
    .sp
    .bx off
    .sp
    .bx off
    .sp
    .bx off
```

results in

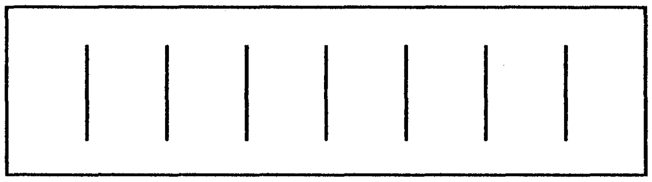


Drawing the middle portion of a box within another (larger) box

You can draw a box that is open at the top and bottom by using slashes (/) between the character position displacements (as shown previously). You can also nest that type of box within a larger box. For example

```
    .bx 1m 40m
    .sp
    .bx new 5m / 10m / 15m / 20m / 25m / 30m / 35m
    .sp 2
    .bx off
    .sp
    .bx off
```

results in

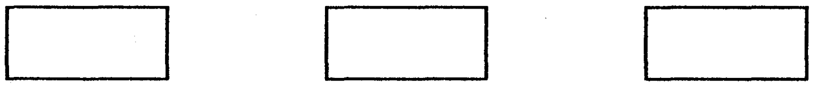


Drawing boxes in a horizontal row

You can draw a row of boxes by specifying a box definition with slashes. For example,

```
    .bx 1m 10m / 20m 30m / 40m 50m
    .sp 2
    .bx off
```

The slash indicates a discontinuity with no horizontal connection. These lines result in:

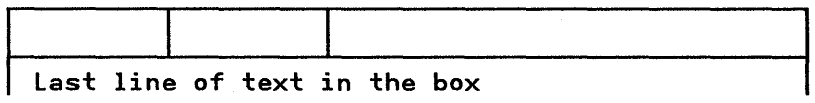


Drawing the top line (only) of a box

When you want SCRIPT/VS to draw the top portion of a box, but not the bottom line, you use the CAN parameter of the .BX [Box] control word to cancel the box definition. For example,

```
.bx 1m 10m 20m 50m
.sp
.bx 1m 50m
Last line of text in the box
.bx can
```

results in

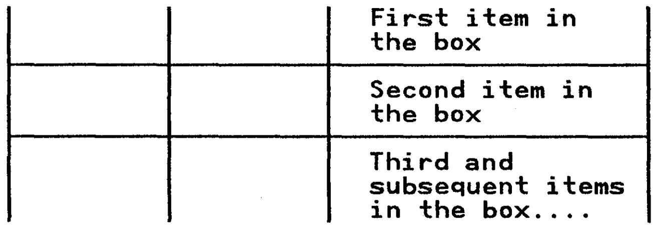


Drawing the middle portion of a box (without top or bottom lines)

When you want SCRIPT/VS to draw a box without horizontal top and bottom lines, use the SET parameter of .BX to specify the positions of the vertical rules. Subsequent text will be formatted and overlaid with vertical rules, but no box top will be drawn. For example,

```
.in 22m
.cl 38m
.bx set 1m 10m 20m 40m
First item in the box
.bx
Second item in the box
.bx
Third and subsequent items
in the box
.bx can
```

results in

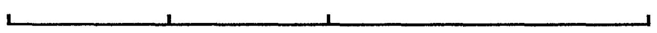


Drawing the bottom line (only) of a box

When you want SCRIPT/VS to draw the bottom line of a box, you use the .BX [Box] control word as you would to define the start of a box <u>and</u> you include the OFF parameter. For example,

```
.bx off 1m 10m 20m 40m
```

results in



<u>**Drawing boxes with the 3800 Printer**</u>

Special considerations apply to boxes when the output is being formatted for a 3800 Printer. Because SCRIPT/VS does not provide three widths of each box character in each font, SCRIPT/VS performs monospace justification inside a box. The following restrictions apply within a box:

- All nested boxes are in the font of the outermost box, regardless of the font changes within the box.
- All fonts used within the box must be of the same pitch as the box itself (that is, the pitch of the current font when the outermost box was begun).
- Proportional fonts (for example, GP12) cannot be used within a box.

You can produce boxes of different line thicknesses containing text in several fonts. For example,

```
.bx 1m 20m
The
.bf GB12
first
.pf
box
.bx off
.sp 2
.bf GB12
.bx 1m 20m
The
.bf GT12
second
.pf
box
.bx off
```

results in:

The **first** box

The second **box**

DEFINING INTERNAL FONTS

SCRIPT/VS extends the concept of font to include underscoring and capitalization on all devices, overstriking on impact printers, and stopping to change typing elements on typewriter terminals.

Use the .DF [Define Font] control word to define internal fonts. For example, The "up" parameter of the .DF control word includes capitalization as part of the font:

```
.df caps up
```

You now capitalize text by entering

```
.bf caps
```

AND RESET CAPITALIZATION BY ENTERING

```
.pf
```

When formatting for the 3800 Printer, formatting attributes such as underscoring and capitalization can be combined with "real" fonts and managed simultaneously. For example,

```
.df gb12 us font gb12
```

redefines the font GB12 to include underscoring as well as the 12-pitch gothic bold font. Now the input line

```
.bf &$CHAR(2)
```

will underscore text formatted in the font GB12.

When formatting for an impact printer, you can create boldface headings and emphasize important phrases by overstriking. For example, the "os" parameter of .DF includes overstriking as part of the font:

```
    .df boldface os rpt 4
```

defines a font that is formed by overstriking the text with itself four times. You can emphasize phrases by changing to the new font with

```
    .bf boldface
```

Overstriking is ignored for devices other than the 1403 and 2741, unless overstriking with the underscore character is specified. For example,

```
    .df under os char _
```

defines a font that underscores text, just as

```
    .df under us
```

does, except that blanks are never overstruck.[37]

When formatting for a typewriter terminal with changeable typing elements, you can define those elements as fonts with the STOP attribute. Whenever you format text in that font, SCRIPT/VS will stop typing to allow you to change elements. See "Interactive SCRIPT/VS Processing" on page 126 for a discussion of the use of the STOP parameter of the .DF [Define Font] control word.

LINE REFERENCE NUMBERS

You can specify that all nonblank lines in the body of a page be serially numbered by entering

```
    .rn on
```

The line reference numbers will be placed to the right of the rightmost column.

[37] Underscoring of blanks is controlled by the .UD [Underscore Definition] control word; overstriking, even with the underscore character, affects only nonblank characters.

CHAPTER 9. THE SCRIPT/VS FORMATTING ENVIRONMENT

The formatting environment is a set of values and parameters that specify exactly how SCRIPT/VS is to format each line on an output page. The formatting environment consists of three parts:

- The active environment, which contains parameters for formatting text
- The page control area, which contains parameters that define the entire page
- The translate tables associated with the .TI [Translate Input] and .TR [Translate Character] control words

PARAMETERS THAT DEFINE THE FORMATTING ENVIRONMENT

The parameters that make up the active environment area and the page control area are listed in Figure 34 on page 360. Each parameter, its corresponding control word, its initialized value, and its special SCRIPT/VS symbol (if any) is listed.

When SCRIPT/VS ejects to a new page (or begins the first page), it prepares the output page in the following manner:

1. It saves the active environment values for body text and initializes the active environment for formatting:

   - Top titles
   - Running heading
   - Running footing
   - Bottom titles

   The active environment is reinitialized before each of these is formatted.

2. Top and bottom page floats are selected from the float queue. If any exist and will fit, they are placed on the page.

   The output page's running headings and footings, running titles, and page floats are now in place on the output page. All "page control" dimensions are fixed for this page, and any changes to these values will take effect on the next page.

3. SCRIPT/VS restores the active environment for body text that it had saved.

   Input lines are processed to produce output lines, which are inserted into the body of the page. When the page is full, or when a page eject occurs, the formatted page is sent to its destination.

The Keep and Float Environments

When floats and keeps (other than inline keeps) are started, SCRIPT/VS saves a copy of the active environment and then modifies the active environment by clearing the values of the .OF [Offset] and .UN [Undent] control words, and restoring the indention to the basic .IN [Indent] value currently in effect. In addition, for page floats, the column width is set equal to the line length value.

When the keep or float ends, the saved copy of the active environment is restored.

The Footnote Environment

When a footnote is started, SCRIPT/VS saves a copy of the active environment and then modifies it:

- The values of the .OF [Offset] and .UN [Undent] control words are cleared, and indention is restored to the basic .IN [Indent] value currently in effect.
- Column width is set to line length. The footnote text goes across the page in single-column format.

When the footnote ends, the saved copy of the active environment is restored.

SAVING AND RESTORING THE CURRENT FORMATTING ENVIRONMENT

The .SA [Save Status] and .RE [Restore Status] control words are used to save and restore the current formatting environment. All three parts of the environment are saved and restored by .SA and .RE:

- The active environment
- The page control area
- The .TI and .TR translate tables

For example, part of an input file that contains a distribution list requires indention and tab settings to format properly. However, the main document indention and tab settings are different. To avoid having to restore the main document's values, use the .SA [Save Status] and .RE [Restore Status] control words:

```
.sa
.in
Distribution list for special publications:
.sk
.in 3m
.ti ¬ 05
.tb 2i 2.5i
.us Name ¬Dept ¬Address
 .
 .
 .
.* End of distribution list
.re
```

CHAPTER 10. CONDITIONAL PROCESSING

SCRIPT/VS provides several methods for processing input conditionally. You can write input files and macros that are capable of making simple decisions, and taking action based on the result. With conditional processing techniques, you can:

- Select the alternative input lines to be processed in a particular run.
- Construct loops that process the same material several times to provide several copies of the formatted output. (Each copy can, of course, contain different specific information, as in the form letter example in "Chapter 11. Combining SCRIPT/VS Files" on page 119.)
- Write macros which cause different formatting based on the logical output device or other variables.
- Provide processing based on the content of an input line.

These capabilities use the basic conditional processing techniques in conjunction with other techniques that are not discussed here. "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147 contains information about the mechanics of writing macros, and "Chapter 12. Symbols in Your Document" on page 129 discusses symbol substitution. Individual control words are described in "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219.

There are three basic conditional processing techniques:

- The .IF control word family
- Conditional sections
- Conditional processing with symbols

THE .IF CONTROL WORD FAMILY

SCRIPT/VS allows you to test a symbol's value to determine whether to process an input line or ignore it. To do this, you can use the .IF [If] control word by itself or in conjunction with the .AN [And], .OR [Or], .TH [Then], and .EL [Else] control words. Using the .IF [If] control word by itself is the simplest way of specifying a conditional statement. This control word is specified in the form:

```
.if comparand1 test comparand2 target-line
```

Each comparand can be up to 255 characters in length,[38] and the shorter comparand will be padded with blanks to match the length of the longer comparand.

The conditions you can test for and the codes you can use are:

[38] The entire input line, after substitution, cannot be longer than 255 characters. When comparing symbols that may potentially have long values, or contain blanks, it is recommended that the .IF control word be performed with substitution off, as described in "Special Techniques for Conditional Processing" on page 113.

Code	Meaning
= or eq	equal to
¬= or ne	not equal to
> or gt	greater than
< or lt	less than
>= or ge	greater than or equal to
<= or le	less than or equal to

The target-line part of the .IF [If] control word can be any valid SCRIPT/VS input line: a control word, a symbol, a macro, or text. The first nonblank character after "comparand2" is treated as the first character of the input line. If the condition is true, the input line is processed by SCRIPT/VS. Otherwise, it is ignored.

Alternative Processing

There may be times when, depending on the results of a comparison, alternative processing may occur. You can use multiple .IF [If] control words to handle this situation or you can use the .TH [Then] or .EL [Else] control words in conjunction with an .IF control word. For example, specifying

```
    .if &street eq Broadway .se branch = Commercial
    .if &street ne Broadway .se branch = Warehouse
```

causes the same results as specifying

```
    .if &street eq Broadway
    .th .se branch = Commercial
    .el .se branch = Warehouse
```

Both of these examples result in the symbol &branch being set to the value "Commercial" if the comparison is equal, and to the value "Warehouse" if it is not.

The .TH [Then] and the .EL [Else] control words cause their targets to be executed or ignored based on the results of the most recently executed comparison in the current file or macro. Therefore, a series of conditionally executed lines can follow a single comparison. For example, specifying

```
    .if &job eq chimney-sweep
    .th .sp 2
    .th .notes height of roof
    .el .us salary
    .el .in 5
    .el .im salary &job
    .th .sp 2
    .el .sp 4
```

causes all of the .TH and .EL control words that follow the .IF control word to be executed or ignored based on the result of its comparison. Other .IF control words that may be contained in the ".notes" macro or the "salary" file do not affect this series of control words since the result of the most recent comparison is preserved across macro calls and imbedded files.

There may also be times when you want to test for multiple conditions. This can be accomplished by using the .AN [And] and .OR [Or] control words in conjunction with the .IF control word. For example, you might have a situation where two conditions have to be true before a certain type of processing can occur. In this situation, specify

```
    .if &city = Rochester .an &state = Minnesota .se zip = 55901
```

which causes the symbol &zip to be set to 55901 if both conditions are true.

Similarly, you may have a situation where only one of multiple conditions has to be true for one type of processing to be done. In this case, you might specify

```
.if &city eq Cupertino .or &city eq Gilroy
.th .carpool &city
.el .se city =
```

The macro .carpool will be invoked if the value of the variable &city is either "Cupertino" or "Gilroy"; if it is neither, the variable &city will be reset to null.

Bypassing Part of an Input File

When you want to bypass a part of your input file, you can use the .GO [Goto] and ... [Set Label] control words. For example:

```
.if &type = 1 .go bypass
    .
    .
    .
...bypass
```

In the above example, if the symbol "&type" has a value of 1, all the control words and text between the .IF and the ... [Set Label] control words (which sets the label "bypass") are skipped.

Conditional processing with the .IF [If] control word can be especially convenient when one file is imbedded in several different masterfiles. You can provide for slight differences among the files by setting the same symbol to a different value in each masterfile, and using that symbol to determine how processing is to be done in the imbedded file.

The SYSPAGE and SYSOUT Comparands

There are two special comparands you can use with the .IF [If] control word family. They are keywords, not symbols. Therefore, they should not be prefaced with an &.

- The keyword SYSPAGE tests whether the page currently being formatted is EVEN or ODD. You can use SYSPAGE to place text on an output page, based on whether the output page is even-numbered or odd-numbered:

  ```
  .if SYSPAGE = EVEN .sx /Evenpage Top Line///
  .if SYSPAGE = ODD .sx ///Oddpage Top Line/
  ```

- The keyword SYSOUT tests whether the destination of the output is the printer (PRINT) or the terminal (TERM). This keyword is provided for compatibility with SCRIPT/370. The SCRIPT/VS system symbols &$LDEV and &$PDEV provide a better way to test which of the many logical and physical output devices possible with SCRIPT/VS is currently in use.

Special Techniques for Conditional Processing

There are several techniques you should be aware of when using the .IF [If] family of control words.

- Comparing Null-Value Symbols

  When you specify the name of a symbol value that might be null, you should precede the symbol name with a character-prefix to avoid a possible error. For example, the input line

  ```
  .se a = ''
  .if &a = ON .go next
  ```

  results in a SCRIPT/VS error because the symbol &a was set to a null value. The conditional statement resolves to:

  ```
  .if = ON .go next
  ```

The "=" character is treated as the first comparand, and "ON" is not a valid comparison. However, the input line

```
.if /&a = /ON .go next
```

resolves to

```
.if / = /ON .go next
```

When the symbol &a has the value "ON," it resolves to

```
.if /ON = /ON .go next
```

That is, the prefix "/" is concatenated with the value of &a to result in "/ON", which satisfies the test. When the symbol &a hasn't been set to ON, "/&a" results in "/" and the test fails, but no error results.

- Comparing Symbols Containing Special Characters

  The .IF [If] control word family, like the .SE [Set Symbol] control word, is capable of resolving symbols in its comparands even if symbol substitution is off. This is essential when comparing symbols whose values might contain special characters, such as blanks and control word separators, or whose values might be very long. For example, with symbol substitution on, the input line

  ```
  .if &needle eq &haystack .th .im lost
  ```

  might result in

  ```
  .if Rachel's MG eq Parking Lot .th .im lost
  ```

  after symbol substitution has occurred. This would result in an error since "Rachel's" would be interpreted as the first comparand and "MG" would be interpreted as an invalid comparator. With substitution off, the symbols &needle and &haystack will be recognized as the comparands, and symbol substitution will be performed on the two comparands separately before they are compared.

- Comparing Potentially Long Comparands

  After substitution, an input line cannot be longer than 255 characters. If your input line might exceed 255 characters after substitution has been performed, the .IF control word should be processed with substitution off.

CONDITIONAL SECTIONS

When your document is likely to be read by several different audiences, you may want to build it so that you can customize it for each. For example, your company might build three very similar devices:

- Class A Widget, which is a very basic machine
- Class B Widget, which is really an improved Class A Widget
- Class C Widget, which includes some, but not all, of the features of Class A and B Widgets, and includes some improvements of its own

Because the devices are very similar, you can write a section of material that applies to all three. You can follow each general information paragraph or section with more specific information applicable to one or two, but not all three, of the device types.

In this way, you can keep all information about Widgets in one input file. When you format the input file for printing, however, SCRIPT/VS can customize it so that all information (general as well as specific) about one of the classes of Widgets is printed.

To do this, you identify those sections of the input file that are to be processed conditionally.

SCRIPT/VS processes a conditional section, or ignores it, based on the setting of a .CS [Conditional Section] control word. Each conditional section number, from 1 to 9, can be used many times in a document. You can associate each type of information to be processed conditionally with its own conditional section number. For example,

Conditional Section Number	Conditional Section Applies To
1	Only Class A Widgets
2	Only Class B Widgets
3	Only Class C Widgets
4	Class B or Class C Widgets (Not Class A)
5	Class A or Class C Widgets (Not Class B)
6	Class A or Class B Widgets (Not Class C)

At the beginning of your document, specify that SCRIPT/VS is to bypass all conditional sections with the IGNORE parameter of the .CS [Conditional Section] control word. The SCRIPT/VS default is to process all conditional sections not specifically bypassed.

```
.cs 1 ignore
.cs 2 ignore
    .
    .
    .
.cs 6 ignore
```

Before you issue the SCRIPT command to process your document, change some of the .CS [Conditional Section] IGNORE control words to .CS [Conditional Section] INCLUDE control words, to process each desired conditional section. For example, to print all material appropriate for readers interested in Class B Widgets, specify

```
.cs 2 include
.cs 4 include
.cs 6 include
```

In the body of your input file, you identify each conditional section by preceding it and following it with the .CS [Conditional Section] control words, using the ON and OFF parameters. For example,

```
.cs 2 on
This material applies only
to Class B Widgets.
It does not apply to either
of the other types.
.cs 2 off
```

Because you can only specify one conditional section number with the .CS [Conditional Section] control word, you must use a separate number to identify sections that apply to either one of two (but not the third) type of device.

When you "nest" the .CS [Conditional Section] control words, you identify a section that applies only when two (or more) conditions are met. For example,

```
.cs 1 on
.cs 2 on
This material is applicable to
people who use the Class A Widget
with the Class B Widget.
It is not to be printed for
readers interested only in Class A
Widgets or only  in Class B Widgets.
.cs 2 off
.cs 1 off
```

Because the .CS [Conditional Section] control word doesn't cause a break, you can process small units of text conditionally. For example, the input lines

```
.cs 1 ignore
.cs 2 ignore
.cs 3 include
This book is written specifically
for the operator of a
.cs 1 on
Class A
.cs 1 off
.cs 2 on
Class B
.cs 2 off
.cs 3 on
Class C
.cs 3 off
Widget.
```

are printed as:

```
This book is written specifically for the
operator of a Class C Widget.
```

The input lines (GML tags, control words, and text) between the .CS ON and the .CS OFF control words are included unless explicitly bypassed as a result of a preceding .CS IGNORE control word. Such a bypass is not a total one: macros and GML tags are resolved.

When ignoring a conditional section n, SCRIPT/VS recognizes one control word only:

```
.cs n off
```

which ends the ignored conditional section. All other input lines (control words as well as text lines) are ignored without further processing. This means that an ignored conditional section that is started in one file cannot be ended in another file that is imbedded by the first, because the .IM [Imbed] control word will not be processed in the ignored section. However, if a conditional section is started in an imbedded file, it can be ended in the outer file, because SCRIPT/VS returns to the outer file automatically when it reaches the end of the imbedded file. No control word is needed to switch back from the end of an imbedded file to the file that imbedded it.

CONDITIONAL PROCESSING WITH SYMBOLS

With set symbols, you can do conditional processing in several ways. The simplest of these is to have a symbol that resolves into one control word or another depending on the conditions. For example, the symbol "xim" could be set to either ".CM" or ".IM" to cause the input line

```
&xim filename
```

to be treated as an .IM [Imbed] control word or a .CM [Comment] control word. If your file has several places at which another file should be imbedded conditionally, the symbol "xim" could be defined once to control all occurrences of the symbolic control word.

Another technique uses the existence attribute (&E') of a symbol to generate a macro name according to whether a symbol exists or not. See "Chapter 12. Symbols in Your Document" on page 129 for details on symbol attributes. The existence attribute causes a string to be substituted with 0 if a symbol does not exist, and with 1 if it does. You could write a macro called "X0" to provide the appropriate processing when a given symbol does not exist, and another called "X1" for when it does exist. Now, the expression:

```
.X&E'&name
```

will resolve to ".X0" if the symbol "&name" does not exist, and ".X1" if it does.

You can also use the symbol length attribute (&L') to perform conditional processing. The length attribute and the following string or symbol are replaced with the length of the string or symbol during substitution. See "Chapter 12. Symbols in Your Document" on page 129 for details. If a symbol called "&num" contains a number that is from one to five digits long, you can develop a 5-digit number by adding the correct number of leading zeros to &num. First, you need to define symbols that contain the number of zeros needed for each possible length the number might be:

```
.se 5z =
.se 4z = 0
.se 3z = 00
.se 2z = 000
.se 1z = 0000
```

If the number is five digits long, you need to add no zeros. If it is four digits long, you need one zero, and so forth. Now, the expression

```
&&L'&num.z.&num
```

concatenates the correct number of zeros to the number to form a 5-digit number. One part of the expression, "&L'&num", is resolved to the number 1, 2, 3, 4, or 5, whatever the length of the number in the symbol &num happens to be. If it is 3, the expression becomes "&3z.&num". The symbol "&3z" is now replaced with 2 zeros, the proper number of zeros for a 3-digit number, and concatenated with the number itself when "&num" is substituted.

CHAPTER 11. COMBINING SCRIPT/VS FILES

SCRIPT/VS provides the ability to combine many SCRIPT/VS input files for processing as a single document. The control words that allow you to do this are:

- .IM [Imbed], which causes SCRIPT/VS to process another file immediately, then return to the imbedding file
- .AP [Append], which causes SCRIPT/VS to process another file immediately and not return to the appending file

IMBEDDING AND APPENDING FILES

An input file can "call" other input files with the .IM [Imbed] and .AP [Append] control words:

- When a file is imbedded into an outer file, the contents of the imbedded file are read and processed as though they were inserted into the outer file immediately following the .IM [Imbed] control word. When the imbedded file completes, SCRIPT/VS resumes processing at the outer file's input line that follows the .IM [Imbed] control word.
- When a file is appended to another file, the contents of the appended file are processed immediately. The appended file replaces the appending file as the current input file. Consequently, when the appended file completes, SCRIPT/VS does not resume processing input lines in the appending file.

You must specify the filename of the file you want to imbed or append. If the SCRIPT/VS file named OUTER processes the input line

```
.im tester
```

SCRIPT/VS stops reading input lines from the OUTER file and begins reading and processing lines from a file named TESTER. Whatever formatting controls are in effect when the file is imbedded remain in effect until respecified by control words in TESTER. When SCRIPT/VS reaches the end of the TESTER file, it continues processing in OUTER with the input line following the .IM [Imbed] control word.

The file TESTER can also contain .IM [Imbed] control words to imbed additional files. The process is the same as when TESTER was imbedded. The imbedded file is read and processed, then SCRIPT/VS returns to the line in the imbedding file that follows the .IM [Imbed] control word.

For example, consider the following four files:

```
MASTER:       FILEA:        FILEB:        FILEC:

.im filea     The quick     brown fox     over
.im filec     .im fileb                   the lazy
dog.          jumps
```

When you issue the SCRIPT command to format the MASTER input file, the result is:

```
The  quick brown fox
jumps over  the lazy
dog.
```

The .AP [Append] control word is similar to the .IM [Imbed] control word, except that when SCRIPT/VS finishes processing the input lines from a file specified in a .AP control word, it does not return to the calling file. For example, when SCRIPT/VS processes the input line

```
.ap names
```

it closes the current input file and begins processing the NAMES file. All the lines from the NAMES file are treated as though they were in the original file. When the end of the NAMES file is reached, SCRIPT/VS does not return to the file that appended it:

- If the file that appended NAMES was the file named in the SCRIPT command, SCRIPT/VS completes processing.
- Otherwise, if the file that appended NAMES was itself imbedded, SCRIPT/VS returns to the next input line in the file that originally imbedded the file that appended NAMES, as shown in Figure 10.

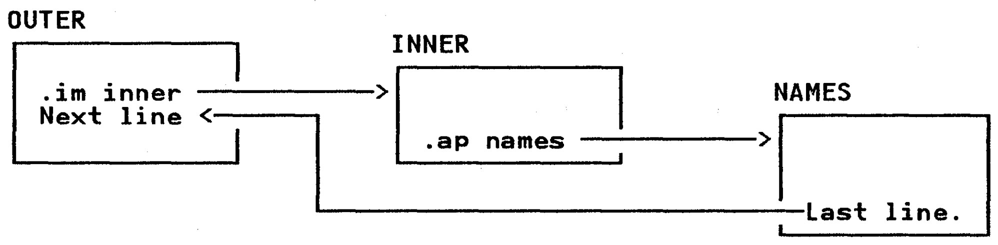


Figure 10. Imbedding and Appending SCRIPT/VS Files

You can pass values to the imbedded or appended file, so the file can be customized each time it is called. See "Chapter 12. Symbols in Your Document" on page 129 for details.

Naming the File To Be Imbedded or Appended

The name of the file to be imbedded or appended is given as a 1- to 8-character name with the .IM or .AP control word:

```
.im file-id
.ap file-id
```

"file-id" is an internal SCRIPT/VS name for the file to be read. The external name of the file can be established in one of three ways:

- You can use the .DD [Define Data File-id] control word to associate the "file-id" with any real file or data set name available in the system under which SCRIPT/VS is executing, as described in "Naming the Input File" on page 13.
- If you enclose the "file-id" in parentheses, SCRIPT/VS uses the "file-id" as the real file or data set name.
- If no .DD control word has been processed for "file-id", SCRIPT/VS uses "file-id" to derive the real name of the file or data set to be read, based on rules appropriate for the system under which it is executing.
  - In CMS, "file-id" is used as the name of a CMS file whose filetype is SCRIPT.
  - In TSO, SCRIPT/VS builds a fully qualified name as described in "TSO Naming Conventions" on page 14. using "file-id" as the second qualifier.
  - In ATMS-III, SCRIPT/VS assumes that the document is in the invoking operator's permanent storage.

When the .DD control word defines the file-id, SCRIPT/VS makes assumptions about the file name based on the environment, as explained in ".DD [Define Data File-id]" on page 244.

In CMS, you should use the .DD [Define Data File-id] control word when:

- The imbedded filename is different from the actual CMS filename
- The filetype is other than SCRIPT
- A specific filemode that is not the first in the CMS search sequence is to be used

In TSO, you should use the .DD [Define Data File-id] control word when the imbedded or appended file is not a member of the partitioned data set (PDS) named in the SCRIPT command, or when the member name is different from the file-id.

In ATMS-III, you should use the .DD [Define Data File-id] control word when the imbedded or appended file is not in an operator's permanent storage.

In the batch processing environment, use the .DD [Define Data File-id] control word when:

- The library document name is different from the imbedded filename
- A password is required to access the file
- The file is stored on a library other than the ones listed with SCRIPT command options

The format and usage of the .DD control word when defining a file are described in ".DD [Define Data File-id]" on page 244.

MASTER FILES

There are several advantages to using imbeds in SCRIPT/VS files:

- For convenience in updating and tracking SCRIPT/VS files, you can use one file as the master file for a SCRIPT/VS document. The master file can contain the formatting controls (for page size, depth, column definitions, and so on) that are to be in effect for the entire document. The remainder of the master file might contain only the .IM control words that imbed the remaining files.
- You can easily reorganize a large document that is composed of many small files that are imbedded in a single master file. When you want to move or remove information, you need only to change the position of the .IM [Imbed] control word in the master file, or to delete it.
- Small files can be shared by several master files. Each master file can imbed the small file where appropriate. Therefore, you do not need to keep duplicate copies of the same information.
- While there may be a limit to the number of records that can be contained in a single disk file, there is no restriction on the number of files that SCRIPT/VS can process. Also, many different people can work on pieces of the same document simultaneously.
- Some CMS and TSO editors have a limit on the number of records in a file. With the .IM [Imbed] control word, you can structure a document by combining many small files, each of which can be edited. The document as a whole can be formatted and printed using the SCRIPT command.

Figure 11 on page 122 illustrates a typical master file structure.

When you are proofreading SCRIPT/VS output files that contain many imbed files, you can use the NUMBER option of the SCRIPT command. As a result, SCRIPT/VS prints (next to each output line)

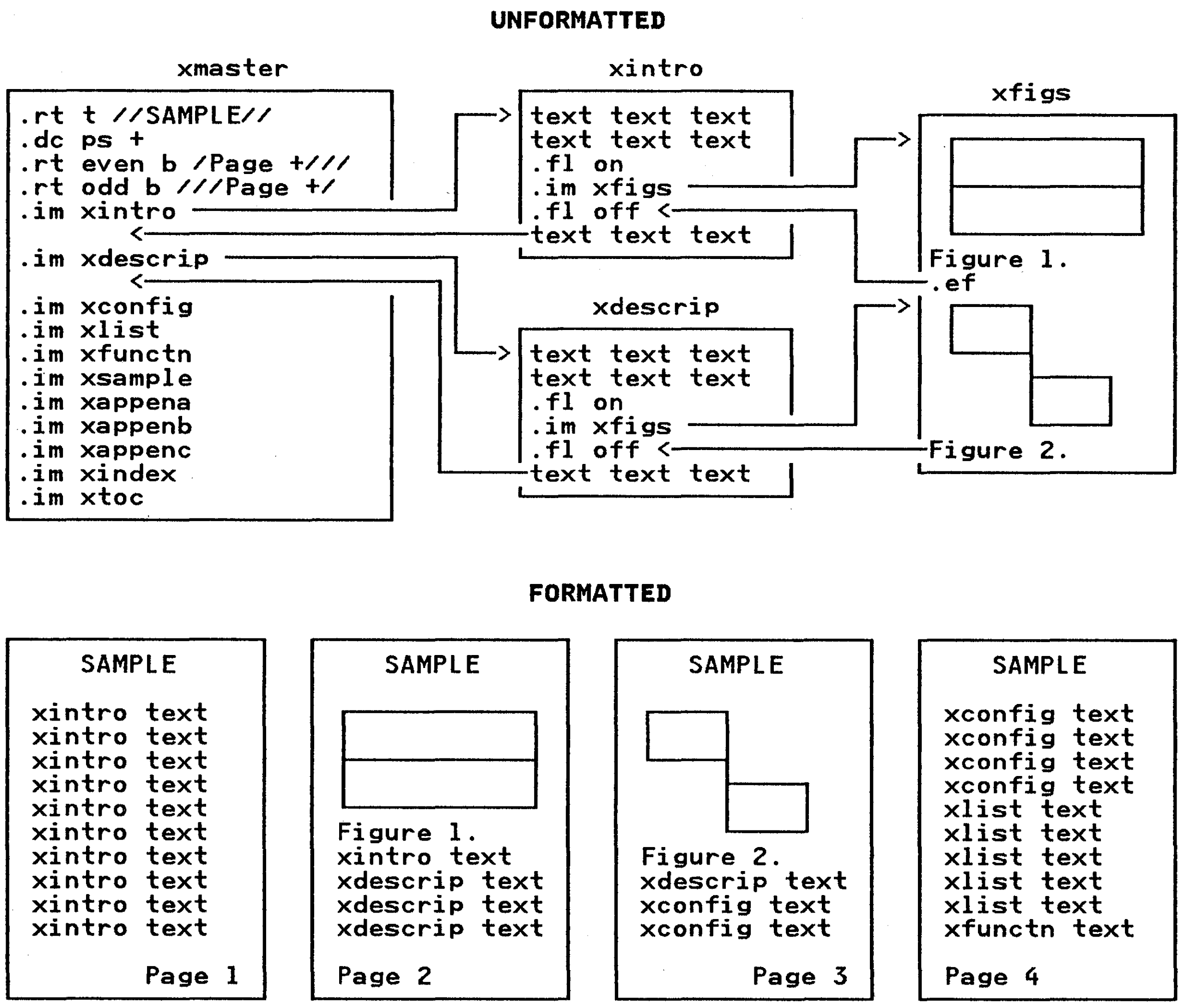


Figure 11. Master File Structure

- The current input source of the file being processed
- The number of the last input line SCRIPT/VS had read when the output line was formatted

The line number and file name are useful when you update and correct the SCRIPT/VS files.

The UNFORMAT option of the SCRIPT command causes SCRIPT/VS to print input lines instead of output lines. SCRIPT/VS produces an unformatted document that contains all the input lines from all imbedded and appended files.

Writing to an Output File

The .WF [Write To File] control word allows you to put input lines into a file dynamically. For example, you can collect figure captions for a figure list in one file and index entries in another.

You can have several .WF files, one for each of several lists you want to build as your document is processed. However, only one .WF file can be open at a time.

When SCRIPT/VS processes the .WF [Write To File] control word, one or more input lines are written to a SCRIPT/VS file named DSMUTWTF.

- You can insert one line into the file with:

  ```
  .wf contents of the input line
    .
    .
    .
  .wf .ce Text to be centered.
  ```

- You can insert a number of lines into the file with:

  ```
  .wf 5
  .in 3m
  .ce 3
  These are the
  lines to go
  into DSMUTWTF.
  ```

- You can also insert a number of lines into the file with:

  ```
  .wf on
    .
    .
    .
  Many input lines
    .
    .
    .
  .wf off
  ```

  Note: The .WF [Write To File] OFF control word must be on an input line by itself.

You can later process the contents of the DSMUTWTF file with the IMBED parameter of the .WF [Write To File] control word:

```
.wf imbed
```

After imbedding the DSMUTWTF file, you can add to the end of it with more .WF control words. You can imbed the DSMUTWTF file into another file many times.

When the contents of DSMUTWTF are no longer useful to you, you can erase the file with the ERASE parameter of the .WF [Write To File] control word:

```
.wf erase
```

The DSMUTWTF file can be erased and reused many times.

Several .WF Files

By using the .DD [Define Data File-id] control word, you can define the file-id DSMUTWTF so that you can use the .WF [Write To File] control word to write lines to several different files. For example, to add lines to the end of the CMS file PART6 SCRIPT A1:

```
.dd dsmutwtf lib part6
.wf on
Input lines
to be added
to PART6.
.wf off
```

Note: If the file (PART6 in the above example) is currently being imbedded or appended, you cannot add lines to it. That is, you cannot write into a file that is currently being read.

To restore the file-id DSMUTWTF to the default real file, specify

```
.dd dsmutwtf lib dsmutwtf
```

Note: In ATMS-III, the .WF control word can only be used to write to a document in CICS/VS auxiliary storage. It can not be used to write to a document in either working or permanent storage.

Delaying the Imbedding of Input Text

The .DI [Delay Imbed] control word is used to store input lines in a SCRIPT/VS file named DSMUTDIM. When you are finished storing lines into it with the .DI control word, SCRIPT/VS continues processing your file's input lines. When a page eject occurs (either because of a full page or because of a control word that causes a page eject), SCRIPT/VS processes any pending keeps and floats and then imbeds the DSMUTDIM file.

When SCRIPT/VS encounters a .DI [Delay Imbed] ON control word, it puts the following input lines into the DSMUTDIM file. The lines are put in DSMUTDIM as they are; SCRIPT/VS does no other processing except to check each line for the .DI [Delay Imbed] OFF control word <u>which must begin in column 1</u>. When the .DI OFF control word is found, SCRIPT/VS continues to format the input lines in the file.

For example, to delay the inclusion of a few input lines until the next page eject occurs, you can specify:

```
.di on
.ce on
********************
*  READ CAREFULLY  *
********************
.ce off
.di off
```

The input lines from .CE ON to .CE OFF are written into the DSMUTDIM file. When one of the lines is a .IM [Imbed] control word, the DSMUTDIM file does not receive the imbedded file's contents. Instead, when the DSMUTDIM file is imbedded at the top of the next page, the file it imbeds is read in and processed.

Rather than begin an open-ended delay imbed with .DI ON, you can specify the number of input lines to be delayed. For example,

```
.di 3
```

causes the next three input lines to be delayed. You don't have to terminate the .DI 3 with .DI OFF. However, .DI OFF ends a delay imbed, whether it was started with .DI ON or with .DI n before n lines have been imbedded.

TERMINATING THE FORMATTING OF A FILE

There are three control words that cause SCRIPT/VS to terminate processing: .EF [End of File], .QU [Quit], and .QQ [Quick Quit].

The .EF [End of File] control word is useful with imbedded files. If a .EF control word occurs in an imbedded file, SCRIPT/VS does not continue imbedding the file but returns to process the outer file. If another .IM [Imbed] control word is encountered that imbeds the same file again, SCRIPT/VS resumes reading and processing with the input line following the .EF control word that was last processed.

You can respecify the start of the file again with the CLOSE parameter of the .EF [End of File] control word:

```
.ef close
```

The next time the file is imbedded, SCRIPT/VS begins reading input lines at the first line.

If the imbedded file is a profile, the contents of the file preceding the .EF [End of File] control word will be processed before the main document; the remainder will automatically be processed after the main document.

The other two control words, .QU [Quit] and .QQ [Quick Quit], cause SCRIPT/VS to terminate processing entirely, regardless of whether the current file is an imbed file or not. When you use the .QU [Quit] control word, processing terminates after SCRIPT/VS prints the remainder of the current page (and any bottom titles and running footings in effect) and after SCRIPT/VS closes all open files. In contrast, the .QQ [Quick Quit] control word causes immediate termination of processing with no final page eject. Therefore, all of the text on the last page will be lost.

The .QQ [Quick Quit] control word can be useful when checking your file for errors. You can specify the TWOPASS option when formatting the file and terminate processing after the first pass completes.

For example, a very long input file named MASTER10 can have the last input line

```
    .qq
```

When you format it at the terminal using the SCRIPT command

```
    script master10 (term twopass
```

the file is completely formatted during the first formatting pass. Errors detected by SCRIPT/VS are displayed at your terminal for you to note and correct later. However, processing terminates before the second pass occurs, when the formatted document would usually be displayed.

MERGING DOCUMENTS FROM SEVERAL SOURCES

You can create a customized document from many different input files by using the .IM [Imbed] and .EF [End of File] control words. An imbedded file can include .EF [End of File] control words to cause a different group of input lines to be processed each time the file is imbedded. This can result in customized documents because each group of lines from the imbedded file can contain the specific information for a particular copy of the basic document.

You can use this technique to create a table whose format and content can be separately updated or altered. To create such a table, you would set up one file containing the table format and the symbolic names for the table entries and another file containing the .SE [Set Symbol] control words that define the actual values for the table entries. For example, consider the following two SCRIPT/VS files:

```
    File: TABLE                    File: TABLSYM

    .tb 3 21                       .se state   'STATE
    .cs 2 on                       .se capital 'CAPITAL
    .cs 1 ignore                   .ef
    .sp 2                          .se state   'Alabama
    .fo off                        .se capital 'Montgomery
    .bx 1 19 36                    .ef
    .se bxoff=                     .se state   'Alaska
    .cs 2 ignore                   .se capital 'Juneau
    .cs 2 off                      .ef
    .im tablsym                    .se state   'Arizona
    &$TAB.&state.&$TAB.&capital    .se capital 'Phoenix
    .bx &bxoff                     .ef
    .cs 1 on                       .se state   'Arkansas
    .fo on                         .se capital 'Little Rock
    .cs 2 include                  .ef
    .sp 2                          .se state   'California
    .ef                            .se capital 'Sacramento
    .cs 1 off                      .se bxoff = off
    .ap table                      .cs 1 include
```

When the command "SCRIPT TABLE" is issued, the table of state capitals will be generated. Each time the file TABLSYM is imbedded, it is read starting with the input line following the .EF control word that ended the last imbed. Each group sets new values for the symbols "state" and "capital". The last time TABLSYM is imbedded, the control word .CS 1 INCLUDE is encountered. This allows the .EF control word in the parent file to be recognized, terminating the table generation. The symbol "bxoff" is set to the word "OFF", so that the last .BX control word will end the box. (The symbol "bxoff" was originally set to null, so that all the .BX control words encountered before the last one merely repeat the same box definition. The actual table looks like this:

STATE	CAPITAL
Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
Arkansas	Little Rock
California	Sacramento

INTERACTIVE SCRIPT/VS PROCESSING

For TSO and CMS, when you use SCRIPT/VS, you do not have to have all of your input text in final form when you issue the SCRIPT command. There are several control words that allow you to interact with SCRIPT/VS as your document is being formatted:

- The .TE [Terminal Input] control word accepts input lines of text or control words as though they were part of an imbedded input file, and processes each line as it is entered.
- The .RV [Read Variable] control word allows you to assign a value to a symbol during SCRIPT/VS processing by entering it at the terminal. When assigning symbol values during SCRIPT/VS processing, the syntax rules for the .SE [Set Symbol] control word _must_ be observed. (See ".SE [Set Symbol]" on page 313 for a description of these syntax rules.)
- The .RD [Read Terminal] control word allows you to type text at a typewriter terminal during SCRIPT/VS output. This control word is useful if you are creating form letters and want to enter names, addresses, or other kinds of variable information directly at the terminal. The text you type is not inserted into the input file and is not processed by SCRIPT/VS.
- The .DF [Define Font] STOP control word allows you to change typing elements at a typewriter terminal.

The .TE [Terminal Input] and .RV [Read Variable] control words are enhanced by using the .TY [Type on Terminal] control word to produce a prompting message, which is displayed at the terminal during SCRIPT/VS processing. The prompting message is not formatted as part of the output.

The .TE [Terminal Input] control word accepts several operands. If you specify (in the input file)

```
.te on
```

SCRIPT/VS reads input lines from the terminal until you type in

```
.te off
```

Then, SCRIPT/VS processing continues with the next line in the file. You can enter SCRIPT/VS control words or text.

You can specify a numeric parameter with the .TE control word. For example, specifying

```
.te 4
```

causes SCRIPT/VS to read four lines from the terminal.

You can also terminate terminal input with the .EF control word, which indicates the end of the current file. The .TE [Terminal Input] control word is essentially an imbed, where the "file" imbedded is the terminal.

The following example uses these control words to process and format the same file an indefinite number of times.

```
...start
.im heading
.ty Enter NAME (1 line)
.rd 1
.ty Enter ADDRESS (2 lines)
.rd 2
.im letter
.ty Any more? (YES or NO)
.rv answer = '
.if /&U'&answer eq /YES .go start
```

The .RV [Read Variable] control word allows one line to be entered at the terminal. It assigns that line the symbol &answer. In the following .IF [If] control word, the uppercase attribute (&U') of the symbol &answer is concatenated to an arbitrary delimiter (/)[39] and is compared to the string "/YES."

Since your response is folded to uppercase, you can enter either "yes" or "YES" and the comparands will be found equal, causing the loop to continue.

If you are planning to format your document for a typewriter terminal that has changeable typing elements, you can tell SCRIPT/VS which parts of the document are to be typed with different elements. You can also instruct SCRIPT/VS when to stop so that the typing elements can be changed. For example, if your document will contain APL characters, you can define your APL typing element as a font by specifying

```
.df apl stop
```

In your text, you would then use the .BF [Begin Font] and .PF [Previous Font] control words wherever you want SCRIPT/VS to stop typing and allow you to change the typing elements. For example, suppose your document contains

```
First, enter the
.bf apl
)load analyze
.pf
command.
```

When the document is formatted to a typewriter terminal, SCRIPT/VS will stop after typing "the" to allow you to change the typing element. When you have changed the typing element, you hit the "ATTN"[40] key and SCRIPT/VS will continue typing the formatted

[39] If you do not enter any text in response to the .RV control word, the value assigned to the symbol &answer is null. When a symbol that can have a null value is used as a comparand with an .IF [If], .AN [And], or .OR [Or] control word, an arbitrary preceding delimiter should be used, as discussed in "Chapter 12. Symbols in Your Document" on page 129.

[40] In CMS, issue the CP command TERM ATTN OFF to suppress the normal CP attention acknowledgment (!) when you hit the "ATTN" key.

document. When SCRIPT/VS stops after "analyze," you can change the typing element again. As before, after you have changed the typing element you hit the "ATTN" key and SCRIPT/VS will continue printing, producing the following results:

First, enter the *)LOAD ANALYZE* command.

Communicating With VM/370

Another useful feature of SCRIPT/VS is the ability to execute CMS or CP commands from CMS SUBSET during SCRIPT/VS processing. To execute a command (or an EXEC procedure or user program), use the .SY [System Command] control word. For example,

```
.sy cp spool printer class s
```

The .SY [System Command] control word is convenient if you ordinarily need to issue several commands before you process a SCRIPT/VS file (you may need certain disks, a particular printer class, as in the above example, and so on). With the .SY [System Command] control word you can put the commands directly in the input file.

If a SCRIPT/VS file imbeds several files from another user's disk, you can include the commands to link to and access the required disks. For example,

```
.sy cp link user2 191 291 rr rpass
.sy access 291 b
.im filea
.im fileb
.sy release 291 (detach
```

When you execute a command during SCRIPT/VS processing, you might not want SCRIPT/VS to continue processing if the command fails. To test the return code from the CMS or CP command, you can check the value of the SCRIPT/VS system symbol, &$RET:

```
.sy exec mysetup
.if &$RET ne 0 .qu
```

If the EXEC procedure MYSETUP completes with a nonzero return code, SCRIPT/VS terminates processing. If the return code is zero, execution continues with the next input line following the .IF control line.

Note: The CMS commands "CP" and "EXEC" are explicitly shown here for clarity. The implied CP (IMPCP) and implied EXEC (IMPEX) functions are not turned off when SCRIPT/VS executes, as they are within an EXEC file.

Communicating With TSO

The .SY [System Command] control word can be used to specify TSO commands and procedures to be executed after SCRIPT/VS completes processing an input file. The commands specified with .SY are passed to TSO for execution in the order they are encountered, provided SCRIPT/VS was not invoked from a TSO CLIST.

For example, the .SY [System Command] control word might be used to display the output file after it has been formatted. To request that the document be sent to an output file, you can specify

```
script infile file('outfile') ...
```

within the file. This causes the output file to be displayed:

```
.sy edit 'outfile' old
```

CHAPTER 12. SYMBOLS IN YOUR DOCUMENT

By using symbols, you can refer to page numbers, variable values, character strings, and control words in your input file. A symbol has a name and a value. When SCRIPT/VS encounters a symbol name, it replaces it with the symbol's current value. After all symbol names in an input line have been replaced with their current values, SCRIPT/VS processes the line.

Define a symbol with the .SE [Set Symbol] control word. For example, to define the symbol &printer, you can issue

```
    .se printer = 'IBM 1403 Printer'
```

Later, you can refer to the symbol "printer" in an input line as "&printer". Each SCRIPT/VS symbol is identified with its prefix, an ampersand (&). The symbol is terminated with either a period (.) or a blank. For example, the input line

```
    Our publisher uses the &printer for output.
```

is processed by SCRIPT/VS and printed as:

```
    Our publisher uses the IBM 1403 Printer for output.
```

but,

```
    Our publisher uses the &printer..
```

is processed as:

```
    Our publisher uses the IBM 1403 Printer.
```

Your document might contain the symbol "&printer" many times, in different places. In the future, when you want the document to describe a different printing device, you can reset the symbol with

```
    .se printer = '3800 Printing Subsystem'
```

At that time, SCRIPT/VS will process your document and substitute the new value for the same symbol:

```
    Our publisher uses the 3800 Printing Subsystem for output.
```

The symbol's name can be up to ten characters long, and may contain upper and lowercase characters, numbers, and the characters @, #, and $.

The symbol's value can be a character string, a numeric value, another symbol, or an arithmetic expression. It can contain compound data items with imbedded blanks and control words. If the symbol's value contains blanks or special characters, enclose the entire value in single quotation marks (as shown in the example above).

Some examples of valid symbol definitions are:

```
    .se corp = 'Scriptographicology, Inc.'
    .se add = 1
    .se incr = &add + 1
    .se mult = &add * 10
    .se test = testa
    .se TEST = testb
```

You can set a symbol to:

- A numeric value:

  ```
      .se number = 25
  ```

- A character string:

```
.se textl = 'IBM 1403 Printer'
```

- A character string that includes a quoted phrase:

  ```
  .se type 'prepared on a 'word processing' machine
  ```

- A SCRIPT/VS control word:

  ```
  .se break = '.br'
  ```

- The value of another symbol:

  ```
  .se printer = '&textl'
  ```

You can perform integer arithmetic with symbols:

- To increment it:

  ```
  .se next = &number + 1
  ```

- To decrement it:

  ```
  .se prev = &number - 1
  ```

- To divide it:

  ```
  .se half = &number / 2
  ```

- To multiply it:

  ```
  .se cost = &number * 20
  ```

- To negate a value:

  ```
  .se negvalue = -&number
  ```

Symbols can also be set using the .RV [Read Variable] control word. The .RV control word allows you to enter symbol values from a terminal during SCRIPT/VS processing in interactive environments. For details, see ".RV [Read Variable]" on page 311.

Symbols can be set to a part of the value of another symbol by using the SUBSTR (substring) parameter of the .SE [Set Symbol] control word. The substring is one or more characters of the character string (the symbol's value). For example,

```
.su off
.se corp = 'Scriptographicology, Inc.'
.se name = substr &corp 1 6
.su on
```

sets the symbol &name to the substring of the value of the symbol "&corp" beginning with character 1 and continuing for 6 characters. Because "&corp" has been previously set to "Scriptographicology, Inc.", this substring results in the symbol &name having the value of the 6-character substring "Script".

In the same manner, the SUBSTR (substring) function can be used to extract characters from a character string that is not another symbol's value. For example,

```
.se name = substr Jonathan 5 4
```

sets the symbol &name to that 4-character substring of "Jonathan" beginning with character 5: the symbol &name will have the value "than". The substring must follow the rules for character string values of a symbol. If the string contains any imbedded blanks or special characters, including arithmetic operators, it must be enclosed in single quotation marks.

You can use the INDEX function of the .SE [Set Symbol] control word to find the location of a string of characters within a symbol value or a string of characters. For example,

```
    .se name = 'Nicola'
    .se location = index &name cola
```

defines the symbol &location to have the value 3, because the string 'cola' starts with the third character of the value of &name (Nicola).

HOW SCRIPT/VS SUBSTITUTES VALUES FOR SYMBOL NAMES

When SCRIPT/VS processes an input line, it first scans for any symbols in the line that require substitution. SCRIPT/VS checks any character string that begins with an ampersand (&) to see if it is a symbol name. When SCRIPT/VS finds a valid symbol, it replaces the symbol's name with its value. A symbol name is terminated either with a blank, a period (.), or the end of the input line. If the symbol name is terminated with a blank, the blank is treated as a normal input character and is left in the input line. If the symbol name is terminated with a period, the symbol value, after substitution, is concatenated with the next input character and the period is removed. Therefore, if a symbol has punctuation immediately after it, you must concatenate the punctuation character to the symbol with a symbol-end period. For example,

```
    This list ends with an &iteml..
```

results in an end-of-sentence period concatenated with the value of the symbol named &iteml. Otherwise, SCRIPT/VS considers a single period as the end-of-symbol indicator and concatenates the symbol with the next character.

You should use this technique when the symbol precedes other punctuation marks or text. For example,

```
    The name of our product is &prodname., which is planned
    for shipment on &shipmo &shipday., 19&shipyr..
```

In this example, values for the symbols are substituted with the adjacent text and punctuation with no intervening blanks. The printed sentence appears as:

```
    The name of our product is Whizbanger, which is planned
    for shipment on 12 September 1982.
```

If you do not place a concatenating period between &prodname and its punctuation (,), SCRIPT/VS regards "&prodname," simply as a character string, and performs no substitution.

You can redefine a symbol as often as necessary in your input file. Each time you redefine the symbol with the .SE [Set Symbol] control word, the new value replaces the old value.

Symbol substitution is performed before the line is evaluated. The result of symbol substitution may cause the original input line to be split into more than one line whether a control word modifier is used or not.

Compound Symbols

When SCRIPT/VS substitutes values for symbol names, it performs as many substitutions as necessary to resolve the symbol name. Because of this, you can use a compound symbol, composed of two or more separately defined symbols. For example, when you define the symbols

```
    .se x = 1
    .se type1 = first
    .se type2 = second
```

the input line

```
    This is the &type&x try.
```

results in:

```
This is the &type1 try. (intermediate result)
This is the first try.
```

Another example of compound symbols is in "Elaborating the System Date" on page 138.

Unresolved Symbols

Sometimes SCRIPT/VS encounters a symbol name that has not yet been defined. In this case, the symbol is unresolved and remains in the input line as a character string that happens to begin with an ampersand. The unresolved symbol is printed on the output page as it appears in the input line.

When you use symbols that are set later in the document than they are referred to (such as a symbol that refers to a page number or a figure number), the symbol will be unresolved when first encountered. When you specify the TWOPASS option with the SCRIPT command, SCRIPT/VS processes the input file twice. As a result, properly defined symbols not resolved during the first formatting pass will be resolved during the second pass.

Inhibiting Substitution

Usually, ampersands that occur in an input file as ordinary text characters are treated as text characters and not as symbol delimiters. The context in which it appears usually prevents the text ampersand from being mistaken for a symbol name. Where a text ampersand precedes a character string that forms a defined symbol name that you want treated as a text character string, there are several ways to inhibit symbol substitution:

- Turn off substitution with the .SU [Substitute Symbol] control word. With the .SU OFF control word, all substitution is turned off. You can turn symbol substitution on again with .SU ON.

- Contrive to make the symbol name unrecognizable by adding punctuation without a delimiting period. For example,

  ```
  I have defined the symbols &AAA, &BBB, &CCC, and others
  for this file.

  The symbol for the day of the month (&SYSDAYOFM) is
  maintained by SCRIPT/VS.

  Use the symbol "&xyz" for this purpose.
  ```

- Translate an unused punctuation mark or special character on your keyboard to the ampersand, and enter the special character in your input whenever you need a text ampersand:

  ```
  .tr ¢ &
  ```

  Because the translation happens after symbol substitution, the text ampersand cannot be mistaken for a symbol-starting ampersand.

- Define a symbol to have the value of an unused hexadecimal code and translate that code to an ampersand. Enter the symbol name in your input whenever you need a text ampersand. The &X' attribute can be used to assign the unused hexadecimal code to a symbol. For example,

  ```
  .se amp = &x'07
  .tr 07 &
  ```

  defines a symbol named "&amp" whose value is the single hexadecimal code 07, and establishes an output translation which maps that hexadecimal code to the character &.

You can use the symbol &amp whenever you want an ampersand to appear.[41]

There are many times when text ampersands are perfectly safe and there is no need to worry about an unexpected substitution. Any time the character string immediately following the ampersand is not a symbol name, no substitution occurs. A character string cannot be a symbol name if:

- It has not been defined as such with a previous .SE [Set Symbol] control word
- It contains a character that would not be allowed in a symbol name (before the first blank or period that ends a symbol name)
- It contains more than ten characters before a blank or period

Canceling a Symbol

When you no longer want to use one of the symbols you've previously defined, you can cancel the symbol:

```
.se oldsymbol off
```

The symbol &oldsymbol will be regarded by SCRIPT/VS as an undefined symbol. It is as though it had never been defined; it is not regarded as a null-value symbol. When you specify

```
.se oldsymbol =
  -or-
.se oldsymbol = ''
```

you redefine the symbol with a "null" value. It exists as a symbol but it has as its value the null string. Note that a null symbol is quite different from an undefined symbol. The null symbol is substituted with a value: the zero-length null string.

Attributes of a Symbol's Value

SCRIPT/VS provides you with the ability to determine some of the characteristics of a symbol in your input file, such as:

- Its existence (&E')
- Its length (&L')
- Its type (&T')
- Its current value (&V')

In addition, you can convert

- A numeric symbol value to a base-26 "number" (that is, a character string: 1 = A, 2 = B, ... 26 = Z, 27 = AA, 28 = AB, ... and so on). (&A' or &a')
- A numeric symbol value to its roman numeral character string equivalent (&R' or &r')
- A lowercase character string to uppercase (&U')
- A character representation of a hexadecimal string to that string (&X')

[41] This technique has been used in marking up this book whenever a text ampersand is required.

&A' converts a number to a character string. The number is converted to a character string that might be thought of as a base-26 number composed of alphabetic letters.

- &A'2 will cause the string B to be substituted.
- &A'26 will cause the string Z to be substituted.
- &A'27 will cause the string AA to be substituted.
- &A'28 will cause the string AB to be substituted.
- &A'705 will cause the string AAC to be substituted.
- &a'28 will cause the string ab to be substituted.

The largest number that can be converted is 65535. Numbers higher than this return a zero.

For both &R' and &A', if the character string to be converted is not a decimal integer number, the result is zero (for example, &R'zorch = 0).

&E' verifies the existence of a symbol. When you use the &E' symbol attribute, the value is substituted with either a 1 or a 0, depending on whether or not the character string following &E' is a defined symbol. For example,

```
.se test = on
The result is &E'&test..
```

results in:

```
The result is 1.
```

If the symbol named &test had not been set, the value of &E'&test would be 0. Any character string that is not a defined symbol name, as in

```
&E'czechoslovakia
```

results in 0.

&L' determines the length of a symbol's value (or the length of any character string, for that matter). For example, after the lines:

```
.se test = 'This is a test.'
.se length = &L'&test
```

the value of &length is 15. If the symbol named &test had not been set, then &length would have a value of 5 (that is, the length of the character string "&test").

&R' converts a decimal number to a roman numeral. The decimal integer number is converted to a character string that represents the number's roman numeral equivalent:

- &R'87 will cause the string LXXXVII to be substituted.
- &R'19&SYSYEAR will cause the string MCMLXXXII to be substituted (in 1982).
- &r'87 will cause the string lxxxvii to be substituted.

The largest number correctly translated to a roman numeral is 3999. For numbers between 4000 and 9999, the character "?" is used to represent the number "5000" or "10000" (for example, &R'6020 = ?MXX and &R'9020 = M?XX). Numbers larger than 9999 are not translated to roman numerals (zero is returned).

&T' analyzes the symbol's type and replaces the character string with:

- N, if the value can be converted to a numeric value that can be used in an arithmetic expression, or
- C, if the value contains nonnumeric data (Characters).

The "N" or "C" that SCRIPT/VS sets is always in uppercase. For example,

```
    &T'1978
```

is replaced with "N", but

```
    &T'DAD
```

is replaced with "C".

&U' converts lowercase characters to uppercase. For example,

```
    &U'hello
```

results in:

```
    HELLO
```

&V' returns the current value of the symbol (as it was last set), without any further substitution. An undefined symbol or a character string has no value attribute, (that is, a value attribute of nothing). For example,

```
    .se a = '&b.linda'
    .se b = 'Be'
```

An occurrence of &a will be substituted with "Belinda" and its length is 7 (however, &L'&a = 8). An occurrence of &V'&a will be substituted with "&b.linda".

Attribute symbol prefixes can be combined. For example, &L'&V'&a is the length of the value of the symbol &a, which is 8.

Note that &V' returns a <u>character string</u> that represents the current value of the symbol as previously set. In other words, a defined symbol has a value; character strings and undefined symbols do not have a value (that is, a character string's "value" is null). For example,

```
    .se a = '&c.linda'
```

- &V'&a yields the character string "&c.linda". The ampersand and the period in "&c." are merely text characters, not symbol delimiters, for this value substitution.
- &V'&c.linda yields the character string "linda". In this case, the ampersand and the period in "&c." act as symbol delimiters. The value of the symbol "&c" is concatenated to the character string "linda". Since "&c" is not a defined symbol, it has no "value."
- &V'&V'&a yields either of two results, depending upon whether or not substitution tracing is in effect from the .IT [Input Trace] control word. Let's see why:

  If substitution tracing is off, &V'&V'&a yields the null string. &V'&a yields the <u>character string</u> "&c.linda", as shown above, as an intermediate result. The value of this character string is the null string.

  If substitution tracing is on, &V'&V'&a yields the character string "linda". &V'&a yields the intermediate result "&c.linda", but in this case substitution stops at this point so that the intermediate result can be traced. After tracing, the string "&V'&c.linda" is evaluated as a separate operation. The ampersand and the period in "&c." now act as symbol delimiters, causing the value of the symbol "&c", which is null, to be concatenated to the string "linda".

Attributes apply only to the symbol (or character string) immediately following them, up to the next delimiter (period or blank). For example,

```
.se a = '&J'
.se b = 'K'
.se JK = 'TIMOTHY'
```

The string &a.&b resolves to "TIMOTHY", since &a.&b resolves to "&J&b", then to "&JK", and finally to "TIMOTHY". However, the string &L'&a.&b results in "2K", because "&L'&a" is evaluated first. SCRIPT/VS provides a length of 2 (for the symbol's value: "&J"), and concatenates the "2" with the character "K". &L'&a&b results in "3", because "&b" is evaluated first and the length SCRIPT/VS provides is the length of the character string "&aK" (because a symbol with that name hasn't been defined in the example).

The symbol attribute names &E', &L', &T', and &V' can be specified, to produce the same result, in either uppercase or lowercase. That is, &L' and &l' will both return the length of a symbol.

However, the symbol attribute &R', which converts a numeric value to roman numerals, and &A', which converts a numeric value to an alphabetic character string, have different meanings when specified in uppercase and lowercase.

&X' converts a character representation of a hexadecimal string into that string. Hexadecimal codes that do not have common keyboard assignments can be entered with the &X' attribute. For example, the bullet character (hexadecimal AF) can be entered by specifying

```
&x'af Step one:
```

This results in the formatted line

```
• Step one:
```

The &X' attribute converts the hexadecimal code "af" to a bullet. The hexadecimal code must contain an even number of hexadecimal digits (0 to 9 and A to F). For example,

```
DATA&x'afad.TRANSFORM&x'b2bdbe.0
```

results in

DATA•[TRANSFORM^{2}]≠0

If an even number of hexadecimal digits is not specified, or an invalid hexadecimal digit is encountered, the value of the 0 attribute is zero.

SYMBOL AND MACRO LIBRARIES

If a symbol cannot be resolved from a definition that has been set with the .SE [Set Symbol] control word, SCRIPT/VS can look for a definition in a library.

A symbol and macro library is a partitioned data set. In CMS, a library is a file whose filetype is MACLIB, which is a CMS simulated partitioned data set. Each symbol definition in the library is a one-line member whose member name is the symbol name. (Macro definitions can reside in the same library, and may occupy as many lines as required.) In ATMS-III, a "library member" is a document or a subdocument.

No trailing delimiter is required of symbols in a library. However, for compatibility with Release 1, the last nonblank character of a symbol value found in a library will be recognized as a delimiter and deleted if, and only if, it is a colon (:). If you want a colon to be the last character of such a symbol value, add another colon as a delimiter.

Before searching a library for a symbol, SCRIPT/VS translates the symbol name to uppercase characters. Even though SCRIPT/VS recognizes the symbols "&libsym" and "&LIBSYM" as separate and unique symbols, the library doesn't. Member names in the library are always in uppercase. Therefore, the symbol names "libsym" and "LIBSYM", even though they are different, will be set from the same library member. You can use the library in two ways:

- To explicitly set a symbol name by declaring that its definition is in a library:

  ```
  .se para lib
  ```

  SCRIPT/VS searches the library specified by the LIB option of the SCRIPT command for the definition of &para (member PARA) and sets it in the SCRIPT/VS symbol table.

- To set an unresolved symbol. During substitution, the library can be searched for a definition of an unknown symbol with the same name as the symbol (when converted to uppercase) only when .LY ON or .LY SYM is specified. If found in the library, the symbol is defined in the SCRIPT/VS symbol table.

When you are sure that none of your symbols are defined in a symbol library, you can issue the .LY [Library] control word to prevent library searches for unresolved symbols. (The initial setting is OFF. You have to specify .LY ON or .LY SYM to search a library for undefined symbols.)

The .LY OFF control word prevents all library searches, for unresolved macros as well as for symbols. The .LY MAC control word allows library searches for unresolved macros.

Note: When you specify that a symbol's definition is in the symbol library with

```
.se libsym lib
```

the current .LY [Library] control word specification is ignored. In the above example, the library is searched to find a definition for &libsym. Remember, the symbol name is translated to uppercase before searching the library.

SCRIPT/VS SYSTEM SYMBOLS

There are several groups of system symbol names that are initialized and recognized by SCRIPT/VS:

- Symbols you can use to obtain the current date and time.
- Symbols you can use to obtain current values of SCRIPT/VS formatting parameters: the current line length, left margin indention, and page length, to name a few.
- The symbol set as a return code from the latest CMS command executed using the .SY [System Command] control word.

A complete list of system symbols is given in Figure 35 on page 361.

Some of the system symbols begin with "&$". These symbols cannot be changed with a .SE [Set Symbol] control word, because they are reserved and contain SCRIPT/VS formatting parameters and controls. Most of the special symbols reflect values under your control. You can change them with the appropriate control word or command option, but not with the .SE [Set Symbol] control word. All symbols that begin with a $ may be entered in either upper- or lowercase, including any non-system symbols that you define.

All other system symbols (those that do not begin with "&$") can be manipulated and modified by .SE [Set Symbol] control words within the input file.

Symbols for the System Date and Time

The symbol names for date and time values that are maintained by the system are:

Symbol Name	Description	Value Range
&SYSYEAR	Year	00-99
&SYSMONTH	Month	01-12
&SYSDAYOFM	Day of the month	01-31
&SYSDAYOFW	Day of the week	1-7 (1 = Sunday)
&SYSDAYOFY	Day of the year	001-366
&SYSHOUR	Hour of the day	00-23
&SYSMINUTE	Minute of the hour	00-59
&SYSSECOND	Second of the minute	00-59

The date and time values are set once and stay in effect throughout the processing of the file. You can use these symbol names to set symbol values for the date and time yourself.

No punctuation is provided by SCRIPT/VS for combining these values. You must supply it yourself when combining them. For example, to obtain the current date and time for printing on your output pages, you might enter:

```
DATE: &SYSMONTH./&SYSDAYOFM./&SYSYEAR
TIME: &SYSHOUR.:&SYSMINUTE.:&SYSSECOND
```

Notes:

- The date and time symbol names must be specified with all uppercase characters.

- Leading zeros are provided with the symbol value whenever appropriate. For example, on the eighth day of the month the value of &SYSDAYOFM is set to "08", rather than to "8". To suppress leading zeros, you can reset the symbol with the following arithmetic expression before you refer to it:

  ```
  .se SYSDAYOFM = &SYSDAYOFM + 0
  ```

  The symbol &SYSDAYOFM will be redefined, for your input file only, without leading zeros. SCRIPT/VS removes leading zeros from the result of arithmetic expressions on the right-hand side of the equal sign in .SE control words.

Elaborating the System Date

If you want to print the date with the names of the months and days, your output page can include the date in the form

    Tuesday, August 31, 1982.

This requires a group of .SE [Set Symbol] control words using the reserved symbols in compound expressions, as follows:

```
.se d1 = Sunday
.se d2 = Monday
.se d3 = Tuesday
     .
     .
     .
.se d7 = Saturday
.se m01 = January
.se m02 = February
     .
     .
     .
.se m12 = December
```

To eliminate the leading zero of &SYSDAYOFM, include

```
.se SYSDAYOFM = &SYSDAYOFM + 0
```

Leading zeros that occur with the other symbols do not present a problem in this example.

The symbolic input line might be:

```
&d&SYSDAYOFW.., &m&SYSMONTH.. &SYSDAYOFM., 19&SYSYEAR..
```

which results in:

```
Tuesday, August 31, 1982.
```

Notice the ending delimiters for the compound symbols &d&SYSDAYOFW and 19&SYSYEAR in the above:

- "&d&SYSDAYOFW..," ends with two periods to prevent the symbol name from being concatenated with the comma and to allow its value to be concatenated with the comma. This compound symbol requires two stages of substitution to be resolved. &SYSDAYOFW ends with the first period. When resolved, the symbol &d3 ends with the second period. In this way, the comma needed for punctuation is concatenated with the name of the weekday.

- 19&SYSYEAR is not a compound symbol. It is resolved with only one stage of substitution. The character string "19" is concatenated with the symbol "&SYSYEAR". The first period ends the symbol &SYSYEAR. The second period is needed (in this example) for punctuation, and is concatenated with the value of the year.

Symbols for SCRIPT/VS Control Values

SCRIPT/VS allows you to examine the formatting parameter values it uses when processing your input file. Some of the values change dynamically. You can obtain the parameter's current value by using the system symbols.

The symbols that represent SCRIPT/VS internal formatting parameters cannot be set by .SE control words in your input file. The name of each of the following reserved symbols begins with "&$" and can be specified using either lowercase or uppercase characters. The system symbols are listed and described in Figure 35 on page 361.

You can use this technique to ensure proper results even though some formatting parameters can change dynamically. For example, the following sequence produces a box the width of the output page:

```
    .se indent = &$IN
    .if &indent eq 0 .se indent = 1
    .bx &indent &$CL
    .in +2m
    .ir +2m
    The .BX control word begins
    a box structure ....
    .bx off
```

which results in

```
The .BX control word begins a box structure using the
current margins. The .IN [Indent] and .IR [Indent Right]
control words shift the margins to position the text within
the box. After the text is processed, the original values
are restored.
```

As another example, you might want to leave a blank page with only a figure caption at the bottom of a single column page. Perhaps the file is to be printed within different master files, each of which requires a different page length. You might code the following sequence:

```
    .pa
    .se lines = &$LC - 1
    .sp &lines
    Figure x. Sample Output
```

You will find that these special symbols can be especially useful when writing SCRIPT/VS macros, or for testing the current environment using the .IF control word family.

The &$RET Special Symbol

In CMS, the &$RET special symbol contains the return code from the CMS or CP command that was most recently executed as a result of a .SY [System Command] control word. You can examine the return code and take conditional action based on its value. For example, the following sequence will imbed a file named OPTDATA only after ensuring that the file exists:

```
    .sy state optdata script *
    .if &$RET eq 0 .im optdata
```

In TSO, &$RET is set to "0" by the .SY [System Command] control word to indicate that the command was stacked for execution after SCRIPT/VS terminates.

In ATMS-III, &$RET is always set to 0, indicating that the .SY [System Command] control word is ignored by ATMS-III.

In batch, &$RET is set to "-3" to indicate that the .SY [System Command] control word is not supported.

The &$LC Special Symbol

The &$LC special symbol contains the number of lines left in the column at the time of symbol substitution. This value does not include running titles, headings, or footings which have been placed on the page, nor does it include keeps, widows, or partially filled output lines which have not been placed in the column at the time of symbol substitution.

The value of &$LC at the time of symbol substitution may not accurately reflect the final position of surrounding text on the page if that text is in a keep, float, or widow, if there is a partially filled output line, or if column balancing is in effect. The value of &$LC at the time of symbol substitution will accurately reflect the final position on the page of text only at the beginning of a new page, section, or column.

PASSING PARAMETERS TO INPUT FILES

SCRIPT/VS has three sets of symbols that are set automatically by parameters passed to a file or macro. These are:

- SCRIPT/VS system symbols, which can be set when the SCRIPT command is issued
- Parameters passed to imbedded files with the .IM [Imbed] and .AP [Append] control words
- Parameters passed to a macro

Setting Symbols with the SCRIPT Command

Use the SYSVAR option of the SCRIPT command when you want to pass values to the input file from the SCRIPT command line.

The symbols that you can set with the SYSVAR option have names starting with "&SYSVAR" appended to one alphameric character: 0 through 9, uppercase A through Z, and @, #, and $. For example,

```
script outline ( sysvar ( a atype 2 nogo
```

This command line sets the symbols &SYSVARA to ATYPE and &SYSVAR2 to NOGO. Lowercase letters assigned to an &SYSVAR symbol are translated to uppercase letters. Consequently, when you include the symbols in an input line, always use the uppercase symbol name and character-string values.

For example, &SYSVARA may be used to bypass parts of the document and &SYSVAR2 may be used to terminate processing before completion:

```
.if &SYSVARA eq ATYPE .go aproc
.if &SYSVAR2 eq NOGO .qu
...aproc
```

When you use &SYSVAR symbols, it is good practice to put comments at the beginning of your input file so that other users who process the file are aware of each &SYSVAR symbol and the meanings of its values.

For details about the SYSVAR option of the SCRIPT command, see "Chapter 2. Using the SCRIPT Command" on page 13.

Symbols Set When a File Is Imbedded or Appended

You can pass parameters to an imbedded or appended file with the .IM [Imbed] and .AP [Append] control words. The symbols &0 through &14 are set to the parameters following the name of the imbedded file. For example,

```
.im finance George 125 $21.50 '18-7'
```

When the file named FINANCE is imbedded, the symbols &0 through &4 are automatically set by SCRIPT/VS:

Symbol Name	Value Set by SCRIPT/VS
&0	4
&1	George
&2	125
&3	$21.50
&4	18-7

Symbol &0 contains the number of parameters passed. Up to 14 parameters can be passed when a file is imbedded or appended. These parameters are called "tokens." Each token can be up to eight characters long, delimited with blanks. The rules that apply to setting the value of a symbol also apply to specifying a token. See "Chapter 11. Combining SCRIPT/VS Files" on page 119 for details about imbedding and appending files.

Symbols Set When a Macro Is Processed

You can pass parameters to a macro when your input file calls the macro. The parameters become macro-local symbols (that is, symbols that are set for the called macro only; not for other macro calls that occur within the called macro). The format of the macro call might be:

```
.burger fries+shake nosauce 'on a great big poppy-seed bun'
```

When the macro BURGER is processed, local symbols within it are automatically set by SCRIPT/VS:

Symbol Name	Value set by SCRIPT/VS
&*	fries+shake nosauce 'on a great big poppy-seed bun'
&*0	5
&*1	fries
&*2	+
&*3	shake
&*4	nosauce
&*5	on a great big poppy-seed bun

Symbol &* contains the entire untokenized input line. It contains all leading blanks after the blank that delimits the macro name. Symbol &*0 contains the number of symbol values passed. The symbols &*1 through &*n contain the individual tokens passed to the macro. Notice that blanks, arithmetic operators, and parentheses normally delimit tokens, but that a single token can contain these and other special characters if it is enclosed in single quotation marks. Also, macro tokens are not subject to the 8-character limit applied to .IM [Imbed] and .AP [Append] tokens. See "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147 for details about specifying symbols within macro instructions.

Note: Symbols whose names begin with an asterisk (*) are treated differently from other symbols. Other symbols are globally available to all files and macros, but symbols whose names begin with an asterisk (*) are local to a particular macro at a particular level of nesting. The symbols "&*" and "&*0" through "&*n" that are used to pass tokens to a macro are "macro-local symbols." Each time a macro is called, a new set of macro-local symbols is established for it. The set lasts until the macro completes.

Unlike other symbols, macro-local symbols, when undefined, are replaced during symbol substitution with the null string.

SETTING THE CURRENT PAGE NUMBER

You can set a symbol to be equal to the value of the current page number when the .SE [Set Symbol] control word is encountered. For example,

```
.se pagenum = &
```

A single ampersand on the right-hand side of the equal sign of a .SE control word is replaced with the character string of the current page number, including its prefix, if any. Elsewhere in your document, you can refer to the page number with its symbol name. To continue the example,

```
For details, see page &pagenum..
```

Whenever the &pagenum symbol occurs in your document, SCRIPT/VS replaces it with whatever the page number was when the .SE [Set Symbol] control word was processed. If the symbol is set before the page is started, the page number will be the same as that of the previous page and not that of the next page. At the start of the document, the page number is 0.

SYMBOLS FOR ARRAYS OF VALUES

An array symbol is a special type of symbol that allows you to assign many values to the same symbol name. Each individual element of the array has, in addition to the name, an element number in parentheses. The element number is also called the index or subscript of the element. When you format your document for output, the entire array of values can be referred to by a single symbol name. An array symbol is defined with the .SE [Set Symbol] control word. For example,

```
     .se name() = value
```

The parentheses indicate that this is an element of an array and "value" is any expression that can legally appear on a .SE [Set Symbol] control word line. The notation () is a shorthand way to specify the "next" element of the array.

When SCRIPT/VS encounters the array symbol value in the form:

```
     &name(*)
```

it replaces "&name(*)" with the values of all the currently defined array elements, in the order in which they are indexed. A comma and a blank separate the individual elements. You can specify different array separator characters using the .DC [Define Character] ASEP control word (for details, see ".DC [Define Character]" on page 241).

When the output line is too long because of the expansion of an array symbol, the line's first part is used as one output line and the remainder is printed on the next output line.

You can also cancel an array symbol by using the OFF parameter of the .SE [Set Symbol] control word. If the symbol is an array symbol and no subscript is provided, the entire array is cancelled.

Controlling the Array Elements

Each element in an array has a value associated with it. You can refer to any element of the array with the array's symbol name and the element's index number in the form

```
     &name(n)
```

where "n" is the positive integer that identifies the position of the element within the array.

An array symbol reference can be used anywhere that a nonarray symbol can be used. If the element "n" exists in the array, its value is substituted just as a normal symbol's value would be. If the symbol exists but has no element "n", a null value is substituted. If the symbol is not defined at all, the symbol is treated as an undefined symbol.

You can specify which array element you wish to set by including a number (identifying its location within the array) within the parentheses. For example, the input line

```
     .se list(1) = &
```

sets element number 1 of the array with the current page number. When you list all the elements of the array, this entry will be listed first, even if it is not the first one set. Here's another example:

```
.se name(1) = 1
.se name(47) = 2
.se name(25) = 3
.se name(2) = 4
.se name(3) = 5
```

The expression

```
&name(*)
```

results in "&name(*)" being substituted as follows:

```
1, 4, 5, 3, 2
```

In other words, SCRIPT/VS places the array element values in ascending element index order, not in the order in which they were defined. In this example, there are many available but undefined element numbers in between those that are defined. Any undefined elements in an array are ignored when the array's values are substituted.

The array element number can be another symbol. For example,

```
.se elem = 1
.se array(&elem) = &
```

No blanks may appear between the symbol name and index. When array symbols are used on the right-hand side of a .SE [Set Symbol] control word expression and symbol substitution is off, symbols used as array subscripts must be simple, not compound, symbols.

Accessing the Index Counter

Every array has an element zero, represented by the symbol name

```
&name(0)
```

Element zero is an index counter that indicates the last element used. It tells SCRIPT/VS which element to set next if you didn't specify one.

Note: When the TWOPASS option of the SCRIPT command is specified, all array index counters are reset to zero for the second pass.

Setting the Index Counter

The expression "name()" is treated as an index counter as well as a symbolic expression. Each time SCRIPT/VS encounters the expression, it assumes that the next element of the array is to be filled. If you never specify a number within the parentheses of an array symbol, SCRIPT/VS begins numbering with element 1.

It is possible to set the initial value of the array index counter, as follows:

```
.se name(0) = n
```

where n is any nonnegative integer. Then, the first occurrence of ".se name()", with no element specified, would be equivalent to ".se name(n+1)" and the counter would be incremented from there.

In this way, you can start the automatic indexing of an array at element 5, for example, and reserve elements 1 through 4 for explicitly specified definitions.

If you do not set the index counter explicitly, it will be incremented from the index value of the element last set. For example,

```
.se name() = first
.se name(3) = second
.se name() = third
```

The first element of the array is set to the value "first", element 2 has a null value, element 3 has the value "second", and element 4 has the value "third".

For substitution of arrays, you can make SCRIPT/VS substitute all elements of the array (except element zero), or you can make it substitute just a single element.

The notation &name(5) causes only element 5 to be substituted. The notation &name(*) causes all elements of the array to be substituted, as previously described.

Any symbol is potentially an array symbol. The symbol &XYZ, for example, is actually element zero of a possible array. &XYZ(0) refers to the same symbolic value as &XYZ. If, after using a symbol like &XYZ, you set another element with:

```
.se XYZ(5) = 'last letters'
```

be careful about the value previously set in element zero (that is, in symbol &XYZ). If the value is not a number, you will get an error message if you ever use the shorthand notation where element zero is supposed to contain the current index.

EXTENDED SYMBOL PROCESSING

A control word can be placed anywhere in an input line as long as it is preceded by the control word separator and a period. You can also invoke a control word or a macro at any point in the input line by setting it as the value of a symbol. This symbol value must also be preceded by a control word separator. When a symbol value begins with the control word separator (;), the rest of the value is treated as though it began a new line. Therefore, a control word that is set as the value of a symbol is processed by SCRIPT/VS as though it were a control word that started in the first character position, even when it occurs in the middle of a text input line. For example, the .BR [Break] control word, defined as the symbol &BR

```
.'se BR = ';.br ;'
```

causes SCRIPT/VS to interpret the symbol "&BR" as though you had a new input line starting with ".br ;". (Because the value of the symbol contains a control word separator, the .SE [Set Symbol] control word is entered with the control word modifier (') to inhibit control word separator scanning for that input line. The control word modifier is described in "Chapter 23. SCRIPT/VS Control Word Descriptions" on page 219.) Thus, the input line

```
This is line one.&BR.This is line two.&BR.
```

is formatted as though it were the following four input lines:

```
This is line one.
.br
This is line two.
.br
```

Note: The control word modifier was used here to set up the symbol 'BR' that contained control word separators. The extended symbol processing rule described here takes effect during substitution and not during control word processing.

Substitution occurs before SCRIPT/VS has classified the line as a control word line or a text line, thus a control word modifier can not prevent the symbol substitution processor from recognizing a control word separator.

The input line

```
.ce Note this; The symbol &BR starts with a semicolon.
```

is formatted as the following four lines:

```
.ce Note this
  The symbol
.br
starts with a semicolon.
```

The extended symbol substitution rule only divided the line into three parts. The first part was a control word line (.CE ...) that was later split into two lines by the control word separator rule.

The input line

```
.'ce Note this; The symbol &BR starts with a semicolon.
```

is formatted as the following three lines:

```
.'ce Note this; The symbol
.br
starts with a semicolon.
```

The control word modifier only suppressed the control word separator rule for the first line after symbol substitution was completed.

CHAPTER 13. WRITING SCRIPT/VS MACRO INSTRUCTIONS

SCRIPT/VS allows you to define your own processing controls, called macro instructions. The contents of a macro instruction can consist of SCRIPT/VS control words, GML markup, symbols, text lines, and other macros.

You can define macros for GML processing, to provide additional formatting controls, or to modify the action taken by a SCRIPT/VS control word.

To process macros, you must explicitly specify .MS [Macro Substitution] ON in your document before SCRIPT/VS encounters any of the macros. If SCRIPT/VS encounters a macro when macro substitution is off, the macro will be treated as an invalid control word.

WHEN SHOULD YOU USE MACROS?

Many macro-like functions can be performed by symbols that are defined as control word strings. Sometimes, though, you may need to define a macro to perform a function that symbol processing alone cannot provide. For example, the control word sequence

```
.se x = &x + 1;.se y = &x
```

is intended to increment the symbols x and y. But because SCRIPT/VS performs symbol substitution before control word execution, &y is set equal to the current value of &x and only &x is incremented.

You can perform this sequence properly by defining a macro. For example,

```
.dm increment /.se x = &x + 1 /.se y = &x
```

After SCRIPT/VS processes the macro

```
.increment
```

&x and &y have equal values, since the two .SE [Set Symbol] control words are processed sequentially.

Macros also allow you to redefine the meaning of SCRIPT/VS control words. For example, you can use the macro facility to define new head levels or redefine the existing ones. Although seven head levels are provided with SCRIPT/VS, you might want to define additional head levels.

HOW TO DEFINE A MACRO

Use the .DM [Define Macro] control word to define macros. Since SCRIPT/VS processes macros as control words, an undefined SCRIPT/VS macro is treated as an invalid control word.

When you define a SCRIPT/VS macro, you must name the macro and specify the input lines to be processed whenever the macro is called. For example, you can write the following macro to redefine the .PP [Paragraph Start] control word:

```
.dm pp /.sk /.in 3 for 2 /&*
```

The macro definition elements (usually control words) are separated by delimiters. The delimiter is the first nonblank character that follows the blank after the macro name. It can be any character that does not appear in the line itself.

The symbol &* represents the entire macro argument (that is, the line passed to the macro for processing). For example, when the input line

```
     .pp On second thought,
```

is processed, &* has a value of "On second thought,".

The form of the .DM [Define Macro] control word shown above is restricted to one input line. The input line is broken at delimiter characters into separate macro lines, and each component becomes a separate line of the macro.

The subscripted form of the .DM [Define Macro] control word allows you to define macro lines on separate input lines.[42] For example, you could redefine the .PP [Paragraph Start] control word as follows:

```
     .dm pp(5) /.sk
     .dm pp(10) /.in 3 for 2
     .dm pp(15) /&*
```

The macro line number in parentheses is also called the subscript. If the number is omitted from the parentheses, SCRIPT/VS will automatically use an increment of 10, starting at 10. Macro line numbers, if included, do not have to be defined in any particular order, nor do they have to be sequential numbers. However, when the macro is used, it is executed in subscript sequence, which is not necessarily the sequence in which the macro lines were entered. The subscripted form of the .DM control word can be used to modify individual lines of a macro without having to respecify the entire macro definition. For example, to increase the indention caused by the previously defined .PP macro, you can issue:

```
     .dm pp(10) /.in 5 for 1
```

or you can cause the .PP macro to start an inline keep by specifying

```
     .dm pp(12) /.kp 6
```

Do not mix the two forms of the .DM control word. If you use the subscripted form you must specify only one macro line with the control word.

Conditional Macro Processing

Macros can be defined to conditionally format a document using the .IF [If] control word family. For example, you may have a series of input files that contain information for several people, none of whom require all of the information. You can define a macro which will execute certain control words only if the document is being formatted for specific individuals:

```
     .dm maybe() /.if &who eq Franz .or &who eq Geoff
     .dm maybe() /.th &*
```

When you specify

```
     .maybe .im sg$sym
```

the file SG$SYM will be imbedded only if the document is being formatted for either Franz or Geoff.

You can also use conditional processing to highlight lines of text differently depending upon the device for which the document is being formatted. For example, the following macro will underscore a line of text, unless the document is being formatted for a 3270 terminal:

[42] Each macro line can, of course, contain several control words, separated by control word separators.

```
.dm hilite() /.sk
.dm hilite() /.if &$PDEV eq 3270
.dm hilite() /.th .up &*
.dm hilite() /.el .us &*
.dm hilite() /.sk
```

You can use the .GO [Goto] control word to instruct SCRIPT/VS to branch to another portion of your macro on certain conditions. For example, to process each token specified with a macro invocation separately in an inner macro, you might define the outer macro as:

```
.dm macro() /.se *i = 1
.dm macro() /...loop
.dm macro() /.process &*&*i
.dm macro() /.se *i = &*i + 1
.dm macro() /.if &*i le &*0 .go loop
```

Macro Naming Conventions

A macro name can be up to 10 characters long, without imbedded blanks or special characters, and is not case sensitive. The name can be the same as the two-letter name of a control word, in which case its definition supercedes the function of the control word. When you enter a macro name as part of your input file (after you've defined it), enter it as though it were a control word, with a period in column 1.

Local Symbols for Macros

In SCRIPT/VS, most of the input to be processed is text. The text can contain any character string, including strings that look like control words or symbols. Macros, control words, and symbols are merely character strings that have special meaning based on the context in which they occur. Therefore, an undefined symbol is regarded by SCRIPT/VS as an ordinary character string.

Within macros, symbols can be defined with an asterisk (*) as the first character of the symbol name. Such symbols are "local" to the macro in which they are defined: They are recognized only within that macro and, unlike ordinary symbols, if they are undefined, they have a null value. You can use a different set of local symbols for each macro, and for each occurrence of a macro call.

For symbol substitution within a macro, the following rules apply:

- All global symbols are considered text character strings if undefined as symbols.
- All local symbols are considered null if not defined.

When SCRIPT/VS processes a macro, it assigns values to certain designated local symbols based on the macro's input text line. The local symbols are named &*0, &*1, &*2, and so on. Values are assigned to a new set of local symbols each time a macro is called.

The symbol &* contains the entire character string on the macro's input line (except for the macro name). The symbol &*0 represents the number of words that make up the character string. The symbol &*1 contains the first word, the symbol &*2 contains the second word, and so on. For example, when SCRIPT/VS encounters the following input line

```
.process fileb 10 filea no
```

it sets the following values for the macro's local symbol values (&*, and &*1 through &*n are called "tokens"):

Symbol	Value
&*	fileb 10 filea no
&*0	4
&*1	fileb
&*2	10
&*3	filea
&*4	no
&*5-&*n	(null value)

When a macro symbol beginning with &* is not set by the input line, it has a null value. When you want to assign a null value to a macro symbol without also assigning null values to all subsequent tokens on the input line, use the percent sign (%) to represent the null-value token. For example, the macro input line

```
    .insert filea 10 % fileb 15
```

results in the symbols being set as:

Symbol	Value
&*	filea 10 % fileb 15
&*0	4
&*1	filea
&*2	10
&*3	(null value)
&*4	fileb
&*5	15
&*6-&*n	(null value)

You can set any symbol with a name that begins with the character "*". A symbol so named is considered a local symbol for the macro whose definition includes it. Such symbols are known only to the macro that defines them. The symbol values are saved when the macro calls another macro, and are restored when the called macro returns to the calling macro. A different set of local symbols is set each time a macro is called, plus another set for when no macro is the current source.

Note: Undefined local symbols are replaced with null values only when the current input source is a macro.

Terminating a Macro

Ordinarily, execution of a macro ends after the last line of the macro has been executed; control returns to the file or macro which invoked the macro.

The .ME [Macro Exit] control word may be used to end execution of a macro prematurely:

```
    .dm score() /.sk 1
    .dm score() /.if &place eq inline .me
    .dm score() /.sx ||-||
    .dm score() /.sk 1
```

If the value of the symbol &place is "inline", the .ME control word causes control to return immediately to the macro's caller, without executing the remainder of the macro.

If the remainder of an input line containing a .ME [Macro Exit] control word is not null, it is saved until after the macro is closed and executed as if it had been part of the macro or file which invoked the macro. This allows a macro to set its caller's local symbols. For example,

```
    .dm macro() / ...
    .dm macro() /.me .se *rc = 4
```

Here the .ME control word's function of prematurely ending the macro is superfluous, since it is the last line of the macro. The remainder of the line, however, is saved and executed as if it had been part of the macro's caller, and results in the setting of a macro local symbol.

The .ME [Macro Exit] control word also allows you to create a "computed GOTO" facility:

```
.dm case() /.se *i = &*1 + 1
.dm case() /.if &*i gt &*0 .mg ||CASE index error.|
.dm case() /.el .me .go &*&*i
```

The CASE macro may be invoked with an index, and a list of labels:

```
.case &function open read write close
```

The CASE macro will use the index to select one of the labels and return a .GO [Goto] control word for that label to its caller.

Redefining SCRIPT/VS Control Words

You can define a macro with the same name as a control word to effectively redefine it, to revise it, or to supplement its function. The definition you code with the .DM [Define Macro] control word is used instead of the SCRIPT/VS-defined function. If you redefine a control word as a macro, the new definition is effective whenever the control word is encountered as long as macro substitution is on (.MS ON), or whenever the macro is called using the .EM [Execute Macro] control word.

When macro substitution is on, you can still specify that a SCRIPT/VS control word function is to be executed, even when a macro of the same name is defined, by using the .EC [Execute Control] control word or the control word modifier. For example, the input line

```
.dm sk /.sp &* /.il 5 /
```

redefines the .SK [Skip] control word, to space lines and indent the first output line after the line space.

When you want the .SK [Skip] control word to be executed but do not want to turn off macro substitution, issue

```
.ec .sk 4      -or-      .'sk 4
```

to skip four lines without indenting the next output line.

When macro substitution is off (.MS OFF) and you want to execute a macro (whether or not the macro's name is the same as a SCRIPT/VS control word), use the .EM [Execute Macro] control word. For example,

```
.em .sk 3
```

results in three line spaces, with the next output line indented five spaces.

Note: When you redefine a SCRIPT/VS control word with a macro of the same name:

- Be sure to define all the functions, implicit as well as explicit, that you want. The macro definition <u>does not</u> modify the control word function; it is used, as a macro, <u>instead of</u> the control word function.
- To make the macro definition effective:
  - Turn macro substitution on (.MS ON), or
  - Use the .EM [Execute Macro] control word to execute the macro definition.
- When the macro definition includes the SCRIPT/VS control word of the same name, use the .EC [Execute Control] control word to specify the control word. An example of this technique is in the following section, "Avoiding an Endless Loop."

Avoiding an Endless Loop

When you define a macro to replace the function of a SCRIPT/VS control word, you might have to turn macro substitution off to avoid an endless loop. For example, you want to redefine the .PP [Paragraph Start] control word to put two line spaces between paragraphs instead of one:

```
.dm pp() /.sk
.dm pp() /.ms off
.dm pp() /.pp &*
.dm pp() /.ms on
```

By turning macro substitution off with the .MS OFF control word, statement 3 invokes the .PP control word, rather than reinvoking the .PP macro.

Sometimes turning off macro substitution is not an adequate solution to the problem of an endless loop. For example, you can take over the .IM [Imbed] control word and cause the name of the imbedded file to be typed whenever it is imbedded by defining a .IM macro:

```
.dm im() /.ty Imbedding &* . . .
.dm im() /.ms off
.dm im() /.im &*
.dm im() /.ms on
```

Macro substitution is turned off to prevent an endless loop from occurring. However, when macro substitution is turned off, substitution is prevented for any macro that might be part of the imbedded file (as well as files it might imbed).

Instead, use the .EC [Execute Control] control word to tell SCRIPT/VS that the input line is to be treated as a control word even though a macro of the same name might be defined. For example, the following lines

```
.dm im() /.ty Imbedding &* . . .
.dm im() /.ec .im &*
```

redefine the .IM [Imbed] control word, preventing an endless loop while still allowing for macro substitution in the imbedded file.

HOW VALUES ARE SUBSTITUTED FOR SYMBOLS WITHIN A MACRO DEFINITION

When symbol substitution is on, the .DM [Define Macro] control word line is scanned for symbol names. If you define a macro that contains a symbol, you usually want the symbol's value substituted for the symbol name when the macro is encountered as an input line, rather than when the macro is defined. Therefore, turn off symbol substitution (using the .SU OFF control word) before you define the macro, to allow the symbol (rather than its value when the macro is defined) to be part of the macro definition. For example,

```
.su off
.dm of /.sk/.in &off after 1/
.su on
```

In this example, &off is a symbol that might have a value when SCRIPT/VS processes the .DM [Define Macro] control word. If substitution is ON, the symbol's value becomes part of the macro definition instead of the symbol &off. The macro .OF would then result in a hanging indention of that amount, rather than of the value of &off when SCRIPT/VS encounters the macro .OF.

USING SYMBOLS AND MACROS AS ASSOCIATIVE MEMORY

When your document contains a large number of figures, updating the document with a new figure might mean that you have to renumber all subsequent figures. When you have to do this task manually, it is time-consuming and prone to error.

With symbols, SCRIPT/VS can automatically keep track of the numbering you need, and provide more convenient figure referencing as well. You can also build a list of figures, including figure numbers and page numbers, automatically. Most important, you can rearrange the figures as often as you please without having the monumental task of renumbering the figures and their references each time.

To number figures, use a "counter": a unique symbol name that refers to (and contains the value of) the current figure number. The figure number symbol is set at the beginning of the input file, or in a separate file that is imbedded at the beginning of the input file. To manage the counter, define macros for figures and figure references in the profile of your document:

```
    .se figctr = 0
    .*
    .su off
    .dm fignum() /.se figctr = &figctr + 1
    .dm fignum() /.se fig@&*1 = &figctr
    .dm fignum() /.se fig#&*1 = &
    .dm fignum() /Figure &fig@&*1..
    .su on
    .*
    .dm figref /Figure &fig@&*1 on page &fig#&*1
```

Whenever you enter a figure in your document, invoke the FIGNUM macro with a unique "identifier" just before the figure caption:

```
    .fl on
    (body of figure)
    .fignum fred
    Example of Aardvark's Table Manners
    .fl off
```

The FIGNUM macro assigns the figure the unique identifier "fred" and:

- Increments the figure counter
- Saves the number of the figure "fred" in the symbol &fig@fred
- Saves the page number of the figure "fred" in the symbol &fig#fred
- Inserts the word "Figure" and the figure number in front of the figure caption

Whenever you wish to refer to the figure you've called "fred" in the text of your document, use the FIGREF macro:

```
    ... as shown in
    .figref fred
```

The FIGREF macro inserts a string containing the appropriate figure number and page number into your document:

```
     ... as shown in Figure 4 on page 123 ...
```

To automatically build a list of illustrations, the following lines may be added to the FIGNUM macro:

```
    .dm fignum(15) /.se *sx '?Figure &fig@&*1..? .?&fig#&*1..?
    .dm fignum(16) /.dm figlist() |.sx &*sx
```

At the end of the first pass, the FIGLIST macro will contain one line for each figure in the document, and each line will consist of a .SX [Split Text] control word which will format a figure number and page number.

Note: The lines of the FIGNUM macro which build the FIGLIST macro appear before the lines of the FIGNUM macro which set the symbols referenced in the FIGLIST macro. This is deliberate: Figures are usually enclosed in floats or keeps, and the page on which they will be placed is not known when the figure is formatted. For this reason, SCRIPT/VS executes .SE [Set Symbol] control words that reference the page number symbol twice: once when first encountered, and again when the page on which surrounding text will be formatted is known. To ensure that the page numbers in the list of illustrations are correct, substitution of these symbols is delayed until the FIGLIST macro is executed, when all figures have been placed.

REDEFINING SCRIPT/VS FORMATTING CONVENTIONS

A control word, in SCRIPT/VS, is used to request a specific SCRIPT/VS function. You can use a macro to redefine the function of a SCRIPT/VS control word.

SCRIPT/VS has implicit formatting functions, too. Input lines that are null reset line continuation, and those that begin with a blank or tab character cause a break. You can use a macro to redefine these functions.

Processing Input Lines That Are Empty

When SCRIPT/VS encounters a null input line (a line that contains no characters or blanks), it generates and executes a .NL [Null Line] control word, which resets line continuation.

To redefine the SCRIPT/VS implicit formatting convention for null lines, define a .NL [Null Line] macro that will be executed whenever a null line is encountered. For example,

```
.dm nl /.sk 2
```

Now, when SCRIPT/VS encounters a null line, the result is two line spaces on your output page.

You can also define the null line to be completely ignored by SCRIPT/VS:

```
.dm nl /.*
```

Processing Input Lines That Begin With a Blank or a Tab

When an input line begins with a blank (called a leading blank) or a tab (called a leading tab), SCRIPT/VS does not concatenate the line with the previous input line. That is, a break occurs.

Breaks are provided by executing the .LB [Leading Blank] control word when a leading blank is encountered, and by executing the .LT [Leading Tab] control word when a leading tab is encountered. Both of these control words function exactly the same as the .BR [Break] control word. However, after the break occurs, the leading blank or tab remains on the input line and is processed as part of the line.

As with null lines, you can control the actions to be taken for leading blanks and tabs by defining a .LB and .LT macro.

When you want the leading blank and leading tab to be processed by SCRIPT/VS as just a blank (or just a tab) that happens to occur as the first character (that is, not processed differently than other blanks or tabs), redefine the control words with:

```
.dm lb /.*
.dm lt /.*
```

The tab or blank at the beginning of the input line will be concatenated with the previous input line. It will not necessarily appear at the beginning of an output line.

Note: The .NL [Null Line], .LB [Leading Blank], and .LT [Leading Tab] functions are not performed for a line that would otherwise call for them when the line is processed in literal mode (that is, preceded by the .LI [Literal] control word). Null text lines still reset line continuation if the previous line ended with a continuation character, but the .NL control word or macro is not executed.

SPECIFYING A MACRO LIBRARY

When a macro name cannot be resolved (because there was no previous definition set with a .DM [Define Macro] control word), SCRIPT/VS may look for its definition in a macro library.

The member name of each macro defined in the macro library is the macro name without the leading period. It is restricted to eight characters. Symbol definitions and macro definitions may be members of the same library.[43]

Note: No trailing delimiter is required of the lines in a macro definition. However, for compatibility with Release 1, the last nonblank character on each line will be considered a delimiter and deleted if, and only if, it is a colon (:). If you want a colon to be the last character of a macro line, add a second colon as a delimiter.

You can use the macro library in two ways:

- To explicitly set a macro name. Use the .DM [Define Macro] control word to instruct SCRIPT/VS to retrieve its definition from a library:

  ```
  .dm para lib
  ```

  SCRIPT/VS searches the library specified by the LIB option of the SCRIPT command for the definition of .PARA and retrieves the definition. The retrieved definition replaces any existing definition.

- To define an unresolved macro. When SCRIPT/VS encounters a macro that has not been defined, the library is searched for a member with the same name as the macro.

When your input file contains macros that are defined in a macro library, specify either .LY [Library] ON or .LY [Library] MAC to instruct SCRIPT/VS to search the macro library for any unresolved macro it encounters:

```
.ly on     -or-     .ly mac
```

The ON parameter specifies that the macro library is to be searched for unresolved macros and symbols. The MAC parameter specifies that the macro library is to be searched only for unresolved macros. You can use the OFF or SYM parameters of the .LY control word to turn off library searching for unresolved macros.

Since searching macro libraries for unresolved symbols is expensive in terms of processing time, it is recommended that .LY MAC be used except for short periods when you expect symbol definitions to be returned; then .LY SYM or .LY ON should be used.

---

[43] Only the first line of a macro library member is read for a symbol definition; for a macro definition, all lines of the member are read and treated as individual lines of the macro definition.

In ATMS-III, the search technique is the same for both symbols and macros. Therefore, it does not matter whether .LY MAC, .LY SYM, or .LY ON is used.

See "Using the SCRIPT Command" for details about the LIB option and macro libraries.

CREATING SCRIPT/VS MACRO LIBRARIES

Macros that are going to be used for multiple documents can be stored in a macro library. How you create your macro libraries is determined by the environment in which you are operating SCRIPT/VS.

In a CMS Environment

In a CMS environment, a SCRIPT/VS macro library must have a filetype of MACLIB. Members can be edited directly using SPF/CMS but not with the CMS editor or the Display Editing System.

A macro can be created or changed by editing a file with a filename that is the same as the macro name and a filetype of COPY. The record format of the file must be fixed, and the record length must be 80 bytes.

The CMS MACLIB command is used to add or replace macros in a macro library. To modify an existing macro, you must have the text of the macro punched to your virtual card reader, and then read into a COPY file. This makes the macro accessible to the CMS editor. (This procedure is described in detail in Virtual Machine Facility/370: CMS User's Guide.)

You can use the LIB option of the SCRIPT command to specify up to eight macro library names. The filetype for all of these libraries must be MACLIB. If no library name is specified via the LIB option, a default name of GML2 MACLIB is used.

In a TSO Environment

In a TSO environment, your macro library has to be a partitioned data set. TSO does not have standard characteristics for a macro library. Therefore, for SCRIPT/VS, you must set up the data set so that it is in variable-record format. The maximum length of a record is 132 bytes. The block size should be chosen based on the physical device on which your library is going to reside.[44]

The standard data set type for a SCRIPT/VS macro library is MACLIB. This data set type is assumed if one is not specified with the data set name.

The standard name of the SCRIPT/VS macro library is SCRIPT.MACLIB. You can concatenate a private library to this macro library using the LIB option of the SCRIPT command. However, when you do this, you must concatenate the private library to the front of the standard library so that SCRIPT/VS will search it first when looking for a macro definition.

Since only one private macro library can be concatenated using the LIB option, if you want to use multiple private libraries, you must allocate and concatenate them prior to invoking SCRIPT. When doing this, you must use the file name (ddname) of SCRPTLIB. If you want SCRIPT.MACLIB to be searched for macro definitions, you must include it in the concatenation when defining SCRPTLIB. Oth-

[44] It is recommended that a standard block size be used for all SCRIPT/VS macro libraries within an installation. Errors will occur if a macro library is concatenated to another one with a smaller block size.

erwise, it will not be searched. (For more information on concatenating libraries, see OS/VS2 TSO Terminal User's Guide.)

Members of a macro library can be added or changed directly using either the TSO system editor or the Structured Programming Facility-II (SPF-II) editor. The SPF-II utility function can be used to delete members or list member names. Since changing or deleting members leaves free space within a macro library that cannot be reused, you should occasionally reorganize your macro libraries.

In an ATMS-III Environment

In an ATMS-III environment, macros can be created as individual documents. However, they can be accessed as either documents or subdocuments. Macros that are accessed as individual documents must have uppercase names and must reside in permanent storage. Macros that are accessed as subdocuments must also reside in permanent storage but do not have to have uppercase names.

If other operators are going to be using your macros, you must store them with a getword of "any". Another operator can then access them by specifying the LIB option and your operator's number. If you are going to use macros that are stored in your permanent storage, as well as macros that are stored in another operator's permanent storage, you must specify both your number and the other operator's number when specifying the LIB option.[45]

For macros that you are going to access as subdocuments, you might want to give them names with a common prefix. This enables you to build and connect them based on the common prefix. For example, if you created these macros

```
testPARA
testKEEP
prodPARA
prodKEEP
```

issuing these ATMS-III commands

```
build;m;test
connect;x;test
```

results in the macros .PARA and .KEEP being retrieved from the documents testPARA and testKEEP.

In a Batch Environment

In a batch environment, the Document Library Facility must be used to invoked SCRIPT/VS. Therefore, any macro libraries that are required for processing a document must be created as sequential data sets and IMPORTed (brought) into the Document Library Facility's Document Library before they can be accessed by SCRIPT/VS. For more information on how to IMPORT documents into the Document Library using the Document Library Facility, and how to access documents stored in the Document Library, see Document Library Facility Guide.

[45] If all of your macros, symbols, and GML tags are going to be accessed as subdocuments and they are all stored in your permanent storage area, you do not have to specify the LIB option. See the ATMS-III Terminal Operator's Guide for more information on creating and using subdocuments.

Generalized Markup Language (GML) is a language that can be used to describe the structure and elements of your document without regard to the particular processing that may be required. Like other languages, GML has a syntax and usage rules, but GML has no fixed vocabulary. You can develop your own vocabulary of tags to describe your documents. SCRIPT/VS actually provides two languages: the SCRIPT/VS formatting language, and GML descriptive language. One way of characterizing the difference between the two languages is this: the formatting language is made up, basically, of verbs that indicate what processing to perform, while GML is made up, basically, of adjectives that describe the structure and elements of a document.

SCRIPT/VS also provides a GML starter set, consisting of a profile and a macro library to support a set of tags for general documents. You can use the starter set as an example of one way to support GML. Or, you can use the starter set of tags, where appropriate, and add your own tags to tailor the GML vocabulary to describe your documents.

The GML functions of SCRIPT/VS are enabled with the .GS [GML Services] control word:

```
    .gs tag on
```

The profile provided with the GML starter set executes this control word.

GML MARKUP SYNTAX

GML tags can appear anywhere in an input document, and are identified by the GML delimiter, which, by default, is a colon (:). A control word should never precede a tag in the same input line. If doing so is absolutely necessary, then use the control word separator symbol (&$CW) instead of the control word separator character. A GML tag name can be up to 8 characters long, and can consist of letters, numbers, and the characters @, #, and $ (except that the first character cannot be numeric). The tag name may be entered in either upper- or lowercase. For example, in the GML starter set provided with SCRIPT/VS, the following tag identifies a place where a list of illustrations should be generated:

```
    :figlist
```

This same tag can also be entered as:

```
    :FIGLIST
```

GML tags indicate where specific document elements begin. Some elements also require an explicit "end-tag" to indicate the end of the element. GML end-tags are identified by the GML end-delimiter, by default a double colon (::), and have the same naming rules as GML tags. For example, an ordered list might be indicated as:

```
    :ol
      .
      .
      .
    ::ol
```

Some GML tags recognize "attributes," which further describe the document element identified by the tag. Attributes follow the tag name, separated by one or more blanks, and have the same naming rules as GML tags. Attributes also have values, which follow the attribute name, separated by an equals sign:

```
    :fig frame=box
```

When an attribute value contains blanks or special characters, it must be enclosed in single quotation marks ('):

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

If the value itself contains quotation marks, they should be doubled.

Some tags recognize attributes that consist of a single word. These are called "value attributes," and have the same naming rules as GML tags. They are entered just as other attributes, but without any equals sign:

```
:ol compact
```

Whenever text follows markup, the text should be delimited by a "markup/content separator" (MCS), which is by default a period (.). For example,

```
:p.While there's no cause for alarm,
there's no room for complacency.
```

The line of text following the markup/content separator (or the last attribute, if no MCS character is entered) is the "residual text" for the GML tag. In the example above, the residual text for the "p" tag is "While there's no cause for alarm,".

The residual text may be null, if no text appears between the end of markup and the next tag. For example,

```
:ol
:li.
A solitary list item.
::ol
```

The residual text for the "ol" tag is null, while the residual text for the "li" tag is "A solitary list item.".

The markup/content separator need not be entered if

- The markup is immediately followed by another tag
- Whatever follows the markup cannot be misconstrued as an attribute

The markup/content separator can appear anywhere on an input line; however, if the MCS character is the period (.), SCRIPT/VS will interpret it as a control word delimiter if it appears in the first character position of an input line. You should, therefore, avoid starting a line with a markup/content separator.

GML markup can span as many lines in the input document as necessary, and blanks between attributes are ignored. For example, a tag, its attributes, and the residual text may all be entered on a single line:

```
:h1 id=gml.GML Support in SCRIPT/VS
```

Or, a tag, each of its attributes, and the residual text may all be entered on separate lines:

```
:h1
  id = gml
     stitle=  'GML Support'
Generalized Markup Language Support in SCRIPT/VS
```

Each input line can have one or more attributes on it, separated by one or more blanks, but each attribute must be entirely contained on a single line. The markup can end on any line, with the residual text line following all on that same line, or all on the next line. When the scanning for GML markup is complete, the APF is then executed.

Residual text is treated as literal text; that is, special processing, such as execution of another control word, is not performed if the line begins with a leading blank, tab, or control word separator. Normally, residual text is formatted along with any text following the markup; however, in format off mode, a tag in the middle of an input line may cause two or more output lines if that tag contains control words that cause a break.

The GML scanning mechanism was designed to enable the creation of APFs and tags that can be used to describe the structure and elements of your documents. It was not designed as a means of introducing text, such as "boilerplate phrases." If such text is deemed necessary, you should avoid, if possible, having the text at the end of the APF. If this is not possible, then use a continuation character at the end of the phrase. For example,

```
.gs tag on
.dm text /phrase
:text.,
xxx :text.,
```

will result in

```
phrase , xxx phrase, (notice the blank before the first comma)
```

whereas

```
.dc cont+
.gs tag on
.dm text /phrase+
:text.,
xxx :text,
```

will result in

```
phrase, xxx phrase,
```

GML scanning may be ended by another tag, by a control word at the start of an input line, or if an end-of-input file condition is encountered.

Changing the GML delimiters

The GML tag and end-tag delimiters, and the markup/content separator can be changed with the .DC [Define Character] control word.

The GML tag delimiter can be set to any character that is not valid in a tag name, except ampersand (&). For example,

```
.dc gml !
```

With this delimiter, the list of illustrations would be identified as:

```
!figlist
```

The GML end-tag delimiter may be one or two characters. If it is a single character, it may be any character that is not valid in a tag name, except ampersand (&) and the GML tag delimiter. For example,

```
.dc gml $ ¢
```

With these delimiters, an ordered list would be identified as

```
$ol
 .
 .
 .
¢ol
```

If the GML end-tag delimiter is two characters, the first must be the same as the GML tag delimiter. For example, in the GML starter set the delimiters are set as:

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

With these delimiters, an ordered list would be identified as

```
:ol
 .
 .
 .
:eol
```

The markup/content separator may be set to any character that is not valid in a tag name, except ampersand (&). For example,

```
.dc gml <
.dc mcs >
```

With these delimiters, tags may be entered as:

```
<h1 id=gml>GML Support in SCRIPT/VS
```

SCRIPT/VS PROCESSING OF GML

When SCRIPT/VS processes a document, an "application processing function" (APF) is invoked for each tag and attribute to perform whatever SCRIPT/VS processing is required. The APFs are written in the SCRIPT/VS formatting language, and are usually macros. This section describes the functions available in SCRIPT/VS to recognize GML markup, and associate the tags and attributes with APFs.

GML Tag-to-APF Mapping

GML scanning is enabled with the .GS [GML Services] control word:

```
.gs tag on
```

When a valid GML tag is found, SCRIPT/VS attempts to locate an APF for the tag. The APF, which may be a macro or a control word, may be found by

- Explicit mapping (established with the .AA [Associate APF] control word)
- Class mapping (established with the .GS [GML Services] PREFIX control word)
- Direct APF mapping (a macro or control word with the same name as the tag)

If no APF is found, a warning message is issued, and the tag is treated as text. If you do not want to be warned about invalid tags, specify

```
.gs tag onno
```

Explicit Mapping

The .AA [Associate APF] control word allows you to explicitly specify the APFs for particular GML tags and end-tags. For example, to define tag-to-APF mappings for the "fig" tag and end-tag, specify

```
.aa fig figure figurex
```

The APF for the "fig" tag will be the FIGURE macro, and the APF for the "fig" end-tag will be the FIGUREX macro.

The .AA control word also allows you to specify the attribute scanning rules for each tag, as described under "Attribute Scanning Rules" on page 163.

The .AA control word is described in more detail in ".AA [Associate APF]" on page 223.

Class Mapping

A single character may be specified with the .GS [GML Services] PREFIX control word which will be added to the front of a tag name to produce an APF name. For example,

```
    .gs prefix @
```

With this class mapping in effect, the APF for the "figlist" tag will be the @FIGLIST macro.

Direct Mapping

If no other tag-to-APF mapping is provided for a tag, a macro or control word whose name matches the tag name is used as the APF. This is the default.

Attribute Scanning Rules

The .GS [GML Services] RULES control word may be used to specify

- Whether attributes are allowed for tags
- Whether value attributes are allowed
- What to do if an invalid attribute is found:
  - Stop the scan and treat the invalid attribute as text, or
  - Step over the invalid attribute and keep scanning
- Whether to issue a message if an invalid attribute is found, or quietly take the appropriate action

Attribute scanning rules may be specified separately for GML tags and end-tags. For example, in the GML starter set provided with SCRIPT/VS:

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

specifies that GML tags can have attributes but not value attributes, and that attribute scanning should stop without a warning message when an invalid attribute is found;[46] GML end-tags will not recognize attributes at all.

The attribute scanning rules for tags given with .GS RULES may be overridden for specific tags with the .AA [Associate APF] control word. For example, in the GML starter set,

```
    .aa ol olist (vat) elist
```

indicates that the APF for the "ol" tag will be the OLIST macro, and that value attributes will be allowed for this tag. The APF for the "ol" end-tag will be the ELIST tag, and since no attribute scanning rules are specified, they are those given with .GS RULES for end-tags (that is, no attributes will be recognized).

The attribute scanning rules for .AA and .GS RULES are described under ".GS [GML Services]" on page 265.

---

[46] Invalid attributes are most commonly text, encountered when an optional markup/content separator has been omitted.

| **Note:** GML markup can not span input files.

Automatic GML Processing

When a GML tag is encountered, SCRIPT/VS automatically

- Purges any attributes not processed by the previous tag,
- Finds the APF for the tag, via either an explicit mapping (.AA), a class mapping (.GS PREFIX), or a direct mapping
- Scans the input for attributes and value attributes, if recognized by the tag, and saves them for processing with .GS EXATT,
- Identifies and saves the residual text line

Attribute Processing

Within the APF for the tag, the .GS [GML Services] EXATT control word can be used to selectively invoke the APFs for attributes. For example, if a "fig" tag is encountered:

```
|    :fig id=fred place=inline frame=box
```

the APF which processes the "fig" tag can specify

```
    .gs exatt frame id as @idf
```

The "frame" attribute will be processed by the FRAME macro; the value of the attribute, "box", will be provided to the FRAME macro as the parameter &*1. The "id" attribute will be processed by the @IDF macro; the value "fred" will be provided as &*1.

The APF for the "fig" tag may also specify

```
    .gs exatt width
```

but since the "width" attribute was not specified with the tag, no macro will be executed.

Attribute execution is described in further detail under ".GS [GML Services]" on page 265.

Value Attribute Processing

Value attributes are presented to the APF for the tag as the parameters &*1, &*2, ... The number of value attributes is provided in &*0.

Residual Text Processing

For many elements, the APF operates by setting up the correct formatting environment, and then allowing the following text to be formatted under the control of this environment. In these cases, the APF does not need to process the residual text line directly; SCRIPT/VS automatically retrieves the residual line and processes it after the APF has completed its function. SCRIPT/VS automatically provides continuation, if necessary, so that if the GML markup ocurred in the middle of a word, the processing (such as starting a new font, for example) will not break the word.

If the APF needs to process the residual line directly, the APF can retrieve the residual line with the .GS [GML Services] SCAN control word:

```
    .gs scan line
```

The residual text, which may be null, is placed in the symbol &line. When an APF explicitly retrieves the residual text, it is the APF's responsibility to provide continuation or other special treatment that may be required, such as turning on literal mode for the residual piece.

USER-DEFINED COMMAND OPTIONS

Another function available with SCRIPT/VS is "User-defined command option" processing. These are command options that begin with the commercial AT sign (@). (See "@user-option: User-defined Options" on page 32.) All such options, along with their associated suboptions, are saved when SCRIPT/VS is invoked. You can write APFs to process the user-options, and then execute them with the .GS [GML Services] EXOPT control word. The EXOPT function is analogous to the EXATT function.

GML TAGS AS SYMBOLS

In Release 1 of SCRIPT/VS, GML tags were processed as specialized symbols, rather than as tags. GML recognition and attribute scanning were performed by symbols and macros provided with the starter set. This was accomplished by interpreting the GML delimiter, by default a colon (:), as an alternate symbol delimiter.

This mechanism can be used in Release 2 by setting:

```
.gs tag symbol
```

This is the default. If you do not specifically enable the GML scanning as described earlier in this chapter, tags will be interpreted as symbols.

When symbol-type GML scanning is in use, the GML delimiter acts like the symbol delimiter (&), with two important differences:

- The symbol name is folded to uppercase
- Compound symbol substitution is not performed

For example, suppose a symbol is defined as

```
.se NOTE = '.@note '
```

The input line

```
:note.By the way, ...
```

contains a symbol ":note." and the value of the symbol &NOTE is substituted:

```
.@note By the way, ...
```

When the input line is processed, the @NOTE macro is invoked.

CHAPTER 15. USING SCRIPT/VS WITH OTHER PROGRAMS

You can use SCRIPT/VS to format an input stream prepared by another program. You can also use SCRIPT/VS as a preprocessor, to prepare an input file for processing by another text processing system or by an application program.

USING SCRIPT/VS AS A POSTPROCESSOR

You can use SCRIPT/VS to format reports using data from data processing files. An application program could access these files, perform the necessary computations, and create an output file. The output file could contain GML markup just as if it had been created with normal text entry procedures. You will then be able to process it with the same flexibility as any of your other documents.

Alternatively, the application program can call SCRIPT/VS as a subroutine. This can be done when the Document Library Facility is installed with SCRIPT/VS. For details on using SCRIPT/VS via the Document Library Facility, see the Document Library Facility Guide.

You can also use SCRIPT/VS to prepare input for itself, except in an ATMS-III environment. For example, you can use the .WF [Write To File] control word to create input files dynamically. These files can later be resubmitted to SCRIPT/VS for further processing.

You can also write formatted output to a file and use it as input for a subsequent invocation of SCRIPT/VS.

USING SCRIPT/VS AS A PREPROCESSOR

When you use SCRIPT/VS as a preprocessor, you want SCRIPT/VS to produce an output file that can be processed by some other text formatter or application program. To use SCRIPT/VS as a preprocessor, you must first thoroughly understand the text formatter that is to receive the output file prepared by SCRIPT/VS.

Your SCRIPT/VS input file can contain any markup appropriate for SCRIPT/VS (that is, GML tags, control words, macros, and symbols) as well as text and implicit formatting conventions (such as leading blanks, leading tabs, null lines, and full stops). You must build a profile and APFs that interpret the SCRIPT/VS markup and generate appropriate formatter controls.[47]

In most cases, you will find it preferable to use GML markup when using SCRIPT/VS as a preprocessor. The following discussion, therefore, will assume that your document's markup observes conventions like those described in the Document Composition Facility Generalized Markup Language: Starter Set Reference.

Developing Preprocessor APFs and Profiles

SCRIPT/VS has a great variety of general document-handling functions which can be used independently of formatting. You can use these functions to create APFs that will translate a SCRIPT/VS document into suitable input for another program, such as a formatter that can support photocomposers.

[47] See "Chapter 14. GML Support in SCRIPT/VS" on page 159 for details about profiles, APFs, and mapping tags to APFs. See "Chapter 12. Symbols in Your Document" on page 129 for details about symbols, and "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147 for details about macros.

For example, the GML starter set APFs for ordered lists and list items automatically generate numbers (or letters) for the items on an ordered list. This is convenient, since it permits the list to be revised without renumbering all the items.

You can create a modified version of the APFs which retain the general processing functions, but eliminate the SCRIPT/VS control words that result in formatting. For example, instead of executing the .SK [Skip] and .IN [Indent] control words, you would insert the appropriate formatting controls of the postprocessor into the output stream. The SCRIPT/VS symbol substitution capability can still be used to calculate parameters for the postprocessor's formatting controls.

Some of the logical sequence of formatting controls might have to be changed, however. The graphic effect of having the first line of a list item printed to the left of the indention for the rest of the list item is achieved, in SCRIPT/VS, with the .IN [Indent] control word. The receiving processor might require a different sequence of formatting controls to achieve the same graphic effect.

When modifying an APF in this way, you can structure its logic and function to produce formatting different from that produced by the original APF. You can change the symbol definition for symbols used to achieve different formatting values.

In addition to creating APFs, you would also create a profile which would map to the new APFs. The profile would also issue control words that would turn off justification and page numbering, and the like, so the output would look like a source file.

```
.pm 0
.tm 0
.bm 0
.wz off
.fo off
```

You might also need to translate special characters which might be unacceptable to the postprocessor.

By having two sets of APFs and two profiles, you could continue to print draft copies of the document on a line printer while getting final output on a photocomposer via the postprocessor.

Redefining Symbols

Many symbols used in source document markup will not require redefinition. For example, those used:

- As abbreviations for lengthy character strings
- As references to generated information which is not format-dependent (such as a figure or section number -- but not a page number)
- To enter unkeyable characters which are represented by the same codes in both SCRIPT/VS and the postprocessor

Handling Directly Entered Control Words

Observing a GML convention for direct entry of control words, like that described in the Document Composition Facility Generalized Markup Language: Starter Set Reference, makes it easy to prepare your document for another processor. The following discussion refers to the specific conventions recommended in that book, but the information is applicable to conventions that may be adopted by your own installation.

Managing a Source Document

The .CM [Comment], .IM [Imbed], and .SE [Set Symbol] control words are executed by SCRIPT/VS before the document is available to the postprocessor. You need take no special action with respect to them.

However, the .RC [Revision Code] and .OC [Output Comment] control words are different; they have a formatting effect. (The .RC control word inserts a revision code character to the left of an output line; the .OC control word places unformatted output comments at the same position in the output as they were encountered in the input.) If the postprocessor has comparable functions, you can define _macros_, called .RC and .OC, to generate the corresponding postprocessor controls. (This technique can be used for all control words if a one-to-one conversion approach is taken.)

If the "revision code" and/or "output comment" functions are not available, you can deactivate them by specifying

```
.dm rc /.cm
```

and/or

```
.dm oc /.cm
```

which defines the .RC and/or .OC macros to be comments.

Preparing for Processing

When you are ready to have SCRIPT/VS prepare your input file for the receiving text processor, take the usual steps needed for SCRIPT/VS execution, as discussed in "Chapter 2. Using the SCRIPT Command" on page 13.

While the file produced by SCRIPT/VS will contain the correct text and markup for your postprocessor, it will not necessarily have the correct physical characteristics. Some postprocessors may require record lengths and formats, or other characteristics, that differ from those produced by SCRIPT/VS. You might have to use a utility program, or code your own, to handle such interface requirements.

CHAPTER 16. AUTOMATIC HYPHENATION AND SPELLING VERIFICATION

SCRIPT/VS can automatically hyphenate and verify the spelling of words. When these functions are activated, words that occur at the end of an output line will automatically be hyphenated if needed, and each word in your document will be checked for correct spelling. The SCRIPT/VS dictionaries, described later in this chapter, are used for both hyphenating words and for spelling verification.

HYPHENATION

During formatting, if hyphenation is enabled and the next word does not fit in the current line, SCRIPT/VS tries to hyphenate the word using the method you have specified with the .HY [Hyphenate] control. You can instruct SCRIPT/VS to:

- Search a SCRIPT/VS dictionary to see if there is an entry for the word to be hyphenated

  and/or

- Use an algorithmic hyphenator to hyphenate the word.

Unless otherwise specified, SCRIPT/VS will first search for the word in the main and addenda dictionaries that make up the SCRIPT/VS dictionary being used. If the word cannot be found there, it will use an algorithmic hyphenator to perform the hyphenation. If you do not want the addenda dictionary searched, you specify

```
.hy noadd
```

If you do not want any of the dictionaries to be searched, you specify

```
.hy nodict
```

If you do not want the available algorithmic hyphenator to be used, you specify

```
.hy noalg
```

You can also use the .HW [Hyphenate Word] control word to provide SCRIPT/VS with the hyphenation points for a specific word. Hyphenation specified using the .HW [Hyphenate Word] control word takes precedence over any other hyphenation method that is in effect, but affects only that specific instance of the word.

Searching a SCRIPT/VS Dictionary

When using a SCRIPT/VS dictionary for hyphenation, any time SCRIPT/VS encounters a word that needs to be hyphenated, it searches the SCRIPT/VS dictionary being used for the word as it appears in the input line. The associated addenda dictionaries are searched first and then, if the word is not found there, the SCRIPT/VS main dictionary with which the addenda dictionaries are associated is searched.

If no match is found in any of these dictionaries, and the word, as it appears, is all in uppercase characters, all of the letters except the first are translated to lowercase and SCRIPT/VS again searches for the word in the addenda and main dictionaries. If no match is found, SCRIPT/VS translates all of the letters to lowercase and repeats the search. (If the word is not all uppercase and any letter but the first is capitalized, the word will not be hyphenated.)

If no match is found this time, SCRIPT/VS removes the prefix and the suffix if any, to yield the word's "root." This form of the word is then searched for in the dictionaries. If no match is found, the word will be hyphenated using an algorithmic hyphenator unless .HY NOALG was specified.

Using an Algorithmic Hyphenator

Unless you use the NOALG parameter of the .HY [Hyphenate] control word to instruct SCRIPT/VS to do otherwise, it will attempt to use an algorithmic hyphenation routine to hyphenate:

- Words that fail to verify using the supplied language dictionaries
- All words if the NODICT parameter of the .HY control word was specified

An algorithmic hyphenation routine for English is provided with SCRIPT/VS. Your installation may provide other algorithmic hyphenators for English or any of the other languages. Any installation provided algorithmic hyphenators must be linkedited to SCRIPT/VS before they can be used during hyphenation processing. For information on how to linkedit such a routine, see the Document Composition Facility Program Directory.

Hyphenating Single Words

Regardless of whether SCRIPT/VS is using automatic hyphenation or not, there may be occasions when you would like a word to be hyphenated if it occurs at the end of a line. The .HW [Hyphenate Word] control word allows you to specify how a word should be hyphenated if hyphenation is necessary.

This may be convenient for long words that are normally hyphenated, or for words that occasionally need hyphenation. For example,

```
Guinevere's
.hw lighter--than--air
laughter was heard
.hw through-out
the kingdom.
```

When this line is processed, SCRIPT/VS uses the hyphens supplied as hyphenation points and suppresses the hyphens it does not need:

```
Guinevere's   lighter-than-air
laughter  was heard throughout
the kingdom.
```

Note that since "throughout" did not require hyphenation when the line was formatted, the hyphen was suppressed. For the hyphenated expression "lighter-than-air," two hyphens are used with the .HW [Hyphenate Word] control word so SCRIPT/VS prints the necessary hyphens. Note that the hyphenation rules for a .HW word apply only in this instance, and nowhere else in the input file where they appear.

Changing the Frequency of Hyphenation

You can increase or decrease the frequency of hyphenation in a document using the MINPT parameter of the .HY [Hyphenate] control word. MINPT controls the minimum hyphenation point (the smallest number of characters acceptable as a hyphenation point for the word).

The default value for MINPT is 4. To change it, you could specify

```
.hy set minpt 3
.hy on
```

SPELLING VERIFICATION

The spelling of words in your input file will be checked by the SCRIPT/VS spelling verification function when you include the SPELLCHK option in the SCRIPT command.

Spelling verification is accomplished by attempting to find each word in the input line in the SCRIPT/VS dictionary (described later in this chapter).

For purposes of spelling verification, a "word" is a string of 2 to 55 characters delimited by "word delimiters." The default word delimiters are listed in Figure 37 on page 363. You may change the word delimiters for spelling verification with the .DC [Define Character] WORD control word.

Punctuation characters are considered part of the word if they appear within it, but are removed before spelling verification is performed if they appear at the end of the word. The default punctuation characters are the hyphen (-) and apostrophe ('). You can change the punctuation characters with the .DC [Define Character] PUNC control word.

When words are verified for correct spelling, the original word, using the case (upper, lower, or mixed) as it occurs in the input line after symbol substitution, is checked against both the main and addenda dictionaries that make up the SCRIPT/VS dictionary being used. If no match is found and the word is in uppercase, all of the letters except the first are translated to lowercase and the word is again checked against both dictionaries. If still no match is found, the first letter is translated to lowercase and the word is again checked against both dictionaries. (If the word is not all uppercase and any character other than the first is capitalized, the word is considered unverified.) If no match is found this time, SCRIPT/VS removes the prefix and suffix, if any, to yield the word's "root." This form of the word is then checked against both dictionaries. If again no match is found, the word is considered unverified. SCRIPT/VS issues a message listing all of the unverified (and potentially misspelled) words in an input line by invoking the .UW [Unverified Word] control word.

Note: Since stem processing is performed only after each word is translated to lowercase, all words placed in the addenda dictionary should be in lowercase if stem processing is desired. No match will be found for a lowercase occurrence of a word if that word was added to the addenda dictionary in uppercase.

Spelling verification is normally performed using the main and addenda dictionaries with stem processing. Words that contain numbers are not checked unless requested with the NUM parameter of .SV [Spelling Verification] control word.

You can specify that:

- The addenda dictionary is not to be used:

  ```
  .sv noadd
  ```

- Full word processing rather than stem processing is to be performed:

  ```
  .sv nostem
  ```

- Words that contain numbers are to be checked:

  ```
  .sv num
  ```

Spelling verification can also be used to verify that proper names start with an initial capital letter. For example, if an entry is made in the addenda dictionary as follows,

```
.du add Teri
```

then "Teri" and "TERI" will both be correctly spelled. However, "teri" will be regarded as misspelled.

Fallibility

SCRIPT/VS spelling verification is not infallible. A misspelled word with a suffix or prefix could possibly yield a correctly spelled word after stem processing. For example, "disbooked" (with the stem "book"), and "missteak" (with the stem "steak") are both "correctly" spelled.

Also, the stem processing algorithms do not handle all exceptions to general spelling rules used in the English Language. For example, the plural of "mouse" must be explicitly added to an addenda dictionary.

THE SCRIPT/VS DICTIONARIES

There are three types of SCRIPT/VS dictionaries that are used for hyphenation and spelling verification:

- Read-only dictionaries of root words provided by IBM with SCRIPT/VS. Each contains about 10,000 words. Since suffixes and prefixes are removed before a word is searched for in this dictionary, the effective dictionary size is significantly larger.
- User dictionaries created by your installation using the Dictionary Maintenance program. These dictionaries contain words that are not in the main dictionaries but are used in most of the documents produced at your installation. These words often reflect the nature of a particular business and usually include technical terms and company acronyms. Once created, these dictionaries are also read-only.
- Addenda dictionaries you create for a specific document or group of documents using the .DU [Dictionary Update] ADD control word. Addenda dictionaries contain words that are not in the main or user-created dictionaries but are frequently used in a specific document or a group of documents. This type of dictionary often includes acronyms that apply to a particular product, jargon, and the names of people and places. It is the most temporary of the three types of dictionaries since it is rebuilt in storage every time SCRIPT/VS processes a document that requires it. Addenda dictionaries can be updated as required.

IBM provides root word dictionaries in nine languages:

- American English
- United Kingdom English
- Canadian English
- Canadian French
- French
- German
- Italian
- Dutch
- Spanish

The unique stem processing routine that IBM provides with each of these languages is used by all three types of SCRIPT/VS dictionaries in performing hyphenation and spelling verification in a given language.

Use the .DL [Dictionary List] control word to specify which language you want to use for hyphenation and spelling verification. This control word automatically activates the corresponding stem processing routine for that language, as well as any user dictionaries that are associated with that root word dictionary.

The hexadecimal codepoints for accented characters in the SCRIPT/VS spelling checking and hyphenation dictionaries are listed in Figure 12.

DUTCH	CANADIAN FRENCH	FRENCH	GERMAN	ITALIAN	SPANISH	Character Name
UC/lc	UC/lc	UC/lc	UC/lc	UC/lc	UC/lc	
/45					/45	"A" Acute
/44	64/44	/44		/44		"A" Grave
/42	62/42	/42				"A" Circumflex
63/43			63/43			"A" Diaeresis (Umlaut)
/48	68/48	/48				"C" Cedilla
/51	71/51	/51		/51	/51	"E" Acute
/54	74/54	/54		/54		"E" Grave
/52	72/52	/52				"E" Circumflex
73/53	73/53	/53				"E" Diaeresis (Umlaut)
					/55	"I" Acute
				/58		"I" Grave
	76/56	/56				"I" Circumflex
77/57	77/57	/57				"I" Diaeresis (Umlaut)
					69/49	"N" Tilde
/CE					/CE	"O" Acute
/CD				/CD		"O" Grave
/CB	EB/CB	/CB				"O" Circumflex
EC/CC			EC/CC			"O" Diaeresis (Umlaut)
	EE/DF	EE/DF				"OE" Digraph
					/DE	"U" Acute
	FD/DD	/DD		/DD		"U" Grave
	FB/DB	/DB				"U" Circumflex
FC/DC	FC/DC	/DC	FC/DC		/DC	"U" Diaeresis (Umlaut)
			59			Ess zet

Figure 12. Codepoint Assignments for Accented Characters: Accented characters in the SCRIPT/VS Spelling Checking and Hyphenation dictionaries are represented using the hexadecimal codepoints shown under each language for uppercase (**UC**) and lowercase (**lc**) characters.

Building a User Dictionary

A user dictionary is created and updated using the dictionary maintenance procedures that are described in "Appendix F. Maintaining User Dictionaries" on page 391. The words that are to be placed into the user dictionary are submitted, with the appropriate JCL, to run as a batch job in a background environment.

The input record for each job is 80 bytes in length and includes

- The appropriate hyphenation for the word
- An indication as to whether or not the word can be automatically deleted from the dictionary if it is not referred to in a specified period of time

- The date on which the word was placed in the dictionary

Once the user dictionary has been built, it must be concatenated to the main dictionary to be accessible to SCRIPT/VS.

Since it is concatenated to the main dictionary, SCRIPT/VS treats it as part of the main dictionary. Therefore, whenever you specify that the main dictionary is to be used for hyphenation and spelling verification, you are automatically specifying that the user dictionary is also to be used.

Whenever you update the user dictionary, it must be reconcatenated to the main dictionary for the updates to go into effect.

Building an Addenda Dictionary

You use the .DU [Dictionary Update] control word to create an addenda dictionary. Each word specified with this control word is delimited with blanks. The word can contain lowercase and uppercase alphabetic characters, the integers 0 through 9, and punctuation characters, as defined with the .DC [Define Character] PUNC control word.

If you are building an addenda dictionary for use with multiple documents, you can create a separate file to contain the .DU [Dictionary Update] control words being used to build it and then imbed this file at the beginning of any input file that requires it.

When you include single hyphens in a word that you are adding to an addenda dictionary, SCRIPT/VS assumes they are potential hyphenation points. Therefore, words that normally contain hyphens (for example, upside-down) should be specified with a double-hyphen for the normally appearing hyphen. For example,

```
.du add up-side--down
```

specifies two potential hyphenation points: between "up" and "side," and between "side" and "down." It also specifies one normal hyphen that is to always appear: between "side" and "down."

Before creating an addenda dictionary, you should use the .DL [Dictionary List] control word to specify the language you are using. This will associate the addenda dictionary with the main dictionary for that language. For example, specifying

```
.dl eam
.du add Paul Ri-ver-front ec-cle-si-asti-cal
```

causes SCRIPT/VS to use the American English root word dictionary, and associate the addenda dictionary with that main dictionary. The new addenda dictionary will contain the words "Paul," "Riverfront," and "ecclesiastical," which are not in the main American English dictionary. Since the entries to this addenda dictionary show the hyphenation points for these words, the addenda dictionary can be used for both hyphenation and spelling verification.

The .DU [Dictionary Update] control word can later be used to add more words to the addenda dictionary, or to delete words previously added. For example, specifying

```
.du add Com-pos-i-tion photo-com-po-ser
.du del Paul Ri-ver-front
```

adds the words "Composition" and "photocomposer" to the addenda dictionary, and removes the words "Paul" and "Riverfront" from it.

If you specify a new language prior to specifying the .DU ADD and .DU DEL control words, the new words will be placed in the addenda dictionary associated with the new language. For example, specifying

```
.dl germ
.du add Aus-wahl-list-en Ent-wick-lung
```

causes the German main dictionary to be used instead of the American English one, and the words "Auswahllisten" and "Entwicklung" to be added to the addenda dictionary associated with this main dictionary.

Note: Multiple languages can be used in processing a document. However, only one language can be active at a time.

Stem Processing

The stem processing function attempts to generate one or more possible root words from which the input word might be derived. Suffix and prefix processing are both performed on the input word. The stem processing function does not generate a root word, or stem, less than three characters long.

When a word's prefix is removed, the resulting stem is not changed. However, when a word's suffix is removed, the stem processing function derives the word's stem based on the spelling rules for the language being used. For example, in English the word "churches" yields the stem "church", and the word "flames" yields the stem "flame."

Words may have to be processed repeatedly to remove multiple suffixes before yielding a stem. For example, the word "conceptions" would lose the two suffixes "s" and "ion" before yielding the stem "concept."

For a summary of the prefixes and suffixes SCRIPT/VS checks for during stem processing, see "Appendix C. Stem Processing" on page 373.

Hyphenation Points

The SCRIPT/VS dictionaries do not contain all possible hyphenation points for all words. Each word placed in an addenda, user, or root word dictionary is divided into four three-character groups following the first vowel. Only one hyphenation point is recorded for each of the four groups.

CHAPTER 17. PRODUCING INPUT FOR STAIRS/VS

The Storage and Information Retrieval System/Virtual Storage (STAIRS/VS) is an IBM program product that provides content-based retrieval of documents using a comprehensive indexing structure. Documents to be stored in the STAIRS/VS data base must be prepared in a Condensed Text Format (CTF). SCRIPT/VS can provide input for STAIRS/VS in this format, as shown in Figure 13 on page 180.

SPECIFYING STAIRS/VS OUTPUT

Use the DEVICE option of the SCRIPT command to specify STAIRS/VS output. (For details, refer to "DEVICE: Specify a Logical Output Device" on page 21.)

When you specify DEVICE(STAIRS) or CTF, SCRIPT/VS formats the input document as it would for device 1403W6. The formatted lines are then converted to STAIRS/VS CTF blocks.

If you specify DEVICE(STAIRS) and PRINT, FILE, or TERM, the document is formatted as it would be for CTF output, but instead is written to the specified destination for proofreading.

RESTRICTIONS IMPOSED ON FORMATTED OUTPUT

The STAIRS/VS program indexes only the text in the body of a document. Consequently, running titles, headings and footings, footnotes, and floats are ignored when preparing STAIRS/VS output. Multiple-column sections are treated as one single column. Underscoring, overstriking, skip, and space are also ignored.

In addition, STAIRS/VS requires that its input not exceed 69 characters per line, or 449 lines per paragraph. In documents prepared for STAIRS/VS, lines exceeding these limitations will be flagged when output is being prepared in "proof" format, and will be truncated when output is being prepared in CTF records. The following error flags are placed in columns 72 through 80 of proof output:

C	- more than 69 characters
L	- more than 449 lines
P	- paragraph id "stepdown"

The STAIRS/VS indexing scheme includes the sentence number within a paragraph. STAIRS/VS considers sentences to be delimited by "full stops": a full stop character followed by two blanks. As described in "Chapter 3. Basic Text Processing" on page 33, SCRIPT/VS will automatically insert an extra blank between sentences when an input line ends in a full stop character, to satisfy this requirement.

STAIRS/VS PARAGRAPH NUMBERING

STAIRS/VS requires that a 3-character "number" be associated with each paragraph of a document placed in its data base.

STAIRS/VS paragraph numbers are composed of a single decimal digit (0 through 9), followed by one or two alphameric characters in ascending order (blank, A through Z, and 0 through 9). SCRIPT/VS, by default, numbers the first paragraph of a document "0", and increments the paragraph number by one with each break when concatenation is on, or by one with each input line when concatenation is off.

Thus, the first and subsequent paragraphs will be numbered:

Offset Dec.	Offset Hex.	Length	Contents
0	0	12	SCRIPT/VS Document Name
12	C	3	Paragraph number of first line
15	F	1	Continued Block Count
16	10	2	Operator Number
18	12	2	CTF Record Length
20	14	5	Read Password
25	19	5	Delete Password
30	1E	45	(Reserved)
75	4B	69	First line of text
144	90	864	Twelve more lines:
144	90	3	Paragraph number of second line
147	93	69	Second line of text
216	D8	3	Paragraph number of third line
219	DB	69	Third line of text
...	...	...	...
936	3A8	3	Paragraph number of thirteenth line
939	3AB	69	Thirteenth line of text

Figure 13. STAIRS/VS Condensed Text Format (CTF) Records: Each record has a fixed length of 1008 bytes, and contains up to 13 lines of text.

```
0       0AE     0A2     0BB
0A      0AF     0A3     0BC
0AA     ...     ...     ...
0AB     0AZ     0A9
0AC     0A0     0B
0AD     0A1     0BA
```

You can reset the STAIRS/VS paragraph numbering counter at any time with the .SO [STAIRS/VS Output] control word. For example, specifying

```
.so pid 20b
```

will cause subsequent paragraphs to be numbered

```
20B     20Z     208
20C     200     209
20D     201     21
...     ...     21A
20X     206     21B
20Y     207     ...
```

When the STAIRS/VS paragraph numbering counter is reset to a value which is equal to or less than the last value used, a new logical document is created, regardless of whether a new document name has been specified with the .SO DOC control word.

When output is prepared in Condensed Text Format, the STAIRS/VS paragraph number is included as part of the CTF record with each line. When output is prepared in "proof" format, the STAIRS/VS paragraph number is printed to the left of the first line of each paragraph.

Information for the Document Name, Operator Number, and Read and Delete Password fields of the CTF block may also be provided with the .SO [STAIRS/VS Output] control word.

CHAPTER 18. DIAGNOSTIC AIDS

The diagnostic aids presented in this chapter are tools to help you find and correct problems caused by incorrectly specified or missing control words. Diagnostic aids in this chapter might also be useful to the Programming Service Representative (PSR). However, this chapter is directed toward the person who needs to find out why the sequence of specified GML tags and control words is not resulting in the desired output.

To use SCRIPT/VS effectively, you need to know the function of each GML tag and control word you use. Many formatting problems can result from tags and control words used incorrectly. Other chapters in this book provide detailed descriptions of each SCRIPT/VS control word. The IBM-supplied APFs that define the GML tags are documented with comments in the APFs themselves.

SCRIPT/VS diagnostic aids are provided as options of the SCRIPT command and as SCRIPT/VS control words.

DEBUGGING WITH THE SCRIPT COMMAND

Some of the SCRIPT command options are useful (for diagnostic purposes) when specified to format an input file that might contain errors.

CONTINUE: Continue Processing After an Error Occurs

The CONTINUE option prevents SCRIPT/VS from terminating the formatting of your file unless a "severe" or "terminal" error occurs.

DUMP: Enable the .ZZ [Diagnostic] Control Word

The DUMP option is useful when the .ZZ control word is included in the input file. The .ZZ [Diagnostic] control word specifies control blocks and data areas to be displayed (or printed) only when the DUMP option is specified.

DUMP has no effect unless your file contains .ZZ control words.

The DUMP option and the .ZZ [Diagnostic] control word are useful only for debugging the SCRIPT/VS program product. See the .ZZ control word description for details about data areas being dumped.

MESSAGE: Control Information in Error Messages

The MESSAGE option allows you to specify when messages are printed, whether or not the message number is to be included, and how the line causing the error was imbedded. The MESSAGE option parameters are:

- DELAY, which accumulates all error messages in the file named DSMUTMSG. SCRIPT/VS prints the message file at the end of the formatted document or includes a message output file with the document's output file if the FILE option of the SCRIPT command is specified.

  DELAY is useful, especially when the input file is printed, because messages are normally sent to your terminal (that is, not printed).

- ID, which identifies each message with its message number. The message number can be used to refer to a more detailed description of the error.

  The message number is also needed by the PSR (Programming Service Representative) if you should need help via an APAR or the RETAIN SEARCH facility.

- TRACE, which enables the trace-back function for those messages that require it. If an error occurred within a file that is imbedded by many other files, the TRACE parameter identifies the previously imbedding files.

NOSPIE: Prevent Entering SPIE Exit Routines

SCRIPT/VS ordinarily establishes a SPIE (Specified Program Interrupt Exit) before formatting begins. Subsequently, if a program check occurs, the system control program passes control to the SPIE routine, allowing SCRIPT/VS to terminate itself. The NOSPIE option inhibits this function, and allows dynamic debugging tools to handle program check processing.

NUMBER: Print the File Name and Line Number

The NUMBER option tells SCRIPT/VS to print the file name and line number of the last-read input line next to each output line. If an error occurs, you can easily locate the area where the error was detected.

PAGE: Selectively Print Pages

The PAGE option allows you to print (or display) part of your formatted document, rather than requiring you to print the entire document.

SPELLCHK: Verify Spelling

The SPELLCHK option enables the .SV [Spelling Verification] control word. Any word not found in the current main, user, or addenda dictionaries is listed with an error message, along with the input line that contained the word.

Because the list of unverified words may be long, you may want to delay message printing when you specify the SPELLCHK option:

```
SPELLCHK MESSAGE (DELAY)
```

Alternatively, since SCRIPT/VS issues the message listing unverified words by executing the .UW [Unverified Word] control word, you may wish to write a .UW macro. When macro substitution is in effect, your macro will be given control instead of the .UW [Unverified Word] control word.

A .UW macro can be used to:

- Eliminate redundant "errors" on unverified words which appear often in a document, by adding such words to the addenda dictionary with the .DU [Dictionary Update] control word.

- Create a permanent file containing all unverified words, by using the .WF [Write To File] control word to record such words.

The SPELLCHK option should be used infrequently, to identify spelling errors and to identify words that should be in the addenda dictionary. You might want to print, edit, and revise several draft copies before formatting the final draft with spelling verification.

TWOPASS: Provide Two Formatting Passes

The TWOPASS option allows you, in an interactive environment, to see all error messages _before_ the formatted document is displayed. (The MESSAGE (DELAY) option displays the messages _after_ the document is displayed.)

When you are working with a very long input file, you can format it in an interactive environment just to detect and correct any errors. With the TWOPASS option, SCRIPT/VS formats but does not display the file on the first formatting pass. All detected errors are displayed, however, before the second formatting pass starts. By inserting the .QQ [Quick Quit] control word at the end of your input file and using the TWOPASS option, you can display all detected errors and not begin the second formatting pass.

If you do not use the CONTINUE option, formatting will stop with the first error. The second pass (and actual output) will not occur unless the file is error-free.

UNFORMAT: Print All Input Lines Without Formatting

The UNFORMAT option allows you to print an input file without formatting it. All input lines (control words, GML tags, macros, symbols, and text input lines) are printed as entered. In addition, other input lines are included as a result of processing the .IM [Imbed] and .AP [Append] control words.

CONTROL WORDS TO ASSIST DEBUGGING

Some of the SCRIPT/VS control words that are useful when diagnosing problems in an input file are described below.

Spelling Verification

The .SV [Spelling Verification] control word is used to check the spelling of words in an input file. The .SV control word is enabled by the SPELLCHK option of the SCRIPT command. See "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171 for details about verifying spelling.

Tracing SCRIPT/VS Processing

One of the most powerful SCRIPT/VS control words is the .IT [Input Trace] control word. This allows you to see the steps taken by SCRIPT/VS when it substitutes a value for a symbol name. You can also see the step-by-step processing of the control words that make up a macro or GML tag's APF. The .IT control word has many other capabilities that allow you to trace specific events during SCRIPT/VS processing.

The Output Line Generated by Input Tracing

When input tracing is activated, SCRIPT/VS generates one or more output lines that describe the sequence of processing required for the input line about to be executed. These lines are displayed as though they were messages: they are written to the same output destination as messages. Each generated output line is in the form:

```
*¢* [name] [nn]   x  <source line>
```

where:

¢ is a code that identifies why the "current source line" is being traced:

C: Control word trace
G: GML substitution trace
M: Macro substitution trace
S: Symbol substitution trace
*: Symbol table snap

name identifies the name of the "current source line." This is usually the name of the file or macro currently being processed. If the name is in parentheses, the current source line does not come from the file or macro currently being processed:

(ATT)	The current source line displays an attribute of the GML tag being scanned.
(BT n)	The current source line comes from a previously saved running bottom title definition.
(FNLDR)	The current source line comes from a previously saved footnote leader definition.
(RHEAD)	The current source line comes from a previously saved running heading definition.
(RFOOT)	The current source line comes from a previously saved running footing definition.
(RULES)	The current source line displays the rules which will be used in scanning the current GML tag.
(SCAN)	The current source line displays the text which will be scanned for GML attributes.
(TT n)	The current source line comes from a previously saved running top title definition.
(VATT)	The current source line displays the value attributes of the current GML tag.

nn is the line number of the "current source line," either within a file or within a macro.

x is the length (number of characters and blanks) of the "current source line."

current source line is the line being traced by SCRIPT/VS. The following description assumes that all traceable events are being traced: control word tracing, symbol substitution tracing, and macro substitution tracing (as specified with .IT ALL):

- When the current source line contains only text, it is not displayed as part of the input trace.
- When the current source line contains a control word (*C*), SCRIPT/VS displays the current source line and then performs the control word function. However, if the STEP parameter of .IT is specified, you can change a "control word" current source line <u>before</u> it is executed. SCRIPT/VS then executes the <u>modified</u> current source line (as described in "Stepping through an Input Trace" later in this chapter).
- When the current source line contains a GML tag (*G*), SCRIPT/VS displays the name of the GML tag and the APF which is called to process it. If the GML tag has attributes, subsequent lines display the line scanned and the attribute rules used in scanning it.
- When the current source line contains one or more symbols (*S*), SCRIPT/VS:
  - Displays the line as it is (*S*) before any symbols are substituted.
  - Displays the line repeatedly, each time showing the next stage of substitution, until each defined and null-valued symbol has been replaced with its value (*S*). Undefined symbol names are regarded as text.
  - At this point, the line is processed as a line of text, or is traced as a control word current source line (*C*) (as described above).

- When the current source line is from a macro expansion (*M*), SCRIPT/VS:
  - Displays the line as it exists in the macro (*M*).
  - If the line contains one or more symbols, SCRIPT/VS traces the line as described above for symbol substitution tracing.
  - At this point, the line is processed as a line of text, or is traced as a control word (*C*) as described above.

Capabilities of the .IT Control Word

The above description made assumptions that allowed a simplified presentation of input substitution tracing. However, the .IT [Input Trace] control word allows you to trace events much more selectively, and to only trace events that interest you.

- When you want to display all traceable events processed by SCRIPT/VS, specify:

  ```
  .it all
  ```

- When you want to trace only symbol substitution (and no other traceable events) specify:

  ```
  .it sub
  ```

- When you want to trace only macro expansions (and no other traceable events) specify:

  ```
  .it mac
  ```

  Symbols that are part of the macro expansion are traced. However, symbols that are not part of a macro expansion will not be traced.

- When you want to trace occurrences of control words that interest you, specify:

  ```
  .it ctl .xx .yy .zz
  ```

  For example, to trace each occurrence of the .IN [Indent], .IL [Indent Line], and .OF [Offset] control words, specify:

  ```
  .it ctl .in .il .of
  ```

  The .IN, .IL, and .OF control words are added to the list of control words currently being traced, called the "control word table."

  When you want to stop tracing for control words, but want to continue the input trace for other kinds of input items previously specified, issue

  ```
  .it ctl
  ```

  The CTL parameter of the .IT control word clears the list of control words being traced.

- When you want to stop tracing control words, but leave the control word table intact for tracing later, issue

  ```
  .it off
  ```

- When you want to turn off all input tracing, specify:

  ```
  .it off
  ```

As noted above, the OFF parameter stops tracing, but does not clear the control word table. When you want to resume tracing the control words currently in the table, issue:

```
.it on
```

To add control words to the control word table, issue:

```
.it ctl .xx .yy
```

- When you want to display the current value of a macro or symbol, specify the SNAP parameter of the .IT control word. For example, if you want to find out the current definition of the @LIST macro specify:

  ```
  .it snap @LIST
  ```

  The current definition of any symbol, as well as any macro by that name, is displayed only once, not continuously. The SNAP parameter does not affect other parameters of the .IT control word, and can be specified even when input tracing is turned off.

Stepping through an Input Trace

The discussion so far assumed that merely displaying the sequence of SCRIPT/VS operations is sufficient for diagnostic purposes. However, SCRIPT/VS allows you to "step through" the lines being traced in an interactive fashion. Each line of the trace is displayed. When the "current source line" contains a control word (*C*), it is not executed immediately after display. SCRIPT/VS displays the line and waits for your response from the terminal. You specify the "step through" function with:

```
.it step
```

Other input trace functions remain in effect. When a traceable event displays a control-word current source line, SCRIPT/VS displays the line and waits for your response. Therefore, the STEP function cannot occur when you format the file with the MESSAGE (DELAY) option specified. The traced lines are displayed as though they were messages, and you cannot respond to a "delayed" message.

The procedure performed for step-by-step control word tracing is:

1. Display the control-word current source line (*C*) at the message destination (that is, your terminal).

2. Wait for your response.

3. Process the response, which might result in:

   a. Executing the traced control-word current source line, or

   b. Displaying a new current source line to be processed before, after, or instead of the traced control-word current source line.

   c. Identifying the function currently reading input from the terminal.

The responses that you can provide interactively are:

- Null (press the ENTER or RETURN key): the traced control-word current source line is executed and SCRIPT/VS continues processing until it encounters the next traceable control-word line.

- STK input-line: means "stack this input line." The traced control word line is executed. The stacked input line is put on a stack and becomes the next control word to be executed (or, if it is traceable, traced before it is executed) <u>after</u>

the currently traced control word line completes its execution.

For most control words, the traced control word executes and then the stacked input-line control word executes. However, if the traced control word is .IM [Imbed], traceable control word lines from the imbedded file are traced and executed before the stacked input line executes. The control words .TE [Terminal Input] and .AP [Append] prevent the immediate sequence of the input line in a similar way.

- PRE input-line: executes the input line before the traced line. The traced current source line is put on a stack. The PRE input line becomes the current source line. If the PRE input line is not a traceable control word, SCRIPT/VS executes it and continues processing until it encounters the next traceable control word line.

  When the PRE input line is a traceable control word line, it is displayed and SCRIPT/VS waits for your response. Your response can be any response allowed for a traced line.

- REP input-line: replaces the traced line with the input line. The traced input line is not executed nor is it put on a stack. Instead, the REP input line becomes the current source line. If the REP input line is not a traceable control word line, SCRIPT/VS executes it and continues processing until it encounters the next traceable control word line.

  If the REP input line is a traceable control word line, it is traced before execution (like any other traceable control word line). You can now enter any response allowed for a traced line.

- Data line: "PRE input-line" is assumed. SCRIPT/VS processes the data line as described above for PRE.

- ?: does not affect the input trace line. SCRIPT/VS identifies the "reader" of terminal input: either "TERMINAL INPUT" when the .TE [Terminal Input] function is expecting your input, or "CONTROL TRACE" when the .IT STEP function is expecting your input.

Note: To get out of step-by-step input tracing, enter a STK, PRE, or REP with one of the following as a data line:

".it off", to turn off all input substitution tracing,

or

".it run", to resume normal input substitution tracing (that is, to stop the STEP function),

or

".qq", to terminate the SCRIPT/VS formatting job immediately.

Using Terminal Entry to Test a Control Word Sequence

A useful tool for testing SCRIPT/VS control word sequences is the two-line input file (user-created) called TEST:

```
.ty Enter SCRIPT input:
.te on
```

When you process the file with the SCRIPT command in an interactive environment,

```
SCRIPT TEST (CONTINUE NOPROFILE)
```

you get the message

```
Enter SCRIPT input:
```

which resulted from the .TY control word. You can enter control words, macros, symbols, GML tags, and text. Each terminal-entered input line is processed immediately by SCRIPT/VS and is used to build a page. When the page is full (or when a page eject occurs), SCRIPT/VS displays the completely formatted page before accepting additional input lines from the terminal.

To end processing and exit to your interactive environment (CMS or TSO), you can enter:

.QQ [Quick Quit] to end all formatting immediately.

.EF [End of File] to end formatting and close the terminal input file.

.TE [Terminal Input] OFF, to turn off the previous .TE ON and (within the context of the TEST file) end formatting.

.QU [Quit] to display the current output page and then end formatting.

The .QQ [Quick Quit] control word ends processing immediately without the final output page being displayed; you will not see the data that has been formatted for the final output section but not yet displayed.

For testing and diagnosing macros and control word sequences, the first input line you enter might be:

```
.it all
```

All subsequent traceable input lines are traced.

When the page eject occurs (you can force a page eject with the .PA [Page Eject] control word), SCRIPT/VS displays the formatted output accumulated so far. You can then resume terminal input.

Be sure to write down whatever you want to save for future use. The "input file" during terminal input is your terminal keyboard. What you enter is not saved in a disk file. You can create a disk file with input line sequences you want to repeat, and then imbed the file from the terminal whenever you want it, by typing

```
.im filename
```

Putting Messages In Macros

When you build a macro (or an APF), you can use the .IF control word to detect errors in input or syntax. The .MG control word allows you to notify the user that an error occurred. Your error message should include a message number and a brief, clearly-written description.

CHAPTER 19. EASYSCRIPT

EasySCRIPT is an early implementation of GML that existed in SCRIPT/370. Before deciding to use EasySCRIPT, you should review the current SCRIPT/VS GML, which is described in the Document Composition Facility Generalized Markup Language: Starter Set Reference.

EasySCRIPT functions are built into the formatter. You don't use any profile or symbol and macro library with EasySCRIPT. EasySCRIPT is designed to be easy to use, but not flexible. Since the EasySCRIPT functions are built in, you can't tailor them to your own installation's requirements, as you can with SCRIPT/VS GML.

EasySCRIPT provides formatting shortcuts that take advantage of SCRIPT/VS to offer a simple way to format many documents. EasySCRIPT tags can be freely intermixed with standard SCRIPT/VS control words. Using these shortcuts, you can:

1. Produce numbered, unnumbered, or bulleted lists automatically.

2. Automatically format headings and a table of contents. And, if you want, you can have EasySCRIPT number your headings using a decimal numbering system. Then, when you add or delete information, the numbering is changed for you.

3. Format text in paragraphs aligned with the current indention level of a list or heading section.

The built-in EasySCRIPT functions can be invoked in either of two ways:

1. As parameters of the .EZ control word. For example, to get the "B" EasySCRIPT function, which formats a bulleted item, you could enter

   ```
   .ez B text of the bulleted item.
   ```

2. As EasySCRIPT "tags." One of the functions of EasySCRIPT is to define a series of symbols that act as tags to substitute the appropriate .EZ control word. These are not true GML tags in the sense that they are delimited with the symbol delimiter (&), not the GML delimiter (:). The reason for this is that EasySCRIPT tags have different meanings if entered in uppercase than if entered in lowercase. GML tags are not sensitive to the case in which they are entered. The control word

   ```
   .ez on
   ```

   enables the EasySCRIPT tags. Each EasySCRIPT tag has the same name as the equivalent parameter of the .EZ control word. The EasySCRIPT tags are included in SCRIPT/VS to allow documents already marked up with them to be processed by SCRIPT/VS.

You can use the EasySCRIPT functions in your own symbols and macros.

EASYSCRIPT TAGS

There are five EasySCRIPT tags. Each tag provides two different sets of functions, depending upon whether it is capitalized or not. The rule is that the capitalized version provides more function.

The five basic tags are:

1. &Hx -- Inserts a decimal numbered heading of level x where x is 1, 2, 3, 4, 5, or 6.

To create documents without the decimal heading numbers, type the "h" in the heading tag in lowercase.

2. &P -- Starts a new major paragraph. A major paragraph resets the indention to zero and produces the necessary spacing.

   To maintain the current indention for a minor paragraph (that is, within a list), type the paragraph tag with a lowercase "p".

3. &Nx -- Inserts a numbered item of level x where x is 1, 2, 3, or 4.

   If you do not want items numbered, enter the tag with a lowercase "n". A list is itemized at the level of indention associated with the number in the tag (level 1, 2, 3, or 4).

4. &B -- Inserts a bulleted item (one that begins with a •) under the current paragraph or numbered item.

   Sub-bullets (items that are introduced with hyphens) may be entered under bulleted items by typing the bullet tag with a lowercase "b".

5. &toc -- Generates a table of contents.

As you can see, all five EasySCRIPT tags begin with an ampersand (&). A tag may be connected to the line that follows with a period or with one or more blanks:

```
&TAG.line
```

is the same as:

```
&TAG line
```

EasySCRIPT Formats

The EasySCRIPT tags for numbered headings, lists, and paragraphs keep track of the current number of an item and the level of indention. It is good practice, if you are using EasySCRIPT, to use it consistently throughout a document. If you duplicate the function of EasySCRIPT, for example, by manually numbering an item, you will lose the benefit of having the other items numbered automatically.

There is no problem, of course, using any SCRIPT/VS control words that do not duplicate the functions of EasySCRIPT.

HEADINGS

Within the text, headings are automatically numbered (when requested) and formatted by EasySCRIPT, regardless of whether you enter them with uppercase or lowercase characters. However, headings placed in the table of contents by EasySCRIPT appear with the number (if requested) and in the same case as they were entered in the input file.

The numbering scheme used by EasySCRIPT when you invoke the uppercase heading tags is a decimal system: headings may be numbered 1.0, 1.1, 1.1.1, 1.2, 1.2.1, 1.2.2, 1.2.3, and so on.

For documents not requiring decimal numbering, enter the heading tag in lowercase (&h1 through &h6). Decimal-numbered headings and nondecimal-numbered headings may be mixed.

When headings are processed, all indentions (from numbered items and bullets, for instance) are reinitialized and all numbered item counters are reset.

A variation of the &Hn tag is &An, which may be used to automatically number appendixes with letter prefixes such as A.0, B.1.1, C.2.1.3, and so on.

Setting the Heading Counter

If you need to manually control the number of a particular heading (for example, if you turn EasySCRIPT off and then want to turn it back on again), you can specify the number of the last heading on the .EZ control word:

```
.ez on &xref
```

The symbol &xref is the counter used by EasySCRIPT to keep track of the heading numbers it is using. This symbol is undefined until all headings have been issued or until it is explicitly set as described below.

To set a number explicitly for EasySCRIPT to use as the last heading number, you can enter:

```
.ez on 3.0
```

After this control word is processed, the next level two heading (&H2) will be numbered 3.1, the next level one heading will be numbered 4.0, and so on. If you do not specify a number or &xref, then the last heading is considered to have been 1.0.

EasySCRIPT Heading Defaults

The default characteristics for headings associated with EasySCRIPT vary from those used by the .H0 - .H6 [Head Level 0 - 6] control words. If you enter the .EZ ON control word, these values are in effect for .H1 through .H6. When you enter .EZ OFF, the normal values are restored.

Figure 33 on page 359 gives the default characteristics of headings used in EasySCRIPT. Remember that if the heading tag is entered in uppercase, the heading is assigned a number; if entered in lowercase, it is not numbered. Otherwise, the characteristics are the same.

You can change the default characteristics of EasySCRIPT heading tags with the .DH [Define Head Level] control word. The change affects only the EasySCRIPT head level, or only the non-EasySCRIPT head level, whichever is currently in effect.

CROSS-REFERENCES TO EASYSCRIPT HEADINGS

EasySCRIPT has a cross-reference feature you can use to refer to the heading numbers that are generated by EasySCRIPT. The symbol "&xref" is the counter used by EasySCRIPT to keep track of the heading level. If you set another symbol using this symbol, for example,

```
.se intro = &xref
```

Then you can refer to the symbol &intro in your text:

```
Introductory material is in section &intro..
```

When SCRIPT/VS substitutes this line, the result may be something like:

```
Introductory material is in section 3.1.
```

EXAMPLES OF EASYSCRIPT FORMATTING

The following shows how you might enter EasySCRIPT tags to control the formatting of a document.

PARAGRAPHS

A paragraph is designated when the first three characters of a line are either "&P." or "&p.". There should be either a period or at least one blank before the first character of the paragraph text. The EasySCRIPT paragraph tags insert a blank line between paragraphs.

The paragraph above is entered as follows:

```
&P A paragraph is designated...
There should be no space between...
The EasySCRIPT paragraph tags...
```

If the paragraph tag is capitalized ("&P"), a major paragraph is indicated; this resets enumeration counts and returns the indention to zero. Major paragraphs are used to break out of a series of numbered items.

If the paragraph tag is not capitalized ("&p"), a minor paragraph is indicated. You should use minor paragraph tags within numbered items because a minor paragraph tag does not reset the indention or list item counter.

AUTOMATIC ITEM NUMBERING

Up to four levels of items can be numbered or lettered. The numbering range is from 1 to 99 and the lettering range is from a to z. The use of any level of numbered items reinitializes item counts of deeper levels.

An item at the first level of indention is formed when a line begins with "&N1". Each successive use of "&N1" results in a blank line to separate items, the next higher item number, and its indented text.

The second level of numbered items results from using "&N2" in a similar way to "&N1". In like manner, the remaining levels are obtained by using "&N3" and "&N4". Following is an example of the numbered and indented, and bulleted items:

- Here is a bulleted item at level one. (A bulleted item is formatted at the current indention.)

1. This is item one of a first-level numbered list.

2. This is item two of a first-level numbered list.

   a. This is item one of a second-level numbered list.

   b. This is another item of a second-level numbered list.

      This is a minor paragraph placed underneath a level-two numbered item to illustrate how the indention is maintained.

      - We can put bulleted items under any level of indention.

        - Sub-bullets, too.

UNNUMBERED LISTS

Unnumbered lists can be formatted using the &n1 through &n4 tags. Following are some examples of unnumbered lists:

This is item one of a level one unnumbered list.

This is item two of a level one unnumbered list.

This is item one of a level two list.

BULLETS

Bullets and sub-bullets can be used instead of numbers and letters for indented items. The format of the EasySCRIPT bulleting tags is:

```
&B.Text of bulleted item.
&b.Text of sub-bulleted item.
```

Bullets and sub-bullets may be used beneath any level of indention (see examples above).

TABLES OF CONTENTS

A table of contents is automatically generated (with any calculated decimal numbering) and inserted at the location of the "&toc" tag (similar to the .TC [Table of Contents] control word). All you need to enter is

```
&toc
```

and SCRIPT/VS formats and prints the table of contents.

CHAPTER 20. COMPATIBILITY WITH EARLIER RELEASES OF SCRIPT

This chapter describes the differences between SCRIPT/VS Release 2 and earlier versions of SCRIPT:

- SCRIPT/VS Release 1 (Program Number 5748-XX9)
- SCRIPT/370 Version 3 (Program Number 5796-PHL)
- SCRIPT/370 Version 1 (Program Number 5796-PAF)

Figure 15 on page 200 shows the changes to the SCRIPT command options, and Figure 16 on page 202 shows the changes to SCRIPT control words. (SCRIPT/370 control words that are obsolete are listed separately, in Figure 14 on page 199, along with the equivalent SCRIPT/VS control word.)

The listed changes are cumulative: SCRIPT/VS Release 2 incorporates and includes changes introduced in SCRIPT/VS Release 1, SCRIPT/370 Version 3, and earlier versions of SCRIPT/370. For details on the functions of individual SCRIPT/VS control words and SCRIPT command options, see the appropriate chapters in this book.

These codes are used in tables in this chapter:

New	The control word or option was introduced in the indicated product.
Changed	The control word or option was changed in the indicated product
V/3	The control word or option was introduced or changed in SCRIPT/370 Version 3.
R.1	The control word or option was introduced or changed in SCRIPT/VS Release 1.
R.2	The control word or option was introduced or changed in SCRIPT/VS Release 2.
Invalid	The option is no longer valid. Its function is performed by a new option in SCRIPT/VS. SCRIPT/VS does not accept or process the old option.

CHANGES TO THE SCRIPT COMMAND

Figure 15 on page 200 shows the changes to SCRIPT command options.

For Release 2, the PRINT, FILE, TERM, and CTF options have been changed to specify mutually exclusive output destinations, and the positioning of the paper with the STOP option has been simplified.

CHANGES TO THE SCRIPT COMMAND LANGUAGE

Figure 16 on page 202 shows the changes made to individual control words. Other differences are discussed below.

Changes from SCRIPT/VS Release 1

- With Release 2, font management is extended to all devices. The .BF [Begin Font] and .PF [Previous Font] control words, and the FONT parameter of the .DH [Define Head Level] control word are always processed. The .DF [Define Font] control word allows you to define fonts when formatting for non-3800 devices, and extends the concept of font manangement for all devices.

The .SF [Save Font] control word performs no function in Release 2; its function has been subsumed by .BF, and the font save stack is now included in the active environment. If you always saved the current font before beginning a new one, and then restored the original font with .PF [Previous Font], you may simply remove the .SF control word, since it no longer serves any purpose. If you have not followed this convention, these statements may be included in your profile to simulate Release 1 font management:

```
    .if &$PDEV eq 3800 .'bf
    .th .dm bf /.'pf /.'bf &*/
    .th .dm sf /.'bf
    .el .dm bf /.cm
    .el .dm pf /.cm
    .el .dm sf /.cm
```

- The .H0 - .H6 [Head Level 0 - 6] control words are no longer implemented with macros. The macros DSMSTDH0-6 and DSMEZSH0-6, which were built and maintained by the .DH [Define Head Level] control word in Release 1, are not supported. If you use only the .DH control word to specify head-level processing, you will be unaware of this change. The Release 2 .DH control word provides all the Release 1 functions, plus several new ones.

  If you explicitly modified the DSMSTDH0-6 or DSMEZSH0-6 macros after they were built by .DH, you can write a .DH macro which simulates the Release 1 control word by building similar macros. The file DSMSTDH, provided with SCRIPT/VS, gives an example of such a macro.

- Columns to the left of a forced column are no longer ineligible for balancing. Each set of nonforced columns within a section will be balanced separately, and skips at the top and bottom of each set of balanceable columns will be discarded before balancing.

- The effects of widow zones may cause results which differ from Release 1. The default for widow handling is "on".

- .SP [Space] and .SK [Skip] requests for more than column depth will be reduced to column depth, and will not be split between columns. With Release 2, when line spacing is set to more than one by the .DS [Double Space Mode], .LS [Line Spacing], or .SL [Set Line Space] control words, the extra vertical white space precedes the text line, rather than follows it, and is discarded when the line falls at the top of a column.

- The maximum depth of a footnote has been increased to one quarter of the page depth. When a footnote exceeds this depth, it is automatically ended and a new footnote started. Footnote leaders always take effect on the next page.

- White space generated because of the .SP [Space] and .SK [Skip] control words will be flagged with the current revision code character if it is the same as that assigned to the preceding text line.

- Substitution is performed on the comparands of the .IF [If], .AN [And], and .OR [Or] control words even if symbol substitution is off. The comparands are no longer limited to eight characters in length, and must be compared with substitution off if either comparand contains blanks or parentheses.

- A trailing delimiter is no longer required of lines read from a SCRIPT/VS macro library. The GML starter set macro library provided with Release 1 of SCRIPT/VS used the colon (:) as the macro line delimiter; for compatibility, trailing colons will automatically be removed from lines read from a macro library.

- Automatic underscoring provided by the .US [Underscore], .UC [Underscore and Capitalize], .DH [Define Head Level], and .DF

[Define Font] control words underscores all nonblank characters. By default, blanks are also underscored. The .UD [Underscore Definition] control word may be used to indicate only whether or not blanks are to be underscored.

When automatic head-level numbering and underscoring are requested with .DH, the heading will be underscored, but the section number will not. If a font is specified with .DH that was defined to include underscoring, both the section number and the heading will be underscored.

- With Release 1, output comments were written to the output destination before the section in which surrounding text appeared, and were prefixed by a hexadecimal 09 carriage control character unless the device type was TERM. With Release 2, output comments of the line form are written to the output destination before the page on which surrounding text appears, and you may supply any carriage control character.

  The Release 1 output comment function may be simulated with this macro:

  ```
      .su off
      .dm oc() /.sc
      .dm oc() /.pm 0
      .dm oc() /.'if SYSOUT eq TERM .'oc '&*
      .dm oc() /.'el .'oc '&X'09.&*
      .dm oc() /.pm
      .dm oc() /.mc
      .su on
  ```

- The blank separating a control word name from its first parameter is optional. If the blank is omitted, SCRIPT/VS will insert it, unless the control word name and first parameter together constitute the name of a defined macro.

Changes from SCRIPT/370 Version 3

- SCRIPT/VS accepts only the 2-character form of control words. SCRIPT/370 allowed you to specify either the two-character name or the control word's long descriptive name.

- SCRIPT/VS sets the symbols &0 through &9 only when the .IM [Imbed] and .AP [Append] control words are processed. SCRIPT/370 also set the same symbols when a macro was invoked.

  Note: In SCRIPT/VS, the number of tokens available with .IM [Imbed] and .AP [Append] is 14.

- SCRIPT/VS sets the symbols &*0 through &*n when a macro is invoked. You should convert all your SCRIPT/370 macros to the SCRIPT/VS form, since the SCRIPT/VS macro processor is much more powerful than the SCRIPT/370 macro processor. In the meantime, you can cause SCRIPT/VS to transfer the macro parameters in &*0 through &*9 to the symbols &0 through &9 to allow unmodified SCRIPT/370 macros to operate correctly. To do this, you must provide a .DM macro that will process all SCRIPT/370 .DM [Define Macro] control words. This macro can be included in your PROFILE file:

  ```
  .su off
  .'dm dm() /.'sa
  .'dm dm() /.'su off
  .'dm dm() /.'se *a=index &V'&* &V'&*2
  .'dm dm() /.'se *b=substr &V'&* &*a
  .'dm dm() /.'se *s=substr &V'&*2 1 1
  .'dm dm() /.'su on
  .'dm dm() /.'dm &*1 &*s..se 0 '&*0&*s..gs vars 1 2 3 4 5 ...
                                                 ... 6 7 8 9&V'&*b
  .'dm dm() /.'re
  .su on
  ```

- For .SE [Set Symbol] control words that set a symbol to the current page number, the form ".se name = &", SCRIPT/VS sets the symbol to the current page number, including its prefix, if any, in its character string form. You should be careful not to use the page number in arithmetic expressions when a non-numeric prefix would cause an error.

- SCRIPT/VS accepts space units when you specify a control word's parameter that defines a horizontal or vertical space or displacement. With SCRIPT/370, the amount of space could be specified only as a number of characters or lines. See "Vertical and Horizontal Space Units" on page 6 for details about space units.

  Exceptions to this are the .HS [Heading Space] and .FS [Footing Space] control words, which specify the number of lines available for top and bottom titles.

- SCRIPT/VS maintains a page's layout parameters until it completes formatting the page. Control words that affect a page's layout always take effect on the next page.

  With SCRIPT/370, a control word that affected the page layout (for example, the .PL [Page Length] control word, which changed the number of lines on a page) would take effect on the current page if possible. Otherwise, it would take effect on the next page. The resulting output was sometimes difficult to predict and plan for.

  With SCRIPT/VS, you can establish the next page's layout and then eject to that page. The current page is not affected by the new page layout parameters. Control words that always take effect on the next page are listed in Figure 26 on page 354.

  When SCRIPT/VS begins to process your input file, the first page has not yet been started. Formatting for the first output page begins when there is text for it, or when a control word (for example, .SP [Space]) that requires the page to be started is processed.

  You can specify all of the dimensions for page one when you put their control words before formatting begins for page one, while page one is still the "next page" to be started.

  A new chapter commonly begins with a .PA [Page Eject] control word. In SCRIPT/370, some page layout control words were usually placed before the .PA, and others after it. Assuming the page layout (including new top- and bottom-titles) is to take effect for the new chapter, the proper sequence in SCRIPT/VS is to place all of these control words <u>before</u> the .PA control word. You can include the following .PA macro definition in your PROFILE file to allow this sequence of control words to operate properly:

  ```
  .su off
  .dm pa() /.if &E'&*1 = 1 .an &T'&*1 = N .'pn &*1
  .dm pa() /.if &E'&*1 = 0 .'pa nostart
  .su on
  ```

- The control words .UC [Underscore and Capitalize], .UP [Uppercase], and .US [Underscore] have been changed in SCRIPT/VS to accept the parameters ON, OFF, or a number. In SCRIPT/370, these control words accepted only a single line of text. In SCRIPT/370, the control word ".US 80" would underscore the line consisting of the characters "80", but in SCRIPT/VS the same control word will underscore the next <u>80</u> lines. The following macros can be included in your PROFILE file to make these three control words operate as the SCRIPT/370 equivalents:

```
.su off
.dm up /.'uc off /.'us off /.'up 1 /.'li 1 /&*
.dm us /.'up off /.'uc off /.'us 1 /.'li 1 /&*
.dm uc /.'up off /.'us off /.'uc 1 /.'li 1 /&*
.su on
```

- Certain page layout parameters, whose values were constant in SCRIPT/370, are based on the logical device type in SCRIPT/VS. These parameters and the control words that affect them are listed in Figure 31 on page 357.

THE SCRIPT/370 DICTIONARY

SCRIPT/370 supported a hyphenation-exception dictionary called SCRIPT XDICT. The XDICT dictionary was used to determine how to hyphenate words that were not correctly hyphenated by the hyphenation algorithm. The user could create and modify his own hyphenation exception dictionaries using the HYPEDIT command.

SCRIPT/VS does not support either the HYPEDIT command or exception dictionaries. Instead, SCRIPT/VS provides dictionaries that support both spelling verification and automatic hyphenation in nine languages.

You can also create and update a temporary dictionary for use when your document is being formatted, called the addenda dictionary, using the .DU [Dictionary Update] control word.

SUMMARY OF CHANGES

Figure 14 and Figure 16 on page 202 summarize the changes to individual SCRIPT control words. Figure 15 on page 200 summarizes changes to the SCRIPT command.

Obsolete Control Word	SCRIPT/VS Equivalent Control Word
.BT	.RT [Running Title] BOTTOM
.CO	.FO [Format Mode]
.CW	.DC [Define Character] CW
.EB	.RT [Running Title] BOTTOM EVEN
.EP	.PA [Page Eject] EVEN
.ET	.RT [Running Title] TOP EVEN
.FI	.FO [Format Mode] ON
.FT	.RT [Running Title] BOTTOM
.HE	.RT [Running Title] TOP EVEN
.HN	.RH [Running Heading]
.JU	.FO [Format Mode]
.LS	.SL [Set Line Space]
.NB	.BC [Balance Columns] OFF
.NC	.CO [Concatenate Mode] OFF
.NF	.FO [Format Mode] OFF
.NJ	.JU [Justify Mode] OFF
.OB	.RT [Running Title] BOTTOM ODD
.OP	.PA [Page Eject] ODD
.OT	.RT [Running Title] TOP ODD
.PS	.DC [Define Character] PS
.SF	.BF [Begin Font]
.TT	.RT [Running Title] TOP

Figure 14. Obsolete Control Words: SCRIPT/VS continues to recognize and support these control words, but their functions have been subsumed by more general control words as indicated.

Option	Code	Changes
2PASS	Invalid	Specify TWOPASS instead.
ADJUST	Invalid	Specify BIND instead.
ADJUSTnn	Invalid	Specify BIND instead.
BIND	New R.1	Shift the page image to the right.
CENTER	Invalid	Specify BIND instead.
CENTERnn	Invalid	Specify BIND instead.
CHARS	New R.1	Specify up to four fonts (Valid for the 3800 Printer only.)
CTF	New R.2	Prepare output in STAIRS/VS Condensed Text Format. Mutually exclusive with PRINT, FILE, and TERM.
DEBUG	Invalid	Specify NOSPIE instead.
DEST	New R.1	Specify a remote output station.
DEVICE	New R.1	Specify a logical output device.
	Changed R.2	STAIRS logical device added.
DUMP	New R.1	Enable the .ZZ [Diagnostic] control word to snap SCRIPT/VS control blocks.
FILE	Changed R.1	Specify the name of the output file.
	Changed R.2	Mutually exclusive with TERM, PRINT, and CTF. Default logical device is 1403W6.
LIB	New R.1	Specify up to eight library names.
MARK	Invalid	
MESSAGE	New R.1	Control the timing and destination of messages.
NOPROF	New V/3	Suppress the PROFILE option.
NUMBERnn	Invalid	Specify NUMBER instead.
OFFLINE	Invalid	Specify PRINT instead.
OPTIONS	New R.1	Specify a file that contains additional SCRIPT Command options.
PAGE	New R.1	Specify one or more ranges of pages to print, or request SCRIPT/VS to prompt you to enter page number ranges.
PAGEnnn	Invalid	Specify PAGE(nnn) to print from page nnn.
PRINT	Changed R.2	Mutually exclusive with TERM, FILE, and CTF.
PROFILE	Changed R.1	You can specify the name of a profile.
	Changed R.2	Part of the profile may be executed after the primary input file is finished, providing an epifile facility.
SEARCH	New R.1	Specify a library to be searched for imbedded files. (Not valid in CMS.)
SINGLE	Invalid	Specify PAGE (nn ONLY) to print a single page, nn.

Figure 15. Changes to SCRIPT Command Options (Part 1 of 2)

Option	Code	Changes
SPELLCHK	New R.1	Enable the .SV [Spelling Verification] control word to perform spelling verification.
STOP	Changed R.2	Sheet alignment has been simplified.
SYSVAR	New V/3	Set Symbol values from the command options.
TERM	New R.1	Display formatted output at the terminal.
	Changed R.2	Mutually exclusive with PRINT, FILE, and CTF.
TLIB	New R.2	Specify libraries containing spelling checking and hyphenation dictionaries. (Valid in CMS only.)
TRANSLATE	Invalid	Specify UPCASE to have lowercase translated to uppercase.
TWOPASS	New R.1	Prepare with two formatting passes, and produce output on the second pass.
UNFORMAT	New V/3	Processes the .IM [Imbed], .AP [Append], and .EF [End of File] control words, and reads lines from imbedded files to include in the unformatted listing. Symbol substitution is performed, but the input line is printed as entered.
UPCASE	New R.1	Folds all lowercase letters to uppercase letters before printing.

Figure 15. Changes to SCRIPT Command Options (Part 2 of 2)

Control Word	Code	Description
...	Changed R.1	Labels are allowed in SCRIPT/VS macros.
.AA	New R.2	.AA [Associate APF] defines the mapping between GML tags and APFs, and specifies the rules for scanning the tag.
.AN	New R.2	.AN [And] works in conjunction with .IF and .OR to conditionally process text and control words.
.AP	Changed R.1	Up to 14 tokens can be passed to the appended file. The tokens are not reset when a macro is called. &0 contains the number of tokens passed.
	Changed R.2	An external file name may be specified in parentheses.
.BC	Changed V/3	Operands ON and OFF restore and cancel column balancing.
.BF	New R.1	.BF [Begin Font] specifies the font in which subsequent text is to be formatted.
	Changed R.2	Saves the current font before beginning a new font.
.BM	Changed V/3	The bottom margin can be specified as an increment to or a decrement from the current value.
	Changed R.1	Default based on logical device. Always takes effect on the next page.
.BX	New V/3	.BX [Box] draws automatic boxes.
	Changed R.1	New options for drawing a box within a box, for drawing fragments of boxes, and for drawing parallel boxes.
	Changed R.2	New option for drawing untopped boxes. Vertical rules always overlay text.
.CC	Changed R.1	Ejects to a new column only when not already at the top of a column.
.CD	Changed V/3	You can define up to nine displacements for columns, even if you initially specify only one column. The remaining displacements are used when you later increase the number of columns.
.CE	Changed V/3	Accepts input text as a parameter.
.CL	Changed V/3	The column width can be specified as an increment to or a decrement from the current value.
.CO	Changed V/3	Operands ON and OFF restore and cancel concatenation.
.CP	Changed R.1	Ejects to a new page only when not already at the top of a page.
.CT	New R.2	.CT [Continued Text] concatenates a line to the previous input line with no intervening wordspace.
.DC	New R.1	.DC [Define Character] specifies characters with special meaning to SCRIPT/VS.
	Changed R.2	New parameters indicate how to treat special characters in the index.

Figure 16. Changes to SCRIPT/VS Control Words (Part 1 of 8)

Control Word	Code	Description
.DD	New R.1	.DD [Define Data File-id] maps external file names to SCRIPT/VS file-ids.
	Changed R.2	External file names may contain blanks and other special charcacters.
.DF	New R.2	.DF [Define Font] defines internal fonts, which may be composed of underscoring, overstriking, capitalization, font changes (3800 only), and pauses for typing element changes (typewriter output only).
.DH	New V/3	.DH [Define Head Level] specifies formatting parameters for seven levels of automatic headings.
	Changed R.1	You can specify a font for each head level.
	Changed R.2	New parameters allow you to specify the font of the table of contents entry, automatic heading numbering, etc.
.DL	New R.2	.DL [Dictionary List] specifies the language to be used for spelling checking and hyphenation.
.DM	New V/3	.DM [Define Macro] allows user-written functions to extend or replace SCRIPT control words.
	Changed R.1	You can specify a macro with more than one input line, and store macros in a library. When a macro is invoked, SCRIPT/VS sets local symbols &*0 through &*n (n being the number of tokens passed to the macro; &*0 contains the value n). The macro can set local symbols that begin with &*.
.DU	New R.1	.DU [Dictionary Update] temporarily adds words to the spelling checking and hyphenation dictionary.
	Changed R.2	Specifies which dictionaries will be searched, and in what order.
.EC	New R.1	.EC [Execute Control] executes a control word, even if there is a macro of the same name.
.EF	Changed V/3	CLOSE operand allows you to suspend an input file.
	Changed R.2	When .EF CLOSE is used in the profile, the remainder of the profile is processed as an epifile after the primary imput file is finished.
.EL	New R.2	.EL [Else] is executed only if the most recently executed .IF, .AN, or .OR was false.
.EM	New R.1	.EM [Execute Macro] executes a macro, even if macro substitution is off.
.EZ	New R.1	.EZ [EasySCRIPT] allows EasySCRIPT tags and control words to be more freely mixed.
.FL	New R.2	.FL [Float] specifies that blocks of text are to be kept together and placed at the top or bottom of subsequent columns or pages.
.FM	Changed V/3	The footing margin can be specified as an increment to or a decrement from the current value.
	Changed R.1	Default is based on the logical device. Always takes effect on the next page.

Figure 16. Changes to SCRIPT/VS Control Words (Part 2 of 8)

Control Word	Code	Description
.FN	New V/3	.FN [Footnote] specifies text to be placed at the bottom of the current page.
	Changed R.1	New parameter: LEADER allows you to define leading text lines to precede the first footnote of each page. Column balancing occurs on pages with footnotes. Footnotes are formatted to the line length instead of to the column width.
	Changed R.2	Footnotes may be of arbitrary size, and will be divided into pieces of at most one quarter page depth. Footnote LEADER always takes effect on the next page.
.FO	New V/3	Operands ON and OFF allow you to restore and cancel formatting (concatenation and justification).
	Changed R.1	New parameters: LEFT, RIGHT, CENTER, EXTEND, FOLD, and TRUNC.
.FS	Changed V/3	The footing space can now be specified as an increment to or a decrement from the current value.
	Changed R.1	Always takes effect on the next page.
.GO	New V/3	.GO [Goto] branches to a labelled line.
	Changed R.1	Allowed in SCRIPT/VS macros.
.GS	New R.2	.GS [GML Services] provides a variety of facilities of use in writing GML APFs.
.Hn	New V/3	Head-level control words for head levels 0 through 6. The default characteristics of each head level are changed when you specify the .EZ ON (Enable EASYSCRIPT) control word, and are restored when you disable EASYSCRIPT (with the .EZ OFF control word).
.HM	Changed V/3	The heading margin can be specified as an increment to or a decrement from the current value.
	Changed R.1	Default is based on the logical device. The control word always takes effect on the next page.
.HS	Changed V/3	The heading space can be specified as an increment to or decrement from the current value.
	Changed R.1	The control word always takes effect on the next page.
.HY	New V/3	.HY [Hyphenate] enables hyphenation.
	Changed R.1	Hyphenation is performed using a dictionary.
	Changed R.2	An algorithmic hyphenator may be used in place of or in conjunction with multiple dictionaries.
.IE	New R.2	.IE [Index Entry] is used in formatting automatic indexes.
.IF	New V/3	.IF [If] conditionally processes text and control words.
	Changed R.2	The comparands are no longer limited to eight characters, and will be evaluated even if symbol substitution is off.

Figure 16. Changes to SCRIPT/VS Control Words (Part 3 of 8)

Control Word	Code	Description
.IM	Changed R.1	Up to 14 tokens can be passed to the imbedded file. The tokens are not reset when a macro is called. &0 contains the number of tokens passed.
	Changed R.2	An external file name may be specified in parentheses.
.IN	Changed V/3	An indent can be specified as an increment to or a decrement from the current value.
	Changed R.2	The duration and extent of the indention may be specified, with or without causing a break.
.IR	New R.1	.IR [Indent Right] specifies indention from the right-hand edge of the column.
	Changed R.2	The duration and extent of the indention may be specified, with or without causing a break.
.IT	New R.1	.IT [Input Trace] enables tracing of symbol substitution, macro substitution, and control word execution.
	Changed R.2	New parameter enables tracing of GML substitution.
.IX	New R.2	.IX [Index] causes the index built from .PI control words to be formatted.
.JU	Changed V/3	Operands ON and OFF restore and suspend justification.
.KP	New V/3	.KP [Keep] specifies text that is to be kept together and treated as a single block during column balancing.
	Changed R.1	New parameter: INLINE allows you to keep text with preceding and following text. You can specify .KP n, to keep the following n lines together as an INLINE keep. Formatting environment saved around keep.
.LB	New R.1	.LB [Leading Blank] is executed whenever a line beginning with a blank is processed.
.LI	Changed V/3	Accepts a data line as a parameter. Operands ON and OFF establish and suspend literal interpretation.
.LL	Changed V/3	The line length can be specified as an increment to or a decrement from the current value.
	Changed R.1	Not allowed within a keep.
	Changed R.2	Default is based on the logical device. The control word always takes effect on the next page. When the column width is [implicitly] changed by .LL, the change is reflected in the current page's layout.
.LT	New R.1	.LT [Leading Tab] is executed whenever a line beginning with a tab is processed.
.LY	New R.1	.LY [Library] enables searching of external libraries for symbol and macro definitions.
.ME	New R.2	.ME [Macro Exit] ends a macro and returns to the macro's caller. If text or a control word is specified, it is processed as though it were part of the calling file or macro.

Figure 16. Changes to SCRIPT/VS Control Words (Part 4 of 8)

Control Word	Code	Description
.MG	New R.1	.MG [Message] issues user-defined messages.
.NL	New R.1	.NL [Null Line] is executed whenever a null input line is processed.
.OC	New R.1	.OC [Output Comment] permits user-defined lines to be inserted into formatted output.
	Changed R.2	User-defined strings may be inserted into formatted lines.
.OF	Changed V/3	An offset can be specified as an increment to or a decrement from the current value. Any new .OF [Offset] control word resets the previous offset value.
.OR	New R.2	.OR [Or] is used in conjuntion with .IF and .AN to conditionally process text and control words.
.PA	Changed R.1	New parameters: NOSTART ends the current page without starting the new page. You may then modify the page layout, headings, footings, etc. The next page is not started until SCRIPT/VS encounters either a control word that requires it or input text. ODD, EVEN, ON, and OFF, allow you to page eject to an odd- or even-numbered page.
.PF	New R.1	.PF [Previous Font] restores the most recently saved font.
.PI	New R.2	.PI [Put Index] is used to build an index. The .IX control word is used to format the index.
.PL	Changed V/3	The page length can be specified as an increment to or a decrement from the current value.
	Changed R.2	Default is based on logical device. The control word always takes effect on the next page.
.PM	New R.2	.PM [Page Margins] specifies the binding margin, and overrides the BIND option of the SCRIPT command.
.PN	Changed V/3	New parameters: FRAC initiates fractional page numbering (decimal point pages). NORM restores normal (ascending integer) page numbering and causes a page eject. PREF specifies a character string prefix for all page numbers. ALPH allows you to specify alphabetic page numbering. n allows you to reset the page number. The control word always takes effect on the next page.
.PT	New V/3	.PT [Put Table of Contents] creates table of contents entries.
	Changed R.1	Both control words and text may be written to the table of contents utility file.
.RC	Changed R.2	The position of the revision code in the intercolumn gutter can be changed. Revision codes are now applied to skip and space, but not to running headings or footings.
.RD	Changed R.2	The STOP parameter allows strings to be typed in the middle of formatted column lines.

Figure 16. Changes to SCRIPT/VS Control Words (Part 5 of 8)

Control Word	Code	Description
.RF	New R.1	.RF [Running Footing] specifies text and control words to be formatted and placed at the bottom of every page.
	Changed R.2	Running footings may be surpressed and restored without being redefined.
.RH	New R.1	.RH [Running Heading] specifies text and control words to be formatted and placed at the top of every page.
	Changed R.2	Running headings may be surpressed and restored without being redefined.
.RI	Changed V/3	Accepts a text line as a parameter.
.RN	New R.2	.RN [Reference Numbers] causes line reference numbers to be printed to the right of the page.
.RT	New R.1	.RT [Running Title] combines the obsolete .BT, .EB, .ET, .OB, .OT, and .TT control words.
.RV	New V/3	.RV [Read Variable] sets a symbol to a value read from the terminal.
	Changed R.1	A user can type in characters without having to enclose them in quotation marks. You can specify a file as the terminal input file by using the .DD [Define Data File-id] control word.
.SE	Changed V/3	The OFF operand cancels a symbol value.
	Changed R.1	New parameters allow you to locate one character string within another or extract a substring. The value of a symbol may be taken from an external library.
	Changed R.2	A symbol may be assigned a quoted string which is not examined for inner quotation marks.
.SF	New R.1	.SF [Save Font] saves the current font.
	Changed R.2	Performs no function; the current font is saved automatically by .BF.
.SK	New V/3	.SK [Skip] specifies vertical white space to be discarded if it falls at the top of a column.
	Changed R.1	New parameter: P, specifies page-width skips. If a conditional skip is followed by another .SK or .SP, the longer of the two is used, not the second. .SL governs the size of skip requests expressed in lines, even if A is specified. .SK requests expressed in other space units are not affected by the setting of .SL.
.SL	New R.1	.SL [Set Line Space] specifies the vertical depth of each column line.
.SO	New R.2	.SO [STAIRS/VS Output] sets the STAIRS/VS paragraph id, document name, and user number.

Figure 16. Changes to SCRIPT/VS Control Words (Part 6 of 8)

Control Word	Code	Description
.SP	Changed V/3	New parameters: A specifies absolute spacing. C specifies conditional spacing.
	Changed R.1	New parameter: P specifies page-width space. If a conditional space is followed by another .SP or .SK, the longer of the two is used, not the second. .SL governs the size of space requests expressed in lines, even if A is specified. .SP requests expressed in other space units are not affected by the setting of .SL.
.SU	Changed V/3	Initial value is ON.
	Changed R.2	The line form of .SU no longer turns off subsequent substitution.
.SV	New R.1	.SV [Spelling Verification] enables spelling verification in conjunction with the SPELLCHK option of the SCRIPT command.
	Changed R.2	Initial value is ON.
.SX	New R.1	.SX [Split Text] specifies a line composed of a left-justified string, a right-justified string, and a string to fill the space between.
	Changed R.2	New parameter allows a centered string between the left and right strings.
.TB	Changed R.2	Up to 99 tab positions may be specified, and individual tab positions may be set or cleared.
.TC	New V/3	.TC [Table of Contents] causes the table of contents to be formatted.
	Changed R.1	Rules for n and input line parameters have been changed.
.TI	New R.1	.TI [Translate Input] specifies character translations that are to be performed before any other processing.
.TH	New R.2	.TH [Then] is executed only if the most recently executed .IF, .AN, or .OR control word was true.
.TM	Changed V/3	The top margin can be specified as an increment to or decrement from the current value.
	Changed R.1	Default is based on the logical device. The control word always takes effect on the next page.
.TS	New R.2	.TS [Translate String] translates single characters to strings.
.TU	New R.2	.TU [Translate Uppercase] specifies the translation to be performed when capitalizing text.
.UC	New V/3	.UC [Underscore and Capitalize] underscores and capitalizes text.
	Changed R.1	New parameters: ON and OFF, allow you to capitalize and underscore large blocks of text.

Figure 16. Changes to SCRIPT/VS Control Words (Part 7 of 8)

Control Word	Code	Description
.UD	Changed R.1	The "required blank" is normally not underscored. The required blank defaults to hexadecimal 41.
	Changed R.2	Indicates _only_ whether or not the blank is to be underscored. By default, blanks are underscored.
.UN	Changed V/3	An undent can be specified as an increment to or decrement from the current value.
.UP	New V/3	.UP [Uppercase] capitalizes text.
	Changed R.1	New parameters: ON and OFF allow you to capitalize large blocks of text.
.US	New V/3	.US [Underscore] underscores text.
	Changed R.1	New parameters: ON and OFF allow you to underscore large blocks of text.
.UW	New R.2	.UW [Unverified Word] is executed when unverified words are found during spelling checking.
.WF	New R.1	.WF [Write To File] specifies lines to be written to an external file.

Figure 16. Changes to SCRIPT/VS Control Words (Part 8 of 8)

CONVERTING ATMS-II AND ATMS-III DOCUMENTS TO SCRIPT/VS FORMAT

The ATMS-to-SCRIPT/VS conversion routine is composed of three separate elements:

- The conversion program, which runs as a processor under the control of the Document Library Facility. This processor scans ATMS documents for ATMS-II or ATMS-III formatting controls and substitutes SCRIPT/VS symbols that invoke similar or equivalent formatting functions.
- The ATMS conversion profile, ATMSPRF2, which is used when invoking SCRIPT/VS to format documents that were converted from ATMS. The profile defines to the formatter the substitutions required for the symbols generated by the conversion program.
- The library of SCRIPT/VS macros that are used to emulate the original ATMS functions.

The conversion routine is designed to convert most ATMS controls, GML tags, and implicit keying conventions to similar or equivalent SCRIPT/VS symbols. The output of the conversion routine can then be formatted using ATMSPRF2 and the library of SCRIPT/VS macros supplied to emulate the original ATMS functions.

There are some functions in ATMS that are not directly convertible. Editing of the document may be necessary to achieve the desired formatting results.

ATMS to SCRIPT/VS conversion limitations include:

- Floating skips
- Hyphenation
- Text block indention
- Line controls within split text
- GML
- Office System/6 OCL and special character codes

In many of these areas the most noticeable difference is that the SCRIPT/VS equivalent of the ATMS function may cause a line break.

Conversion Technique

ATMS documents to be converted can be in ATMS FT00 output format or any other sequential format. If the conversion routine is being used for a document in ATMS FT00 format, SCRIPT/VS takes the information contained in the document header records (page width, page depth, and tab settings) and inserts it into the output as SCRIPT/VS symbols. If the conversion routine is being used for a document that is in a format other than ATMS FT00 and the ATMS Application Control Definition (ACD) character is not the default (!), the real ACD character must be passed to the attribute processor using the PARM parameter. See Document Library Facility Guide for details.

Hyphenating Words

In ATMS, hyphens in a word at the end of an input line indicate potential hyphenation points should that word fall at the end of an output line. If the word does not fall at the end of an output line, the hyphens are removed.

The input processor combines the word parts together and builds a .HW [Hyphenate Word] SCRIPT/VS control word to obtain the same effect.

Conversion Program Operation

The ATMS file(s) in FT00 format may be imported into the Document Library Facility or used directly as input to the formatter in batch mode.

Conversion of the ATMS controls and ATMS GML into SCRIPT/VS symbols can be accomplished during an IMPORT or READ operation, or the SCRIPT/VS formatting process. After each access method logical record has been read from the source document, an input processing program that has been associated with the content attribute of ATMS is given control by the Document Library Facility. This input processing program converts the ATMS controls and ATMS GML as described above. When the record conversion is complete, the formatter, or the IMPORT or READ routine, gains control in order to continue with the task.

Non-Format Command Conversion

The following describes the conversion of each ATMS nonformatting control.

End of Embedded Control

The !x is deleted.

ATMS GML Identifier

The !mname is converted to :name (where the : is the default SCRIPT/VS GML delimiter). Whenever the name has had special characters translated to @ (at sign) or truncated to ten characters if necessary, a message is issued indicating the original name and its resultant name. It does not matter whether the name is in upper or lowercase letters.

Subdocument Identifier

The subdocument identifier !i is converted to a .SE [Set Symbol] and some .DM [Define Macro] control words with all of the units that follow the !i being converted to elements of the macro.

The macros thus defined must be known to SCRIPT/VS when formatting documents that reference the macros through the ATMS !m syntax. To accomplish this, the subdocuments containing the macros may be specified on the SCRIPT command statement through the use of the SYSVAR option. For example, to use SCRIPT/VS to format an ATMS document (ATMSDOC) that contains !m's that are defined by another subdocument (SUBDOC), the following command is required:

```
    SCRIPT ATMSDOC (PROFILE(ATMSPRF2) SYSVAR(A SUBDOC))
```

The IBM-supplied ATMS profile document (ATMSPRF2) examines SYSVARs A through J to determine if they have been set. If so, their values are taken as the names of documents to be imbedded prior to the start of formatting of the primary document. This limit of 10 names can be changed by the user by altering ATMSPRF2 at the user's own installation.

Formatting Control Conversion

ATMS formatting controls are identified by the occurrence of an Application Control Definition (ACD) (usually !) and an Application Type Definition (ATD) (t,l,m,f,i,x), and are converted to SCRIPT/VS symbols by the SCRIPT/VS ATMS attribute processor.

It must be understood that in the following descriptions, the ATMS controls are converted to SCRIPT/VS symbols by the attribute processor. The SCRIPT/VS symbols are resolved at format time to control word separators and macros that do not exist in the attribute processor output.

The definitions of the SCRIPT/VS symbols created by the attribute processor are contained in ATMSPRF2.

The conversion macros are defined in ATMS2 MACLIB.

Explicit Paragraphing Specification

The !tf control inter-paragraph space is placed before the paragraph rather than after the paragraph as in ATMS.

!tf; causes text to be formatted corresponding to the parameters set in the previous !tfn1;n2;n3;n4. Note that the control without the following ; resets the format settings to the values set by the first explicit paragraphing !tf in the file (or the default values).

The !tfe ends the explicit paragraphing mode so that paragraphing is controlled again by entry conventions.

Implicit Paragraphing Specification

ATMS recognizes the end of paragraphs by the following conventions:

- A double CR at the end of a paragraph. The use of the double CR does not affect the paragraph spacing in explicit paragraphing.
- Indention of the first line of a paragraph by at least one tab (with certain restrictions).
- Issuing most text format (!t) controls.

The input processor will recognize these conditions in !tf (formatted mode) and insert the appropriate symbols.

Floating Skip

The ATMS floating skip control !t+nn;a is forced to the top of the page.

Width/Depth Control

The ATMS width/depth control !tw when converted causes a line break unlike ATMS.

Text Alignment Controls

!tal, !tar, !tac, and !taj when converted cause a line break unlike ATMS.

Floating Keeps

The !tif control causes all pending floats to be placed on the page.

Text Block Indention

The ATMS indent block control (!tib) can only be partially supported in SCRIPT/VS. The second parameter, the number of blocks to be indented, is only supported for formatted paragraph blocks. In !tu mode it is not supported. The first parameter, the amount of indent for blocks, sets the indention value of all text of the same mode (!tu or !tf).

Page Number Control

The ATMS page number symbol !lpn resolves to the default SCRIPT/VS page number symbol &.

Stop Code

The ATMS typewriter input capability specified by !lsc resolves to a generated bullet character, the same as is done for ATMS operations on the peripheral queues. This is consistent, because the input processor is preparing data, stored in the Document Library Facility, for formatting by SCRIPT/VS. The optional spaces entered by the ATMS user will be removed by the input processor.

Split Text

The ATMS split text control !lst may not be used on a line with other line controls (results are unpredictable).

Revision Markers

The inclusion of markers in the output is controlled in ATMS by the print command option (m). Similarly, when revision markers are to be printed by SCRIPT/VS in documents converted by the ATMS conversion processor, a SYSVAR with the name "M" with any value must be specified; otherwise, the revision marker will not be printed.

Counters

The ATMS counters are handled by two controls, !tset and !lcn of the form:

```
    !tset;identifier;value;style
```

where identifier is

```
    pn-page number
    cN-all counters
    cn-specific counter 0 thru 9
```

value is

```
    0 to 65535
      or
    +0 to +65535
```

style is

```
    a or la for upper and lowercase alphabetic
    r or lr for upper and lowercase roman
    n for arabic
```

These controls are simulated using SCRIPT/VS control words and symbols.

Note: Counters may not be used in split text lines.

Hexadecimal Code	Character	Hexadecimal Code	Character
4A	¢	B0	^{0}
4C	<	B1	^{1}
4F	|	B2	^{2}
5F	¬	B3	^{3}
6E	>	B4	^{4}
8B	{	B5	^{5}
8C	≤	B6	^{6}
8F	┼	B7	^{7}
9B	}	B8	^{8}
9F	■	B9	^{9}
AB	└	BB	┘
AC	┌	BC	┐
AE	≥	BE	≠
AF	•	BF	—

Figure 17. Character Codes Recognized by ATMS-III Conversion: The triplet (character-backspace-character) conventions for special characters defined in ATMS-III Terminal Operations Guide are recognized and translated into a single hexadecimal character.

Triplets and Backspaces

In ATMS there is an entry convention involving backspaces for characters which do not occur on the keyboards but which can be represented on the output printers by graphics. These entry conventions are defined in ATMS-II Terminal Operations Guide and the ATMS-III Terminal Operations Guide.

The input processor will convert defined triplets character-backspace-character to a single hexadecimal character that represents the triplet. All other instances of backspaces are left unchanged.

The special characters and their hexadecimal codes are listed in Figure 17.

ATMS CONTROL - SCRIPT/VS SYMBOL RELATIONSHIP

Figure 18 on page 216 identifies the ATMS controls and the SCRIPT/VS symbols to which they are converted.

The substitution for the SCRIPT/VS symbols is contained in ATMSPRF2 and should be looked at in conjunction with this list.

The contents of each macro which is eventually invoked by the substitution is contained in ATMS2 MACLIB.

ATMS Input	Conversion Output
!fname	&@F name
!iname	.SU OFF
	.SE name = '&@CONT.&@CW..@name'
	.DM name OFF
!lcn;+	&@LC N +
!lda;x	&@LDA X
!lde;x	&@LDE X
!lpn	&@LPN.
!loe	&@LOE.
!los;x	&@LOS X
!lre	&@LRE.
!lrs;x	&@LRS X
!lsc	&@LSC.
text1!lst;xtext2	&@LST @text1@x@text2
!lue	&@LUE.
!lus	&@LUS.
!mname	:name
!t(	&@TBKP
!t)	&@TEKP
!t)(	&@TEBK
!t+nn;x	&@SKIP nn X
!tac;n	&@TAC N
!taj;n	&@TAJ N
!tal;n	&@TAL N
!tar;n	&@TAR N
!tcm	&@TCM
!tds	&@TDS
!tfn1;n2;n3;n4	&@TF n1 n2 n3 n4
!tfe	&@TFE
!thh	&@THH
!thm;n	&@THM n
!tib;n1;n2	&@TIB n1 n2
!tif	&@TIF
!til;n1;n2;n3	&@TIL n1 n2 n3
!tir;n1;n2;n3	&@TIR n1 n2 n3
!tj	&@TJ
!tle	&@TLE
!tls	&@TLS
!tm;n1;n2	&@TM n1 n2
!tnj	&@TNJ
!tnp	&@TNP
!tpd;n1;n2	&@TPD n1 n2
!tps;nxx	&@TPS Nxx
!trs;n	&@TRS n
!tset;id;val;style	&@TSET ID val STYLE
!tss	&@TSS
!ttab;n1;...;nm	&@TTAB n1 ... nm
!ttab-;n1;...;nm	&@TTABM n1 ... nm
!ttab+;n1;...;nm	&@TTABP n1 ... nm
!tts	&@TTS
!tu	&@TU
!tuc	&@TUC
!tufnn	&@TUFnn
!tufcnn	&@TUFCnn
!tuhnn	&@TUHnn
!tuhcnn	&@TUHCnn
!tw;n1;n2	&@TW n1 n2
!twz;n	&@TWZ n
!x	null

Figure 18. ATMS-III Controls to SCRIPT/VS Conversion

CHAPTER 22. COMPATIBILITY WITH TSO/FORMAT

The TSO Data Utility program product provides users of the Time Sharing Option (TSO) of OS/VS2 with a FORMAT function. TSO/FORMAT allows TSO users to enter formatting controls into TSO text data sets that indicate the type of formatting required.

SCRIPT/VS provides TSO/FORMAT users with an easy migration to more powerful formatting. SCRIPT/VS control word syntax is identical for many TSO/FORMAT control words, and SCRIPT/VS control words which are new to TSO/FORMAT users provide many new or enhanced functions.

Creating a TSO/FORMAT-Compatible Environment

SCRIPT/VS provides a symbol and macro facility which allows you to process TSO/FORMAT documents without modifying the documents themselves. Figure 19 lists the TSO/FORMAT control words that are not directly supported by SCRIPT/VS. You can define a SCRIPT/VS macro with the name of the TSO/FORMAT control word that executes the equivelent SCRIPT/VS control word. See "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147 for details.

You can place your macro definitions in a Profile to ensure that they are always available when you process TSO/FORMAT documents. For details, refer to "PROFILE: Specify a Profile" on page 28.

The SCRIPT Command in TSO

The SCRIPT command is used to call SCRIPT/VS to format an input file, and is similar to the FORMAT command. See "Chapter 2. Using the SCRIPT Command" on page 13 for details about the SCRIPT command, its options, and its TSO naming conventions.

TSO/FORMAT Control Word	SCRIPT/VS Equivalent Control Word
.AD [Adjust]	.RI [Right Adjust]
.BL [Blank]	.TR [Translate Character]
.EN [End]	.CE [Center] OFF
.FI [Fill]	.FO [Format Mode] ON
.HI [Hanging Indent]	.UN [Undent], .OF [Offset]
.NF [No Fill]	.FO [Format Mode] OFF
.NJ [No Justify]	.FO [Format Mode] LEFT
.PI [Paragraph Indent]	.PP [Paragraph Start]
.RP [Reprint]	
.ST [Stop]	.QU [Quit]

Figure 19. Unsupported TSO/FORMAT Control Words: A SCRIPT/VS control word which provides an equivalent function is listed for each TSO/FORMAT control word, except .RP [Reprint]. SCRIPT/VS provides no equivalent of the Reprint function.

CHAPTER 23. SCRIPT/VS CONTROL WORD DESCRIPTIONS

This section describes each control word in the SCRIPT/VS language. All parameters are shown with descriptions of their effect on processing. Usage notes and examples are included.

CONTROL WORD SYNTAX

All control words have two-character names. A control word is identified by a period (.) in the first character position of an input line, followed by the two-character name. If the control word accepts parameters, they follow the control word name and are separated from each other by blanks:

```
 .du add raccoon giraffe llama
```

The blank separating the control word name from the first parameter is optional; if you omit it, SCRIPT/VS will insert it. Thus,

```
 .cecenter this line
```

will be processed as

```
 .ce center this line
```

Note: If you omit the first blank, and the control word name and first parameter together form a valid macro name, the macro will be processed, rather than the control word, if macro substitution is on.

The Control Word Separator

The control word separator character may be used to enter several control words on a single line:

```
 .sk .5i;.fo on;.in 10m
```

SCRIPT/VS scans every control word line for the control word separator character. If one is found, the line is divided at that point, and the part of the line before the control word separator is processed as a complete control word line. The remainder, to the right of the control word separator, becomes the next input line. The period in ".fo on" in this example appears in the first character position, allowing ".fo" to be recognized as a control word.

The control word separator character may also be used to place a control word within a line of text. For example,

```
 an under;.us on;score;.us off;d word.
```

results in:

an under<u>score</u>d word.

SCRIPT/VS also scans every text line for the control word separator character. If one is found, <u>and</u> is immediately followed by a period and a two-character control word name, the line is divided at that point. The part of the line preceding the control word separator is processed as a line of text with continuation, and the remainder of the line, to the right of the control word separator, becomes the next input line. If a control word separator character is found in a text line, but is not followed by a control word, it is treated as text.

Note: Macros are not recognized in text lines. The .EM [Execute Macro] control word must be used to process macros in text lines.

The character to be used as the control word separator may be changed with the .DC CW [Define Character] control word.

Macros

SCRIPT/VS macros are invoked in the same way as control words with a period in the first character position of an input line. Macro names, however, may be up to ten characters long. Parameters may be specified on a macro line in the same way as on a control word line. If a macro called "brachiate" were defined with the .DM [Define Macro] control word, it would be invoked as a control word:

```
 .brachiate parm1 parm2
```

If a macro is defined with the same name as a control word, the macro will be processed instead of the control word when macro substitution is on. This allows you to redefine the function of a control word.

The Control Word Modifier

The SCRIPT/VS control word processor recognizes a single quotation mark (') after the period as a control word modifier. Any control word can be entered with the modifier:

```
 .'ce Center this line.
```

The control word modifier changes the usual operation of the control word processor in two important ways:

1. No macro search is done. Even if a macro of the given name exists, the control word is invoked, not the macro.

2. No control word separator scan is done. Any control word separators in the line are left there as ordinary text characters. Thus, a control word entered with the control word modifier must be the last control word on that line.

Since no control word separator scan is done, a control word that accepts a line of text may be entered with the control word modifier to protect any separator characters that appear in the line as part of the text:

```
.'ce centered line; one line.
.'h3 Using the ; in text
```

However, if the line contains a symbol that, when resolved, contains a control word separator character in the first position of the new input line, it will not be ignored. You must either specify a .DC CW OFF immediately before the line or you must use the control word separator symbol (&$CW).

<u>Type 1 Control Words</u>

There are several control words that all have the same syntax and accept the same parameters, called Type 1 control words. All the Type 1 control words are analyzed by a common preprocessor before the individual control word processors get control, and they therefore have certain things in common. (There are other control word types as well, and their syntax is explained in the individual control word descriptions.)

The fictitious control word .tl is used in this discussion to represent any Type 1 control word:

.tl	**[<u>1</u> / n / ON / OFF / line]**

<u>where:</u>

n is a positive integer that indicates the number of input lines to be processed by the Type 1 control word. The default is 1, meaning that the next input line after the control word is to be processed by this control word.

ON starts an open-ended range of input lines to be processed by the Type 1 control word, until terminated with the OFF parameter.

OFF stops the effect of the Type 1 control word, whether it was started with the ON parameter, or with a number in "n" that has not yet been exhausted.

line is a single line that is to be processed by the Type 1 control word. The single input line

```
.tl this is a line
```

is equivalent to the two lines

```
.tl 1
this is a line
```

The line given on a Type 1 control word is assumed to start with the first nonblank character. Thus, the following two forms operate identically:

```
.tl this is a line
.tl   this is a line
```

The keywords ON and OFF and numbers given in "n" are recognized only if they are the only parameters on a control word line. If there are other parameters, it is assumed to be a "line" to be processed. Thus, Type 1 control words in the form:

```
.tl On old Olympus' towering top
.tl 555 Bailey Avenue
```

are taken as control words that have a line of text, not as requests to process large numbers of input lines.

<u>Space Units</u>

All control word parameters that specify horizontal or vertical dimensions may be specified in any recognized space units, unless otherwise noted in the control word description. The recognized space units are:

```
Centimeters - aaCM
Character spaces - aa
Ciceros/Didot points - aaCbb
Em-spaces - aaM
Inches - aaI
Line spaces - aa
Millimeters - aaMM
Picas/points - aaPbb
```

where aa is any valid number. Space amounts in inches, centimeters, and millimeters may have up to two decimal positions.

NOTATIONAL CONVENTIONS

The format of each control word is described as shown above in a format box. The notation conventions used are:

1. Keywords that must be entered as shown are in UPPERCASE. If the keyword can be abbreviated, the abbreviation is shown in uppercase letters, and the rest of the word in lowercase, as in ROman. (You may enter the control word and the keyword in uppercase or lowercase.)

2. Parameters for which you must supply the value are shown in lowercase letters.

3. If there is a default value for a parameter, it is underscored. In some cases, one parameter can have a default but other parameters must be specified.

4. The initial setting and default of a control word are not always the same. For example, for the .MS [Macro Substitution] control word, the initial setting is OFF, but the default is ON.

5. A single optional parameter is shown in [small brackets]. This parameter may be specified or omitted.

6. A parameter that allows you to choose one of several possibilities, or none, is shown as a list enclosed in large brackets, as in the Type 1 example above.

7. A list enclosed in {braces} indicates that one of the choices must be specified. For example, the notation

   ```
   { ON      }
   { OFF     }
   { INCLUDE }
   { IGNORE  }
   ```

   signifies that this parameter must be specified as ON, OFF, INCLUDE, or IGNORE.

8. A single required parameter is shown without any brackets or braces.

9. If the format box has internal horizontal lines, as in the .DM [Define Macro] control word description, each segment of the box depicts an alternative form of the control word.

10. An ellipsis (...) indicates that a parameter can be repeated. The form "d1 ... d9" indicates that you may specify up to nine "d" values, separated by blanks. The form "c ..." indicates that you may specify as many values of c as will fit on that input line.

... [SET LABEL]

The ... [Set Label] control word marks a line of your SCRIPT/VS file or macro so that that line may be referred to in a .GO [Goto] control word.

...	**label [line]**

where:

label is a name of up to eight characters that can be used to refer to this line of your SCRIPT file or macro.

line is the active part of this input line. The first nonblank character after the label is treated as if it were the beginning of the line; it may therefore be a control word, but a text line associated with a label may not begin with blanks. If the input line has a label only and no active line, then the next line to be processed is the one following the labeled line.

Default: None

Notes:

1. When the ... control word is encountered, SCRIPT/VS saves the information necessary to enable it to find this line again if a .GO [Goto] control word is encountered. Any valid SCRIPT/VS input line may follow the label, or the label alone may occupy the line.

2. Use of labels and the .GO control word is restricted to one input file or macro. That is, when a new file is imbedded or appended, a new set of labels is in effect while that file is being processed. SCRIPT/VS can only branch to a label within the same input file or macro.

3. Every label in a particular file must be unique. If two identical labels are found in the same file, an error message is issued.

   Multiple labels with the same name are tolerated in macros, but when searching for labels in a macro, only the first occurrence of a label will be found.

4. The .GO function can be relatively inefficient in files. You should use it sparingly in situations where it is the best way to achieve the required results. When going to a label that is later in the input file, it is most efficient when the label is not far from the .GO; when going to a label that is earlier in the file, it is most efficient when the label is near the beginning. Label processing in macros is a much more efficient operation than in files. However, it is more efficient to branch to a label that is earlier in a macro as labels in macros are always searched for from the top of the macro.

5. A space is not required after the control word itself. To set a label called "HERE", either "... HERE" or "...HERE" may be used.

6. The ... for a label must begin in column 1. That is, it must be the first control word on the input line.

Example:

Suppose you had a file called REPORT1 that contained a summary of activity for January, another file, REPORT2, for February, REPORT3 for March, and so forth. Now, if you wanted to create a year-to-date report by imbedding all the report files up to last month's report, you could use this sequence of SCRIPT/VS control words:

```
.se ctr = 1
...loop .im report&ctr
.se ctr = &ctr + 1
.if &ctr lt &SYSMONTH .go loop
```

The first time the .IM [Imbed] is processed, the value of the symbol "&ctr" is 1, so the filename "report&ctr" becomes "report1". The next control word adds one to the value of the symbol; it is now 2. If the month is later than March (month 03), then the value of the counter is less than the month number, and the loop is processed again. This time the filename "report&ctr" becomes "report2". The loop continues until the counter is equal to the current month number.

.AA [ASSOCIATE APF]

The .AA [Associate APF] control word may be used to associate a GML tag with the Application Processing Functions (APFs) that are to be invoked to perform the tag's processing functions and specify the rules for scanning the attributes for the tag. Two APFs may be associated with the tag -- one for the processing when the tag is preceded by the GML tag delimiter, and one for the processing when the tag is preceded by the GML end-tag delimiter. (See the description of the .DC [Define Character] GML and .GS [GML Services] PREFIX control words.) An APF is a SCRIPT/VS executable macro or control word.

The .AA [Associate APF] control word is discussed in "Chapter 14. GML Support in SCRIPT/VS" on page 159.

```
.AA   tag   ┌ OFF        ┐  [(rulkeys)]  ┌ OFF        ┐  [(rulkeys)]
            │ NULL       │               │ NULL       │
            │ .          │               │ .          │
            │ =          │               │ =          │
            └ [.]apfname ┘               └ [.]apfname ┘
```

where:

tag	specifies the GML tag to be associated with an APF.
OFF	indicates that this explicit tag-to-APF association is to be deleted. In this case, an APF can still be determined for a tag by class mapping. You can use .AA to set up an explicit association for a start-tag only, or an end-tag only, and let the other APF be determined by class mapping.
NULL	indicates that this GML tag (or end-tag) is to result in no processing.
.	also indicates that this GML tag (or end-tag) is to result in no processing.
=	indicates that the APF association for this tag (or end-tag) is to remain unchanged.
[.]apfname	indicates that the specified macro is to be executed when the GML tag (or end-tag) is encountered.
rulkeys	are scan rule keywords that set the rules for attribute scanning for this tag to APF association. The recognized rule keywords are ATT, NOATT, VAT, NOVAT, STOP, NOSTOP, MSG, and NOMSG. See ".GS [GML Services]" on page 265 for more information about the meaning of these keywords and the scanning rules they define.

Notes:

1. The first set of parameters apply to the tag. The second set of parameters apply to the end-tag.
2. If no scanning rules are given for this .AA, then the current rules, as most recently set by .GS RULES, are used. Once a .AA association is set up, it has scanning rules with it that remain unchanged, even if new rules are defined with .GS RULES.
3. You can change the APF association with another .AA without changing the scanning rules, or you can change the rules without changing the APF association. If you specify "=" for the APF name, and give a new list of rules, the association remains the same, but the rules are changed. If you specify a new APF name, and don't give a list of rules, the APF association is changed, but the rules remain the same.
4. An empty list of rules, that is, left and right parentheses with nothing in between, causes the current rules to be used. (The current rules are also used if this is the first .AA for this tag, and you gave no rules.)

.AN [AND]

Use .AN [And] control word in conjunction with the .IF [If] control word to process SCRIPT/VS input lines conditionally. The result of the test performed is logically ANDed to the result of the most recently performed .IF [If], .AN [And], or .OR [Or] control word to determine whether the target is to be processed.

The .AN [And] control word is discussed in "The .IF Control Word Family" on page 111.

.AN	**comparand1 test comparand2 target**
	SYSPAGE test { EVEN } target **{ ODD }**
	SYSOUT test { PRINT } target **{ TERM }**

where:

comparand1 is any string to be used as the first comparand. This comparand may be the value of a set symbol.

comparand2 is any string to be used as the second comparand. It too may be the value of a set symbol.

test is a 1- or 2-character code that tells SCRIPT/VS what kind of comparison to make between the two comparands. The following codes are recognized by SCRIPT/VS:

Codes	**Meaning**
eq =	equal
ne ¬=	not equal
gt >	greater than
lt <	less than
ge >=	greater than or equal
le <=	less than or equal

target is any valid SCRIPT/VS input line. It may be a control word or text. If this condition and the most recently performed .IF [If], .OR [Or], or .AN [And] are both true, the target line is processed next, with the first non-blank character after .AN treated as the first position of the subject line. Otherwise, the target line is ignored, and processing continues with the input line that follows the .IF control line.

SYSPAGE is a special .AN keyword that tests whether the page that SCRIPT/VS is currently processing is an even- or odd-numbered page.

See the .IF [If] control word for a description of SYSPAGE, ODD, and EVEN.

SYSOUT is a special .AN keyword that tests whether SCRIPT/VS output is being directed to the offline printer or to the terminal.

See the .IF [If] control word for a description of SYSOUT, TERM, and PRINT.

Notes:

1. For readability, an optional "D" may be added without intervening blank to the .AN control word. This allows the control word to be written as ".AN" or ".AND".

2. The .AN [And] and .OR [Or] control words, in conjunction with .IF [If], .TH [Then], and .EL [Else], allow you to construct complex logic statements.

3. The .AN control word itself does not cause a break; the target control word might, if it is processed.

4. Each of the comparands may be up to 255 characters in length, and the shorter comparand will be extended to the length of the longer with trailing blanks.

5. If substitution is off when the .AN control word is encountered, all valid symbols in the comparands will be resolved before the comparison is made. (Symbols containing imbedded blanks, parentheses, or control word separators must be compared with substitution off so that the test to be performed and the target of the .AN can be identified.)

Examples:

- The following input lines are all equivalent:

```
.if &a./&c eq &b./&d .ty Yes.
.if &a eq &b .if &c eq &d .ty Yes.
.if &a eq &b .and &c eq &d .ty Yes.
```

.AP [APPEND]

Use the .AP [Append] control word to insert an additional SCRIPT/VS file at the end of the file just processed.

The .AP [Append] control word is discussed in "Imbedding and Appending Files" on page 119.

.AP	**{ file-id } [token1 ... token14]**
	{ (filename) }
	{ ('docname') }

where:

file-id is an 8-character SCRIPT/VS name for the file to be appended.

An 8-character name can be associated with an external file or data set with the .DD [Define Data File-id] control word. If no .DD has been executed for the file-id, an external file or data set name is built by SCRIPT/VS from the file-id, using the rules for the current environment, as described in "Naming the Input File" on page 13. If a .DD [Define Data File-id] control word has been issued for the file or data set identified in the .AP control word, the 8-character name is used internally, regardless of whether the 8-character name or the parenthesized file or data set name was specified in the .AP control word.

filename is the real name of the file or data set to be appended, enclosed in parentheses. If no .DD has been executed for the file or data set, SCRIPT/VS will assign an 8-character name to be henceforth associated with that file or data set.

docname is the name of a document to be appended. If the document name contains lowercase or special characters, it must be enclosed in single quotation marks (') and parentheses.

tokens are positional values with a maximum length of 8 characters to be passed to the file to be appended. The first token (word) becomes the value of the symbol &1, the second token becomes the value of the symbol &2, and so forth. The symbol &0 contains the number of tokens that were passed; up to 14 may be specified.

Notes:

1. When the .AP control word is encountered, the current file is closed, and the specified SCRIPT file is processed as a continuation of the SCRIPT/VS input from the previous file. Text or control words following the .AP control word in the current file are not processed.

2. The .AP control word is especially useful for iterative processing of a file. See the example given under the description of the .EF [End of File] control word.

3. The .AP control word closes the current file and starts reading input lines from the appended file. In this sense, the .AP control word marks the end of the file; since it is closed, SCRIPT/VS will not return to it when the appended file is finished. If the .AP control word is not actually at the end of the file, the lines after it are not read.

4. The symbols &1 through &14 are reset whenever a .IM or .AP control word is processed, and the token &0 is reset to the number of non-null tokens. If you want to leave token &1 unset but set token &2, you may use a percent sign (%) in place of token1 (or any other token you want left unset).

5. Error messages, trace output, and identifiers provided by the NUMBER option of the SCRIPT command, all

use the internal 8-byte file-id to describe a file.

Example:

```
.ap abc 10
```

The input file is closed. The contents of the SCRIPT/VS file ABC are processed immediately following the line of the current file which precedes the .AP request. The token 10 is passed to the appended file, so if the file ABC contains a control word of the form:

```
.in &1
```

the result is:

```
.in 10
```

.BC [BALANCE COLUMNS]

Use the .BC [Balance Columns] control word to cancel and restore column balancing for multiple column formatting.

The .BC [Balance Columns] control word is discussed in "Page Sections and Section Breaks" on page 68.

.BC	[<u>ON</u> / OFF]

where:

ON indicates that you want SCRIPT/VS to balance columns. ON is the initial setting as well as the default.

OFF indicates that you do not want SCRIPT/VS to balance columns when a page eject or column definition is encountered.

Initial Setting: ON

Default: ON

Notes:

1. When column balancing is in effect, the number of lines in each column is made as equal as possible before the material on that page is printed.
2. If a blank line that was generated by the .SK [Skip] control word ends up at the top of a column after balancing, it is discarded.
3. When column balancing is off, the number of lines in each column is determined by explicit .CB [Column Begin] control words or by filling all columns, but no attempt is made to equalize the number of lines in all columns.
4. If a column is started explicitly by a .CB [Column Begin] control word, or by a .CC [Conditional Column Begin] or .H0 - .H6 [Head Level 0 - 6] control word that causes an eject to a new column, the new column is ineligible for balancing: text from preceding columns will not be moved into it during column balancing.
5. If a page eject occurs while processing multiple columns, this does not mark any column ineligible for balancing. A column eject that changes the current column from the <u>last</u> column of a page to the <u>first</u> column of the next page is the same as a page eject.

.BF [BEGIN FONT]

Use the .BF [Begin Font] control word to save the current font and begin a new font (a set of characters of one size and style).

The .BF [Begin Font] control word is discussed in "Using Fonts with the IBM 3800 Printing Subsystem" on page 48 and "Defining Internal Fonts" on page 107.

.BF	**[font-id]**

where:

font-id is the name of the font to be started. The name may be either one of those specified with the CHARS option of the SCRIPT command, or a SCRIPT/VS font created with the .DF [Define Font] control word. If no font name is specified, the current font will be used.

Default: The current font.

Notes:

1. When the .BF control word is encountered by the formatter, all subsequent text characters are formatted using the specified font. The specified font remains in effect until another .BF or .PF [Previous Font] control word is encountered with a different font.

2. The system symbol array &$CHAR contains the names of the fonts specified with the CHARS option of the SCRIPT command. &$CHAR(0) contains the number of fonts specified, &$CHAR(1) contains the first font specified (or the default for the logical device, if the CHARS option was omitted), &$CHAR(2) contains the second font, and so on. In addition, the .DF [Define Font] control word allows you to create SCRIPT/VS fonts to provide such formatting functions as underscoring and overstriking.

3. The .BF control word saves the current font before beginning the new font; the .PF control word restores the previous font. Up to 16 fonts can be saved.

4. Refer to "PRINT: Produce Printer Output" on page 27 for details on printing documents formatted for the 3800 under TSO.

Examples:

- This line is in the normal font for this document.

  ```
  .BF GB12
  ```

  has this effect.

- You can specify the font to be started symbolically:

  ```
  .bf &$CHAR(2)
  ```

  starts formatting in whatever font was the second one named on the CHARS option of the SCRIPT command.

- If you have previously defined a SCRIPT/VS font, such as

  ```
  .df hilite us up font GB12
  ```

  then

  ```
  .bf hilite
  ```

  HAS THIS EFFECT.

.BM [BOTTOM MARGIN]

Use the .BM [Bottom Margin] control word to specify the amount of space to be reserved at the bottom of output pages, overriding the initial value established for the device.

The .BM [Bottom Margin] control word is discussed in "Changing the Page Length" on page 57. Figure 8 on page 56 shows the relationship of the .BM [Bottom Margin] to the layout of a SCRIPT/VS output page.

.BM	[v / +v / -v]

where:

v specifies the amount of space to be reserved at the bottom of output pages. v must be large enough to accommodate the footing margin (.FM) and the footing space (.FS), both of which are allocated from the bottom margin area. If +v or -v is specified, the current value of the bottom margin is incremented or decremented. If no value is specified for v, the initial setting is restored. The maximum value for the bottom margin is the page length (.PL) less the top margin (.TM) less space for one line.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

1. The value set by the .BM control word applies on the page after the one on which the .BM control word is encountered, and all subsequent pages until another .BM is encountered.

2. The value given may not be so large that the top margin plus the bottom margin fill the entire page. An error message is issued if you try to set the bottom margin to equal to, or more than, the page length minus the top margin. If you intend to increase the bottom margin so that you can increase the footing margin or the footing space beyond what the old bottom margin would allow, be sure to do it in that order. The rule is, increase the bottom margin before the footing margin or footing space, but decrease the footing margin or footing space before the bottom margin.

3. If you specify .BM 0, the footing margin and the footing space are also made zero automatically.

4. The .BM control word is not allowed in a keep.

5. The size of the bottom margin is not affected by line spacing.

.BR [BREAK]

Use the .BR [Break] control word to ensure that the next input line is not concatenated with the previous line or lines.

The .BR [Break] control word is discussed in "Breaks" on page 38.

.BR	

Notes:

1. The .BR control word is necessary only when SCRIPT/VS is concatenating input lines. It causes the preceding line to be formatted as a short line, if it is shorter than the current column length.
2. Many other control words have the effect of a break. No .BR control word is necessary when one of these is present. See Figure 25 on page 354 for a list of these control words.
3. A leading blank or tab on an input line has the effect of a break.
4. The .BR control word can ensure that some other control words are not effective too early or too late, for example:

   ```
   .br;.tr $ 40
   ```

   may be used to prevent the translation from being effective on the preceding text line, and

   ```
   .tr $ $;.br
   ```

   may be used to make sure the translation does not affect the next line.

Example:

```
Heading:
.br
New paragraph...
```

On SCRIPT/VS output, these lines appear as:

```
Heading:
New paragraph...
```

If the .BR control word were not included, the lines would print as:

```
Heading: New paragraph...
```

.BX [BOX]

The .BX [Box] control word defines and initializes a horizontal rule for SCRIPT/VS output and defines vertical rules for subsequent output lines. With this control word, you can format tables, charts, or text within neatly formatted boxes. The .BX control word is designed to work only in nonmixed pitch situations.

The .BX [Box] control word is discussed in "Drawing Boxes" on page 101.

.BX	[NEW / SET / OFF] [d1 [/] ... dn]
	[CAN / CHAR cname]

where:

d1...dn are the distances from the left margin at which you want vertical rules placed in output text. This format of the control word initializes the box and draws a horizontal line, with vertical descenders at the columns indicated. A slash (/) indicates a discontinuity between columns, with no horizontal rule connecting these columns. Subsequently, entering the .BX control word with no operands causes SCRIPT/VS to print a horizontal line across only the positions not slashed out, as in the initial .BX [Box] specification. This allows two or more separate boxes to be drawn side by side. d1-dn may be specified in space units but the designated values will be converted to M spaces, since the .BX control word is supported for monospaced fonts only.

NEW if no columns are given, or if no box is now going, the NEW function is ignored; otherwise, a new box is started, and the previous box is stacked. This capability allows boxes to be drawn inside boxes.

SET causes SCRIPT/VS to stack the current box, and prepare for a new box. Unlike a "NEW" box, however, the box is not started until the next line of text is processed; consequently, the box will have no initial horizontal rule.

OFF causes SCRIPT/VS to finish drawing the box, by printing a horizontal line with vertical ascenders at the columns in effect. If this box was started as a "NEW" box, the previous box is reinstated when this one is ended. If columns are specified with OFF, and no box is currently in effect, then a box bottom will be drawn according to the column specifications given.

CAN causes the box to be cancelled without a box bottom. If the box being cancelled is a nested box, the next higher box is reinstated.

CHAR Allows you to override the assumed box character set that SCRIPT/VS uses to draw boxes. When you specify CHAR, you must also specify the name of the box character set to be used.

cname is the name of the box character set SCRIPT/VS is to use for drawing all subsequent boxes. The valid names are:

TRM	terminal character set
32T	3270 text characters
TNC	1403 TN or 3211 T11 character set
38C	SCRIPT/VS 3800 character set
GPC	box characters for 3800 GP12 font
32A	3270 APL characters
APL	APL box characters

Default: Repeat previous box definition.

Notes:

1. The .BX control word describes an overlay structure for subsequent text that is processed by SCRIPT/VS. After the .BX d1 d2... line is processed, SCRIPT/VS continues formatting output lines as usual. However, after a line is completely formatted and before it is printed, SCRIPT/VS places vertical lines in the columns indicated by d1, d2, and so on.

   If a data character occupies the same position as a vertical line, the vertical line takes precedence over data characters in the same column.

2. The .BX control word causes a break.

3. The characters used to draw the box depend on the logical device for which formatting is taking place. SCRIPT/VS assumes an appropriate box character set for each logical device, but you can override this by using the CHAR parameter to force any of SCRIPT/VS's box character sets to be used instead.

4. A .BX control word with different columns specified may be used while a box is being drawn. When this happens, vertical ascenders are put in for all the old columns and vertical descenders are used for all the new columns, a horizontal rule is drawn, and the vertical rules are then placed in all subsequent output lines in the new columns designated.

5. The column specification for the .BX control word uses a different rule than is used elsewhere in SCRIPT/VS. In control words like .IN, .TB, .CD, the numbers in the control word represent not columns but displacements. The SCRIPT/VS control word .TB 5 means that a tab character should be expanded to enough blanks to fill up <u>through</u> column 5; the next word starts in column 6. In the .BX control word, .BX 5 means to put vertical rules <u>in</u> column 5. Thus, you can use the same numbers for a .TB control word as for a .BX control word, and the vertical bar will be placed in the column just before the beginning of the word following a tab. Further,

you can define a box that is to be the full column width symbolically with the following control word:

```
.bx 1 &$CL
```

because the number represents the actual column where the vertical rules should be placed.

6. Problems of vertical alignment will occur if the .BX control is used to draw boxes in mixed pitch situations. This will occur if the font being used has characters of different widths, or if monospaced fonts are used which have different widths.

7. The characters to be used for box drawing must be in all fonts which are used within a box.

8. If you define boxes which extend beyond the edge of the column, text in subsequent columns may be displaced.

Examples:

- There is a SCRIPT file called MARYHADA that looks like this:

  ```
  Mary had a little lamb,
  Whose fleece was white as snow,
  And everywhere that Mary went,
  The lamb was sure to go.
  ```

  The following input sequence could be used to center this material in a box that is the same width as the current column length:

  ```
  .bx 1 &$CL
  .ce on
  .im maryhada
  .ce off
  .bx off
  ```

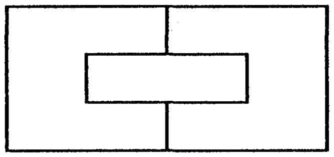
  The result:

  ```
  ┌───────────────────────────────┐
  │    Mary had a little lamb,    │
  │Whose fleece was white as snow,│
  │ And everywhere that Mary went,│
  │   The lamb was sure to go.    │
  └───────────────────────────────┘
  ```

- An example of a nested box:

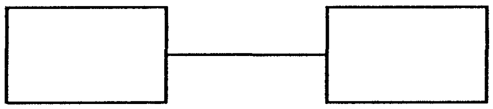
  A nested box was created using the following control word sequence:

  ```
  .bx 10m 20m 30m
  .sp
  .bx new 15m 25m
  .sp
  .bx off
  .sp
  .bx off
  ```



- The following shows the effect of the slash (/) between column specifications:

  ```
  .bx 5m 15m / 25m 35m
  .sp
  .bx new 15m 25m
  .bx can
  .sp
  .bx off
  ```



.CB [COLUMN BEGIN]

The .CB [Column Begin] control word causes subsequent text to start a new column of output.

The .CB [Column Begin] control word is discussed in "Page Sections and Section Breaks" on page 68.

.CB	

Notes:

1. Use the .CB control word when you want to make the following text appear at the top of a new column. If the current column at the time a .CB is encountered is the last column on the page, the column eject is the same as a page eject, since the next column is the first column of the next page. If the current column is not the last column on the page, the new column is made ineligible for column balancing so that the material following the .CB will be at the top of the new column.
2. If a floating or delayed keep is waiting for the start of a new column, then the text that follows the .CB appears after the keep.
3. A column eject may be performed by certain other control words if the conditions warrant it. If this happens, the function is the same as the unconditional column eject that is caused by the COLUMN-BEGIN control word. The other control words that can cause a column eject are:

   .CC [Conditional Column Begin]
   .H0 - .H6 [Head Level 0 - 6]
   .KP [Keep]
   .FN [Footnote]
4. This control word acts as a break. It is not allowed in a keep.

.CC [CONDITIONAL COLUMN BEGIN]

The .CC [Conditional Column Begin] control word causes a column eject if less than a specified amount of space remains in the current column.

The .CC [Conditional Column Begin] control word is discussed in "Page Sections and Section Breaks" on page 68.

.CC	[v]

where:

v is the amount of vertical space that must remain in the current column for processing to continue without a column eject. If v is omitted, and there is text in the current column, a column eject is performed to the top of the next column. This is equivalent to the effect of .CB [Column Begin]. However, if the current column is empty, then no column eject is done. If the previous column is exactly full, and the current column is empty, the .CC control word with no "v" specified does not cause an eject, and the previous column is subject to column balancing, if this is in effect.

Notes:

1. When the .CC control word is encountered, SCRIPT/VS checks to see if there is enough space left in the column. If not, a break followed by a column eject is performed. If the current column is not the last column on the page, the new column is made ineligible for column balancing so that the material following the .CC will be at the top of the new column.

   However, if the specified amount of space or more remains in this column, and no column eject is done, subsequent column balancing may divide the text within the specified vertical space. To ensure that text is kept together, use the .KP [Keep] control word.
2. This control word is not allowed in a keep.

.CD [COLUMN DEFINITION]

Use the .CD [Column Definition] control word to define how many columns of output are to be formatted on each page and where each column is to start.

The .CD [Column Definition] control word is discussed in "Chapter 5. Multicolumn Page Layout" on page 67.

.CD	**n [p1 ... p9]**

where:

n is the number of columns of output to be formatted onto each subsequent output page. It may be any number from 1 to 9.

p1...p9 are positions where the columns are to be placed on the output page, relative to the left edge of the paper (column 1). A position parameter of 0 indicates that a column should be flush with the left edge of the page text area, as defined with the BIND option of the SCRIPT command. Positions may be specified in space units, and may be given in any order.

Initial Setting: 1 0.

Default: None.

Notes:

1. The .CD [Column Definition] control word causes a **section break** when it is processed. This means that all the text up to that point is processed and positioned on the page using the old definition before the new definition becomes active, even if the new definition is the same as the old.

2. The gutter between columns is obtained by defining the column width to a value less than the distance between column starting positions.

3. The positions of the columns do not control how wide the columns are to be; you must set the column width, using the .CL [Column Width] control word, to control this. If the current column width is greater than the distance between columns, the text from a column may extend into the next column. In this case, no gutter is created; the text of the next column is abutted against the text of the previous column. The practice is not recommended as the results may be unpredictable - especially when formatting with multiple fonts. You should take care to define compatible column width and starting positions.

4. If you specify fewer positions than the number of columns, and had previously specified positions on another .CD control word, those values remain in effect for any columns not respecified. Whenever a .CD control word is used, there must be positions for each column available, either on this control word line, or previously given on another .CD. If you specify .CD n without specifying any positions, and no previous column definition has been specified, the arbitrary values 0, 46, 92, 0, 0, 0, 0, 0, 0 are used.

5. If you use several different column formats in a document you can create symbolic names (with the .SE [Set Symbol] control word) or macros (with the .DM [Define Macro] control word) to establish column definitions, column widths, and so on. If you use a single one-column format and a single multiple-column format, you can switch back and forth using the .SC [Single Column Mode] and .MC [Multicolumn Mode] control words.

6. This control word is not allowed in a keep.

.CE [CENTER]

Use the .CE [Center] control word to center output lines between the margins.

The .CE [Center] control word is discussed in "Positioning Lines on the Page" on page 44.

```
.CE    [ 1    ]
       [ n    ]
       [ ON   ]
       [ OFF  ]
       [ line ]
```

where:

n specifies the number of input lines to be centered. If omitted, 1 is assumed. If .CE n is specified when .CE ON is in effect, centering is turned off when n lines have been centered, or when .CE OFF is encountered.

ON specifies that subsequent text lines are to be centered.

OFF terminates centering mode if it was ON, or if n has been specified and has not been exhausted.

line is a line of text to be centered. The line is considered to start with the first nonblank character after the .CE control word.

Default: 1

Notes:

1. The keyword ON or OFF, or the number of input lines to be centered (n), must be the only parameter on the control word line. A string of words that happens to start with one of these is interpreted as a single line to be centered. For example, the control word lines:

   ```
   .ce on top of old smokey
   .ce 555 Bailey Ave.
   ```

   are taken to be of the ".CE line" form, not requests for large numbers of lines to be centered.

2. The line(s) are centered between the current left margin, including any indent and offset values in effect, and the right margin. When centering is in effect, no formatting is done on the line. That is, the line is centered as it stands, and it is not filled from other input lines or justified. If a tab character appears in the line to be centered, the tab is resolved before the line is centered.

3. This control word acts as a break.

4. If the line to be centered is longer than the current column length, the excess words are centered on a separate output line.

5. The .RI [Right Adjust] control word is a variant of .CE. If either of these control words is processed, the other is cancelled.

6. Contrast the .CE control word with .FO CENTER. The latter allows lines to be formatted by concatenating words until the line is nearly full, but then the filled line is centered instead of being justified, as would be the case with .FO ON.

Examples:

- To center one line:

  ```
  .ce OFF THE RECORD
  ```

  When this line of the file is displayed, the characters "OFF THE RECORD" are centered between the margins:

  ```
                 OFF THE RECORD
  ```

- To center several lines:

  ```
  .ce on
  IBM Santa Teresa Laboratory
  Bailey Avenue, San Jose
  95150
  .ce off
  ```

  Each of the 3 lines between ON and OFF is separately centered:

  ```
     IBM Santa Teresa Laboratory
       Bailey Avenue, San Jose
                95150
  ```

.CL [COLUMN WIDTH]

The .CL [Column Width] control word sets the width of each column of SCRIPT/VS output.

Figure 8 on page 56 shows the relationship of the .CL [Column Width] to the layout of a SCRIPT/VS output page.

.CL	[<u>0</u> / h / +h / -h]

<u>where:</u>

h is the width of each column of formatted output. It may not be larger than the logical device width. It may be expressed in horizontal space units.

+h increases the current column width by the specified amount.

-h decreases the column width by the specified amount.

Initial Setting: Same as Line Length.

Default: Restore same width as Line Length.

<u>Notes:</u>

1. The .CL control word should be used in conjunction with the .CD [Column Definition] control word to define the width of each column from the displacements given. If the column width is greater than the space left between columns, the columns may overlay each other. An inter-column gutter is allocated by making the column width about three em-spaces shorter than the distance between columns.

2. The left and right margins of top titles and bottom titles (running titles), and running headings and footings, are governed by the line length, not the column width. (The line length can be changed with the .LL [Line Length] control word.)

3. If the column width has never been set explicitly, it has the same value as the line length. If you set the column width to zero (.CL 0), this makes it the same as though you had never explicitly set the column width. Note that changing the column width by means of the .LL control word means that the column width change will take effect immediately, even though the line length change will not take effect until the following page.

4. This control word causes a break.

.CM [COMMENT]

Use the .CM [Comment] control word to place comments within a SCRIPT file.

.CM	**[comments]**

where:

comments may be anything; this input line is not used in formatting the output. However, since this is a control word, the input line is scanned for control word separators.

Notes:

1. The .CM control word allows comments to be stored in the SCRIPT file for future reference. These comments can be seen when you edit the file, or when you print it using the UNFORMAT option of the SCRIPT command.

   The comments may also be used to store unique identifications that can be useful when attempting to locate a specific region of the file during editing.

2. If you want an entire line to be ignored, and not scanned for control word separators, you can use another form of comment. Any line that begins with ".*" is ignored. ".*" is not considered to be a control word, but .CM is.

3. The .CM control word can be used in conjunction with the .IF control word to enable or disable strings of control words. For an example of how to do this, see the discussion of the .IF control word.

Example:

```
 .cm Remember to change the date.
```

The line above is seen when examining an unformatted listing of the SCRIPT file, and it reminds you to update the date used in the text.

.CO [CONCATENATE MODE]

Use the .CO [Concatenate Mode] control word to cancel or restore concatenation of input lines and truncation at the current column length.

This control word is provided for compatibility with earlier releases of SCRIPT. The same function is provided by the .FO [Format Mode] control word.

.CO	**[<u>ON</u> / OFF]**

where:

ON restores concatenation of input lines. ON is the initial setting, as well as the default value.

OFF cancels concatenation of input lines. If justification is still in effect, .CO OFF results in each line being padded with blanks to the column length.

Initial Setting: ON

Default: ON

Notes:

1. When SCRIPT/VS is concatenating text, output lines are formed by shifting words to or from the next input line. The resulting line is as close to the specified column width as possible without exceeding it or splitting a word; if justification is OFF, output resembles normal typist output. Concatenation is the normal mode of operation for the SCRIPT command.

   When SCRIPT/VS is not concatenating text, there is a one-to-one correspondence between the words on the input and output lines. If SCRIPT/VS is still justifying text, each line that is less than the column length is padded with blank space to fill the column.

2. Concatenation is one component of format mode, as controlled by the .FO [Format Mode] control word. The .CO control word is provided for those occasions when you must be

able to control concatenation separately, but all ordinary formatting combinations are controlled by the .FO control word, and you should use it instead of .CO whenever possible.

3. This control word causes a break.

.CP [CONDITIONAL PAGE EJECT]

The .CP [Conditional Page Eject] control word causes a page eject to occur if less than the specified amount of space remains in the current column.

.CP	**[v]**

where:

v is the amount of vertical space that must remain in the current column for additional lines to be processed without a page eject. If "v" is omitted, a break and a page eject will be done if necessary to get to the top of a page. A break and a page eject will not be done if the current page is empty.

Default: Causes a page eject unless there is no data on the current page.

Notes:

1. The .CP control word can be used to guarantee that enough space (up to the maximum column depth) will exist in one column to accommodate blank space left by .SP [Space] for a figure to be inserted later.

2. To keep formatted text together, use the .KP [Keep] control word.

3. This control word is not allowed in a keep.

Example:

```
.cp 2i
```

If less than two inches of space remain on the current column, a page eject is issued before processing continues. If two inches or more remain, processing continues on the current column.

.CS [CONDITIONAL SECTION]

The .CS [Conditional Section] control word allows you to designate sections of the input file that are to be processed conditionally, or ignored.

The .CS [Conditional Section] control word is discussed in "Conditional Sections" on page 114.

```
.CS    n    { ON       }
            { OFF      }
            { INCLUDE  }
            { IGNORE   }
```

where:

n	specifies the conditional section code number from 1 to 9.
ON	marks the beginning of conditional section n.
OFF	marks the end of conditional section n.
INCLUDE	tells SCRIPT/VS to process all the input lines between the ON and the OFF control words for conditional section n.
IGNORE	tells SCRIPT/VS to bypass every line for conditional section n that falls between .CS n ON and .CS n OFF.

Initial Setting: All sections are included.

Notes:

1. The .CS [Conditional Section] control word allows you to designate specific sections of your input file that may be ignored or included conditionally. You may have up to 9 separate section codes, and specify which section numbers are to be included and

which are to be ignored. Each section code may be used for many sections. The .CS control word is used to designate conditional sections, and also to specify whether they are to be included or ignored. The ON and OFF operands identify the beginning and end of a conditional section; the INCLUDE and IGNORE operands indicate whether or not SCRIPT/VS should process the input lines within the conditional sections.

2. You can use conditional section codes to separate sections of a document that apply to different versions, and specify which version is to be formatted. You may also use this technique to identify confidential sections of a manual that you may sometimes wish to exclude.

3. Since the .CS control word does not cause an automatic break, you may turn conditional sections on and off within a paragraph or even within a sentence without disrupting normal output formatting.

4. By default, all conditional section codes are assumed to be set to INCLUDE unless explicitly set to IGNORE.

5. A conditional section may contain SCRIPT/VS control words as well as text. If the section is ignored, all control words contained in that section will be ignored, except the control word

   ```
   .cs n off
   ```

   which marks the end of the section.

6. Conditional section definitions may be nested to a depth of 9 (that is, a conditional section may contain another conditional section). A nested section is included only if all outer nestings specify INCLUDE. Otherwise, the inner nesting is never noticed, since it is part of an outer section that has been ignored. If a conditional section is nested within another one, the entire section should be enclosed by the outer section.

7. The .CS control word may be used in conjunction with the .RC [Revision Code] control word to mark the conditional sections. The .TE [Terminal Input] control word may be used in interactive environments to specify which sections are to be included while the input file is being processed.

<u>Example:</u>

```
.cs 1 ignore
.cs 2 include
   .
   .
In this version of the system there
can be only
.cs 1 on
256
.cs 1 off
.cs 2 on
1000
.cs 2 off
entries in a MACLIB file.
```

Since only conditional section code 2 is to be included, the generated output line is "In this version of the system there can only be 1000 entries in a MACLIB file."

.CT [CONTINUED TEXT]

The .CT [Continued Text] control word causes the line given to be treated as a continuation of the previous text line. If no line is given, the control word means "continue nothing," which cancels continuation that may be in effect from a continuation character on the previous text line.

.CT	**[line]**

<u>where:</u>

line is the line to be considered a continuation of the previous text input line, even if the previous line did not end with a continuation character. The formatter normally considers that no word may span input lines. A text word may span input lines if the first line ends with a continuation character defined with the CONT parameter of the .DC [Define Character] control word or if the second line is the parameter of the .CT control word. A more complete discussion of continuation may be found under the CONT option of the .DC [Define Character] control word.

Notes:

1. If line is omitted, continuation is ended even if the previous text line ended with a continuation character.
2. If concatenation is not being performed and a .CT [Continued Text] control word immediately follows a .RE [Restore Status] control word, the .CT [Continued Text] is ignored.

.CW [CONTROL WORD SEPARATOR]

The .CW [Control Word Separator] control word allows you to change the symbol used to separate multiple control words on a single line. The initial value for the control word separator symbol is the semicolon (;) character.

This control word is provided for compatibility with earlier releases of SCRIPT. The same function is provided by the .DC [Define Character] CW control word.

.CW	[c]

where:

c specifies the character to be used as the "control word separator" character. Any character may be used. If the character "c" is omitted, no character is assigned as the control word separator, and therefore you cannot have more than one control word on a line.

Initial Setting: Semicolon (;).

Default: Nothing. (No separator character.)

Notes:

1. All control word lines are scanned for control word separators before they are processed, unless they are specified with the control word modifier. The control word modifier allows the line that accompanies a control word to be treated as text, which may therefore contain control word separators as ordinary text characters.

   The control word modifier is a single quotation mark immediately after the period. The control word

   ```
   .ce one; two
   ```

   is scanned before being processed into the two lines ".CE one" and "two". But the line

   ```
   .'ce one; two
   ```

   uses the control word modifier to allow the entire string "one; two" to be centered.

   The .CW [Control Word Separator] control word should always be specified with the control word modifier.A .CW control word line, like all unmodified control word lines, is scanned for control word separators before being processed. If you were trying to make sure that the control word separator was set to semicolon by issuing ".cw ;", just the opposite would happen if the semicolon happened to be the current separator; the line would be separated before being processed into the line ".CW", followed by no more on that line. The control word, when processed, would undefine the semicolon as the separator.

   If you always use the control word modifier with .CW, no separator scan will be done, and the character will be preserved as the parameter on the control word:

   ```
   .'cw ;
   ```

   will correctly ensure that the separator is set to ;.

2. The control word separator is treated as a text character when it appears in a text line, except when immediately followed by a period and a two-character control word id. Thus,

   ```
   Do ;.us on;not ;.us off; stop!
   ```

   will be formatted as

   Do <u>not</u> stop!

   On the other hand,

   ```
   Centering is off; .ce turns it on
   ```

   will be formatted as text, since the control word separator is not immediately followed by a period.

3. When the .CW control word is processed, the initial value for the control word separator (;) is

reset. It may be necessary to change the control word separator character if it is inconvenient to type the initial-value character, or if the initial-value character is used as part of a control word operand, such as part of a symbol specification.

4. If a symbol value begins with the control word separator, the rest of the symbol value is treated as though it occupied the first position of the line.

5. Control word separators are recognized on a .CM [Comment] line, but not on a ".*" line.

6. The following control words must begin in column 1 and may not be placed after a control word separator:

   ```
   .cs n off
   .di off
   .wf off
   .li off
   ...label
   ```

   When SCRIPT/VS is ignoring a conditional section, preparing a delay imbed, writing to a file, reading literal lines, or searching for a label, no control word processing is done. Each input record is checked in column 1 for the presence of the control word that ends the special processing mode.

7. Control words that accept text data (for example, .US or .CE), should not contain the current control word separator as text, unless the control word modifier is used to prevent scanning for the separator.

<u>Examples:</u>

- Simple change:

  ```
  .'cw ,
  .sp 2,.of 5,This section...
  ```

  The above line is equivalent to the lines:

  ```
  .sp 2
  .of 5
  This section...
  ```

- Temporary cancellation to get the separator character into a symbol value:

  ```
  .'cw
  .se 2col = ';.cd 2 0 46;.cl 43;'
  .se 1col = ';.cd 1;.cl 89;'
  .'cw ;
  ```

  In the sequence above, the control word separator is temporarily canceled so that the regular separator (;) can be used as part of the .SE [Set Symbol] control word line. Since the symbols &2col and &1col contain the appropriate control words, they can now be used instead of the actual control words involved. Since the control words are in a symbol that begins with the control word separator, they can be recognized as control words even if the <u>symbol</u> is encountered in the middle of a line. Since the symbols end with control word separators, the effective next line can be concatenated to the symbol name. With the symbols &2col and &1col set as shown, the line:

  ```
  This is a line.&2col.Start 2 cols.
  ```

  Has the same effect as the sequence:

  ```
  This is a line.
  .cd 2 0 46
  .cl 43
  Start 2 cols.
  ```

.DC [DEFINE CHARACTER]

Use the .DC [Define Character] control word to define various special characters that the formatter will recognize as having a special significance.

```
.DC     { ASEP  }  [ c ...  ]
        { CONT  }  [ hh ... ]
        { CW    }  [ OFF    ]
        { PS    }  [        ]
        { STOP  }  [        ]
        { RB    }  [        ]
        { GML   }  [        ]
        { MCS   }  [        ]

        { PUNC  }  [ c ...  ]
        { WORD  }  [ hh ... ]
        { IXI   }  [        ]
        { IXB   }  [        ]
```

where:

c specifies the character (or characters) to be recognized. The character may be any single character.

hh specifies the character (or characters) to be recognized, expressed as a 2-digit hexadecimal code.

If a parameter is given with no following character or hexadecimal code, then the character is restored to its initial setting. For the ASEP, PUNC, WORD, IXI, and IXB parameters, more than one character or hexadecimal code may be specified, separated by blanks. In this case, single characters and 2-digit hexadecimal codes may be intermixed on the same control word line.

OFF causes the character to be undefined. If for example, .DC CW OFF is specified, then there is no control word separator.

ASEP allows the definition of up to four characters which are to be used to separate array elements when an array is substituted in a document using the &name(*) form. All characters to be used to separate array elements must be specified, including blank characters (as 40). The initial and default values for the ASEP characters are the comma (,) and blank (40).

CONT defines a continuation character for text lines. The formatter normally considers that no word may span input lines. Use of the continuation character defined with the CONT parameter allows words to span input lines. When the last character of an input text line is a continuation character, the normal interword space is not added when this line is concatenated to the next, but existing blank characters preceding the line continuation character are retained.

If the formatter control or text which follows the line continuation character causes a break, continuation is cancelled for that line. A null line also cancels continuation for the previous line.

The line continuation character is recognized at the end of a line, whether the line contains text or control words, or a mixture of both. The continuation character may not be used to extend a control word line, but it may extend the text data that is associated with that control word. There is no default line continuation character.

CW specifies the character to be used to separate control words on a single line. The initital value for the control word separator character is the semicolon (;). If the specified control word separator character is hexadecimal 00, the control word separator will be undefined. The effect is the same as if .DC CW OFF were specified.

STOP specifies the characters to be recognized as end of sentence characters. If any of the STOP characters occurs at the end of an input line, or precedes a " or) at the end of a line, and the line is not the last before a break, and justification is off, SCRIPT/VS will insert an extra blank before concatenating with the following input line. If the same character is defined as a continuation character and a stop character, it will be interpreted as a continuation character if it occurs at the end of the line.

PUNC specifies the characters in the current language that are to be recognized as punctuation for spelling checking. Punctuation characters are defined as those characters which, when occurring in a word, will be retained when the word is checked against the dictionary, but when they occur at the end of a word, they will be removed before checking takes place. The default punctuation characters are the hyphen (hexadecimal 60), and the apostrophe (hexadecimal 7D). Punctuation characters given with this option will add to the currently defined default characters.

PS specifies the character to be used as the page number symbol. It may be any character other than blank. The default page number symbol is ampersand (&). Every page number symbol in running titles (.RT), running headings (.RH), and running footings (.RF) is replaced with the current page number every time the running title, heading, or footing is formatted to be placed on a new page.

WORD specifies the delimiters in the current language that are to be used in the recognition of words for spelling verification. The default word delimiters are shown in Figure 37 on page 363.

The characters given with this option will add to the currently defined word delimiter characters. The end of a line will always be recognized as a word delimiter unless the line continuation character is used.

RB defines the character to be used as a required blank. Required blanks are not recognized as interword spaces for formatting, but they are translated to ordinary blanks after formatting is complete. The initial and default required blank is hexadecimal 41. The current required blank character is always available in the system symbol '&$RB'.

GML defines delimiters for GML tags and end-tags. (GML tags and end-tags may be separately defined by the .AA [Associate APF] and .GS [GML Services] PREFIX control words. See "Chapter 14. GML Support in SCRIPT/VS" on page 159 for a discussion of GML tag mapping.)

If only one character is specified, it is taken to be the GML tag delimiter and GML end-tags will not be recognized. If two characters are specified, the second is taken to be the GML end-tag delimiter. If three characters are specified, the second and third characters are taken to be the GML end-tag delimiter.

The GML delimiters may be any characters that are not allowed in a symbol name, except blank, period, or ampersand. That is, the GML delimiter may not be set to the characters blank, period, ampersand, a-z, A-Z, 0-9, or the characters #, @, and $. A .DC GML control word that attempts to set one of these characters as the GML delimiter causes an error message, except .DC GML 40, which is equivalent to .DC GML OFF. The default GML tag and end-tag delimiters are colon (:) and double colon (::), respectively.

IXI (Index Ignore) defines characters that are to be ignored when sorting index entries. Characters to be ignored are completely removed from index entries for purposes of sorting.

IXB (Index Blank) defines characters that are to be treated as blanks when sorting index entries. Characters designated with .DC IXB in effect do not participate in the sort, but they still occupy a character position. This is true even if the blank itself is designated as an IXI (ignore) character.

MCS (Markup Contents Separator) defines the character to be recognized in addition to blank (hexadecimal 40) as the delimiter between the GML tag attributes and the text which follows them.

Default: Restores the initial setting for specified character.

<u>Notes:</u>

1. The settings for initial values are shown in Figure 24 on page 345.

2. The .DC CW control word has the same effect as the .CW [Control Word Separator] control word, except that with .DC, you do not have to use the actual character specified on the control word line; you can specify it as a 2-digit hexadecimal code. This capability is useful to prevent misinterpretation of the control word separator character in cases where it is already set to the value specified on the control word. See the discussion of the .CW [Control Word Separator] control word for additional information.

3. The .DC PS control word has the same effect as the .PS [Page Number Symbol] control word. See the discussion of the .PS [Page Number Symbol] control word for additional information.

<u>Examples:</u>

- Continuation Character

  In the following examples, the plus sign is used as the continuation character. The continuation character may not occur in the middle of a control word line. For example,

  ```
  this is
  .up part+
  ially uppercase
  ```

  results in:

  ```
  this is PARTially uppercase
  ```

  The continuation character will, however, allow the user to create one "logical" line from a number of input lines. For example:

  ```
  .ce 1
  this is a sin+
  .up gle line
  ```

  will result in the line:

  ```
        this is a sinGLE LINE
  ```

- PUNC and WORD

  Note that there is only one delimiter table to hold both punctuation and word delimiter characters. The latest specification for a character will be that in effect. For example:

  ```
  .dc word +
  ```

  will cause the + character to be recognized as a word delimiter character.

  ```
  .dc punc +
  ```

  will cause + to be recognized now as a punctuation character.

- IXI

  By default, "William Steinburg" will be placed before "Williamsburg" in an index. If you specify

  ```
  .dc ixi 40
  ```

  then the former entry will be sorted as if the entry were "Williamsteinburg", and will be placed after "Williamsburg".

- IXB

  By default, "Mother Superior" will be placed before "mother-in-law" in an index. If you specify

  ```
  .dc ixb -
  ```

  the latter entry will be sorted as if the entry were "mother in law", and placed before "Mother Superior".

.DD [DEFINE DATA FILE-ID]

The .AP [Append] and .IM [Imbed] control words require a file-id for the file to be imbedded or appended. This file-id is an internal SCRIPT/VS name for a file or data set in the host environment in which SCRIPT/VS is executing. The .DD [Define Data File-id] control word allows you to associate a 1- to 8-character internal SCRIPT/VS file-id with a real file or data set identifier. If no .DD has been issued for a file-id, a valid identifier is constructed from the file-id, based on assumptions and rules established for each operating environment.

.DD	**file-id**	[**<u>LIB</u>**
		DD
		DSN
		TERM]
			[**filename**
			(filename)
			('docname')]
				[**CATALOG**
				DATA(datatype)
				VERSION(number)
				PROC(processor)
				PARM('parameters')]

<u>where:</u>

file-id is an 8-character SCRIPT/VS name for the file being defined. An error will result if the name corresponds to a file or data set which is already in use as an imbedded file. If the file-id corresponds to a file or data set that was previously read and terminated by a .EF [End of File] control word, that original file is closed before the redefinition is made.

LIB is the default, and indicates that the file to be referred to exists in the library of the environment in which SCRIPT/VS is operating.

In CMS, the filename given is a normal CMS filename, followed optionally by a filetype and a filemode.

In TSO, the LIB option indicates that the dsname refers to a PDS member whose data set name is specified by the SEARCH option of the SCRIPT command.

In the batch environment, LIB indicates that the filename is a Document Library Facility file, whose filename may be followed optionally by a library number and password.

DD specifies that filename refers to a DD name. This option is applicable only in the TSO and batch environments. Use of the DD option implies that the user has supplied a JCL DD card with a ddname of "filename", or preallocated the data set by use of the TSO ALLOCATE command.

DSN specifies that filename refers to a fully or partially qualified data set name. This option is applicable only in the TSO and VS2 environments.

TERM specifies that, for file-ids DSMTERMI and DSMTERMO <u>only</u>, the input or output is to be restored to the terminal. This option is useful when the terminal input or output has been changed with a previous .DD control word for either of these file-ids.

filename specifies the actual name of the file to be given the specified file-id. If the filename contains blanks or special characters it must be enclosed in parenthesis.

docname is the name of the document to be given the specified file-id. If the document name contains lowercase characters, it must be enclosed in quotation marks (') and parentheses.

CATALOG specifies that the data set to which the file-id refers is to be cataloged when it is is closed. This option is valid only for SCRIPT/VS utility files (see Usage Notes) when used with the DSN option in TSO to create a new data set. In all other cases it is ignored. CATALOG is especially useful when creating output files with the .WF control word. It is possible to create many different .WF files by specifying a different data set name with .DD. Normally, these data sets would be deleted when closed.

DATA	(batch only) specifies the file's datatype. Refer to the Document Library Facility Guide for details.
VERSION	(batch only) specifies the file's version number. Refer to the Document Library Facility Guide for details.
PROC	(batch only) specifies a processor for the file. Refer to the Document Library Facility Guide for details.
PARM	(batch only) specifies parameters for the processor named by the PROC option. Refer to the Document Library Facility Guide for details.

Default: LIB

Notes:

1. If the .DD control word is used with a parameter that is not allowed in the formatter's operating environment, a message will be issued and the control word will be ignored.

2. The file-id PROFILE is used for the profile specified with the PROFILE option of the SCRIPT command.

3. SCRIPT/VS has a number of file-ids that are used by the system. Some of these may be the subject of the .DD control word. These file-ids are:

```
DSMTERMI - terminal input
DSMTERMO - terminal output
DSMUTCTF - STAIRS/VS CTF output
DSMUTDIM - .DI file
DSMUTMSG - message file
DSMUTTOC - Table of contents file
DSMUTWTF - .WF file
```

   For example, if the file-id DSMTERMI is associated with a disk file, then whenever a .RV or .RD is processed the data will be read from the specified file. This capability is of particular use in the batch environment.

   Whenever any of these file-ids is the subject of a .DD control word, the host system file name associated with the existing definition of that file-id is closed, and, in TSO, it is also deallocated.

Examples:

- To give a file-id of "filel" to a Document Library Facility file named "title", with a password of "p2301" which exists in library "13425", the .DD control word statement would be:

```
.dd filel lib 13425 title/p2301
```

  In this example, the option LIB could have been omitted, as it is the default.

- To give a file-id of "alpha" to member "mem3" of a partitioned data set named "userid.doc.text", the .DD control word statement would be:

```
.dd alpha dsn doc(mem3)
```

- In CMS, if there are two SCRIPT files called "names", one on your A-disk, and the other on your C-disk, the control word .IM NAMES would ordinarily imbed the one on the A-disk, following CMS search order. The file on the C-disk would be imbedded if the following .DD were in effect:

```
.dd names names script c
```

  Note that in this example the keyword LIB was omitted because it is the default.

.DF [DEFINE FONT]

Use the .DF [Define Font] control word to define the identifier of a font which is to be invoked using the .BF [Begin Font] control word. This allows such internal formatting functions as underscoring and capitalizing to be managed with the .BF [Begin Font] and .PF [Previous Font] control words, and provides the ability to selectively overstrike text (on impact printers) and temporarily stop typing (on typewriter terminals) to change elements. It also allows you to alter the names and characteristics of the fonts named in the CHARS option of the SCRIPT command.

The .DF [Define Font] control word is discussed in "Defining Internal Fonts" on page 107.

```
.DF     font-id  { US           }
                 { UP           }
                 { UC           }
                 { STOP         }
                 { OS RPT n     }
                 { OS CHAR c    }
                 { BOX cname    }
                 { FONT [name]  }
```

where:

font-id specifies the identifier of the font being defined by the .DF control. After the definition, this identifier may be used on a .BF [Begin Font] control word.

US specifies that this font causes underscoring.

UP specifies that this font causes text to be folded to uppercase.

UC specifies that this font causes both folding to uppercase and underscoring.

STOP specifies that when this font begins, the interactive typewriter terminal will stop and wait for the user to signal ATTENTION. The STOP parameter is used for changing fonts at a typewriter terminal where each "font" is actually a different typing element. When the typewriter stops, you can change the element. The STOP parameter has meaning only in the CMS environment; it is ignored in other environments, or when the output destination is not a typewriter terminal.

OS specifies that this font is to be formed by overstriking. The OS option is ignored for logical devices other than 2741 and 1403, unless overstriking with the underscore character is specified.

- **RPT** specifies that text lines are to be overstruck such that each character is overstruck with itself n times. A maximum of five overstrikes are allowed.

  - **n** specifies the number of times that each character is to be overstruck with itself.

- **CHAR** specifies that overstriking is to take place using the character specified. If neither the RPT nor the CHAR option is given, the underscore character will be used.

  - **c** specifies the character that is to be used as the overstriking character.

BOX specifies a box character set to be used in constructing box corners and intersections with the .BX [Box] control word.

- **cname** specifies the name of the box character set to be used. See the CHAR parameter of the .BX [Box] control word for a list of valid box character set names.

FONT specifies the external font to be used. If FONT is not specified, the external characteristics of the current font are used.

- **name** specifies the name of an external font. The name must be one of those given with the CHARS option of the SCRIPT command. If not specified, the current font is used.

Notes:

1. As many parameters as necesssary may be specified with .DF [Define

Font].

2. If the font being defined already exists, either because of a previous .DF [Define Font] control word, or because it was specified with the CHARS option of the SCRIPT command, the previous definition is completely replaced with the new one.

3. When using the STOP parameter under CMS, issue the command

```
cp term attn off
```

to suppress CP's attention acknowledgment.

Examples:

- To define a new font for 1403 output which causes capitalization and overstriking, specify

```
.df bold up os rpt 3
```

- To define a new font for 3800 output which causes underscoring in a bold font, specify

```
.df hilite us font GB12
```

.DH [DEFINE HEAD LEVEL]

Use the .DH [Define Head Level] control word to override the default characteristics of the head levels that are generated with the .H0 - .H6 [Head Level 0 - 6] control words.

The .DH [Define Head Level] control word is discussed in "Chapter 6. Head Levels and Table of Contents" on page 73.

.DH	n [options]

where:

n is the number of the head level to be defined. It may be a number from 0 to 6.

options are keywords that indicate how to change the definition of head level n. If no options are given, the default characteristics of the head level are restored. The options recognized are:

BR do a break after the head.

NBR no break.

DOT specifies that in the table of contents, the text of the heading will be separated from the page number by a "dot-leader," and the page number will be right-aligned. (This is the default for all head levels that have a table of contents entry.)

NODOT specifies that the table of contents entries for this head level will not have a "dot-leader" separating the text of the heading from the page number. In this case, the page number is not right-aligned, but it is separated from the text by two character spaces.

FONT fontname specifies the name of the font to be used for the heading, or the word "OFF". If OFF is specified, the previous specification of FONT will be set off . If the fontname and OFF are omitted and the FONT option is the last one specified, no syntax error will result. This is useful when using the &$CHAR system symbols as the font names, since these symbols have null values when there is no corresponding font.

NUM specifies that this heading is to be numbered with a decimal number that reflects the level of the heading. The first level one heading is numbered 1.0, the first level two heading is numbered 1.1, the first level three heading is numbered 1.1.1, and so on. You can set the counters used for head numbering with the HCTR option of the .GS [GML Services] control word, but this is not necessary if you want the numbering to start with 1.0 and increase sequentially from there.

| You can not number level 0 headings.

NONUM specifies that this heading is not to be numbered. (This is the default for all head levels.)

OJ	outjustifies the head level (this means right adjust it if it falls on an odd-numbered page).
NOJ	do not outjustify the head level.
PA	do a page eject before the head level if necessary (if not already at the top of a page).
NPA	no page eject.
SECT	cause a section break before and after the head level. The SECT option means that the head level itself will be formatted in a single column, the full width of the page, regardless of the current column definition.
NOSECT	no section break is required.
SKBF v	v is the amount of space to be skipped before the head level.
SPAF v	v is the amount of space to be skipped after the head level. This space is conditional.
TC	table of contents entry wanted.
NTC	No table of contents entry.
TCIN h	h is the amount the table of contents entry associated with the head level is to be indented.
TFONT fontname	specifies the name of the font to be used in the table of contents for this head level entry, or the word "OFF". It works similarly to the FONT option.
TO	table of contents entry only; the heading will not be printed in the text and neither will functions such as skip, space, and so on.
NTO	no "TO"; the heading will be printed in the text.
TS	space before table of contents entry.
NTS	no "TS"; the table of contents entry will not be preceded by a space.
UP	put the head level in uppercase.
NUP	don't put it in uppercase.
US	underscore the head level.
NUS	don't underscore it.
Default:	If no options are specified, the initial setting for the head level is restored.

<u>Notes:</u>

1. The .DH [Define Head Level] control word allows a maximum of 14 options on the line. If you wish to change more head level variables than can be done with 14 options, you must do it with more than one .DH control word. Each time .DH is processed, only those variables specified are changed. All other variables remain the same.

2. If a head level control word is processed that causes an entry in the table of contents, the table of contents entry is saved with the specifications that are in effect at the time that head level is processed. If you change the definition of that head level later, the new definition only affects later occurrences of that head level control word.

3. For a list of the default characteristics associated with the heading levels 0 through 6, see the discussion of .H0 - .H6, and Figure 9 on page 74.

4. The use of .EZ ON will cause a new set of heading definitions to be used. Therefore, each time .EZ ON is used, you must respecify the head definitions that you want to use.

5. If you want to add a function to a head level control word that is beyond the scope of .DH [Define Head Level], you can either write an entire macro to replace the head-level control word in question, or you can provide a .DH macro to create head-level macros like those in Release 1, and then add to those macros using the .DM [Define Macro] control word. The sample file DSMSTDH contains an example of how such a .DH macro could be written.

.DI [DELAY IMBED]

Use the .DI [Delay Imbed] control word to defer the inclusion of a portion of a SCRIPT file until the next page eject occurs.

```
.DI   [ 1    ]
      [ n    ]
      [ ON   ]
      [ OFF  ]
      [ line ]
```

where:

n	specifies the number of lines to be delayed. If omitted, 1 is assumed.
ON	starts an open-ended delayed imbed. Subsequent lines, until a .DI OFF is encountered, are included in the delay imbed file.
OFF	ends a delayed imbed, whether it was started with .DI ON or with a specified n that has not been exhausted.
line	is an input line that is to be delayed.

Default: 1

Notes:

1. The specified lines of the current file are saved in a temporary file called DSMUTDIM. When the top of the next output page is reached, this temporary file is imbedded and processed by SCRIPT/VS. After the inclusion of the saved lines, normal processing resumes.

   However, any text that was formatted prior to the page eject (such as a widow zone) will not be reformatted. It should be noted that DCF may have to format up to three lines before it can determine that a page eject must be performed.

2. This control word does not cause a break. However, a control word that causes a break may be included as the last line of the delayed imbed to ensure that the text of the delayed imbed is not formatted with the text that follows the imbed. The results are not readily predictable if a control word that causes a break is not included.

3. An automatic page eject is not performed at the end of the inclusion. If you want SCRIPT/VS to resume normal processing on a new page, you should end the delayed section with a .PA control word.

4. The .DI OFF control word must begin in column 1, not after a control word separator. When SCRIPT/VS is processing a delay imbed it is not processing input lines except to look for .DI OFF on a line by itself.

5. No .DI control word is put into the delay imbed file.

Examples:

- To delay the inclusion of one line:

  ```
  .di .pa
  ```

  The single line ".pa" is written into the delay imbed file. At the end of the current page, a blank page, except for top and bottom titles, is generated. Output resumes on the page after the blank page.

- To include a figure at the top of the next page:

  ```
  .di 3
  .sp .5i
  .im figure5
  .sp 5
  ```

  The current page is processed as if the .DI and the three following lines had not existed. At the top of the next page, the three lines are processed. This results in spacing a half-inch, imbedding the file named FIGURE5, followed by spacing five lines.

.DL [DICTIONARY LIST]

Use the .DL [Dictionary List] control word to specify the dictionaries to be used for hyphenation and spelling verification. The dictionaries specified must be all of the same language.

The .DL [Dictionary List] control word is discussed in "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.

.DL	**name ... name**

where:

name is the name of a dictionary. Dictionary names may be up to four characters in length. The dictionaries listed may be either one of the IBM-supplied base dictionaries or any user-created dictionaries.

Notes:

1. The addenda dictionary used will be that of the first dictionary named. The base dictionary for the current language will follow the addenda dictionary in the search sequence, followed by any user-created dictionaries. The names of the IBM-supplied dictionaries are

DUTH	- Dutch
EAM	- English (American)
EUK	- English (United Kingdom)
ECAN	- English (Canadian)
FNAT	- French
FCAN	- French (Canadian)
GERM	- German
ITAL	- Italian
SPAN	- Spanish

2. If the .DL [Dictionary List] control word is not used, the default language specified by your installation will be used for spelling checking and hyphenation.

3. If the first dictionary named is one of the IBM-supplied base dictionaries, any user-created dictionaries currently in use will no longer be used, and the stem processing routines for the new language will be loaded.

4. For a description of the function and use of the formatter's spelling checking and hyphenation capabilities, see the discussion in "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.

.DM [DEFINE MACRO]

Use the .DM [Define Macro] control word to establish macro definitions for sequences of SCRIPT/VS control words or text lines. SCRIPT/VS macros are invoked by preceding them with periods, as SCRIPT/VS control words. No macro substitution is performed unless the .MS [Macro Substitution] control word has been processed to turn macro substitution ON.

The .DM [Define Macro] control word is discussed in "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147.

```
.DM     name    [ /line1/.../linen[/] ]
                [ x                   ]
                [ LIB                 ]
                [ OFF                 ]

        name[[n]]  [ /line[/] ]
                   [ x        ]
                   [ OFF      ]
```

where:

name is the symbolic name you want to assign to the macro, so that you can invoke it with the control line:

```
.name
```

It may contain a maximum of 10 nonblank characters which may be upper- and lowercase alphabetic, numeric, and the characters @, #, and $.

name(n) indicates that the line that follows is to be stored as part of the macro definition in line n. By this means, multiple line macros may be defined. The values for n need not be sequential when the macro is defined, but if the same value for n is given on two uses of the .DM control word, only the latest value for the line will be stored. They are executed in numerical sequence. When a line number is given with the name, only one line of the macro may be given. Each line of the macro is defined with a separate .DM control word. (n) must follow the macro name <u>without</u> intervening blanks, and must be a positive integer or zero. If n is omitted, that is, "name()" is specified, macro elements are assigned with line number increments of 10. Macro element zero has the same significance as array element zero and can be assigned a number (using the .DM control word) which will control the start of automatic line number assignment. If you set macro element zero to other than a valid number, the value that you set will be lost. It will never be executed with the rest of the macro.

is any character used to delimit the individual lines in the macro. The final delimiter may be omitted.

line is any SCRIPT/VS control word line or line of data that you want to include in the macro definition. It may contain symbolic names, or any of the special macro variables &*, or &*1 through &*n (see Notes). If line is omitted, the macro (or macro line if n is given) is stored as a null macro or macro line.

x indicates that you want the current value of a macro or macro line assigned to the symbol &x. x may be any single alphameric character. (If you give two or more characters, SCRIPT/VS treats the first as a delimiter and the others as a line to be inserted in the macro definition.)

LIB causes the macro to be defined by retrieving its value from a library. The name of this library may be defined using the LIB option of the SCRIPT command. If LIB is used to define a macro, the definition retrieved from the library completely replaces the current definition (if one exists). If LIB is specified, but no definition with the macro name given exists on the library, the macro will be undefined. Since macro names are in uppercase only, names are folded to uppercase before the library is accessed. The LIB parameter sets up an entirely new macro definition; no line number may be given with the macro name. The LIB option may be used independently of the .LY [Library] control word.

OFF deletes a macro definition or a line from a definition.

<u>Notes:</u>

1. The following symbols have special meanings within macros:

   <u>&*:</u> is the line passed to the macro when it is invoked. Thus, if the macro defined with:

   ```
   .dm typit() /.ty ***
   .dm typit() /.ty &*
   .dm typit() /.ty ***
   ```

   is invoked with the line

   ```
   .typit Hello!
   ```

   then the symbol &* has the value "Hello!". The processing of this macro results in the lines:

   ```
   ***
   Hello!
   ***
   ```

   being displayed at your terminal.

   <u>&*0:</u> Contains the number of tokens passed when the macro is called. Using the above example, the value of &*0 is 1.

   <u>&*1 - &*n:</u> Are the tokens passed to the macro when it is called. You can pass as many tokens to a macro as will fit on the input line. If the .typit macro is invoked

   ```
   .typit Processing section 5...
   ```

   then &*1 has a value of "Processing", &*2 has a value of "section", and &*3 has a value of "5...". The value of &*0 is 3.

2. Macro calls are treated as invalid control words if you do not use the .MS [Macro Substitution] control word:

   .ms on

3. Symbol names that are used in a macro definition are substituted at the time the .DM control line is processed, if substitution is on. If you want to use variable symbols in a macro to be substituted at execution time, you must use the control word

   .su off

   before defining the macro with the .DM control word.

4. Values for the symbols &*1 through &*n are established whenever a macro is invoked. These values are local to the current level of macro invocation.

5. A macro name may be the same as the two-letter name of a control word. Such a macro effectively redefines the control word by getting control whenever the control word is encountered.

6. Macros may be invoked recursively. In order to avoid looping situations for recursive invocation SCRIPT/VS keeps invocation counts for macros. Any given macro may not be opened more than 99 times, and no more than 255 macros of any name may be open at the same time. If either of these situations occurs, a severe error message is issued and processing is terminated.

7. If macros are defined with multiple macro lines on a single line of input, the macro will be stored as if it had been entered on separate lines with an increment of 10, and the new definition will completely replace any existing definition with the same name. However, subsequent macro lines defined using sequence numbers will behave as if all lines had been added in this way.

8. Macros defined using sequence numbers may be defined using sequence numbers in any order. However, the macro will be executed as if the lines had been entered in sequence. Macro lines may be redefined at any time within a document, or lines added or inserted into an already existing macro definition. This addition or redefinition of lines will take place based on the sequence number specified.

9. If an entire macro is assigned to a symbol "x" it will be stored in the form:

   #line1#line2#line3#linen#

   where # represents a separator character of hexadecimal FF. If only a single line of a macro is assigned to a symbol "x", it will be stored in the form #line#. If you want to print this symbol, the .TR control word must be used to convert this character to one that is available on the printer being used, if this is required. When using the symbol assignment capability, remember that the maximum length for a symbol is 252 characters.

   The symbol assignment capability can be used to test the existence of a macro or a macro line, as follows:

   - if the macro (or macro line) does not exist, the symbol "&x" is assigned a null value (&L'&x=0).
   - if the macro (or macro line) does exist, but has a null value, the symbol "&x" is assigned a value of hexadecimal FFFF, which is two consecutive separator characters (&L'&x=2).
   - else, the symbol "&x" will have the value of the complete definition of the macro (or macro line) as described before (&L'&x>2).

10. Use of the LIB option of the .DM control word allows a macro definition to be explicitly retrieved from the library. Use of the .LY control word allows macro definitions to be retrieved from the library when a macro is <u>used</u> in a document where a definition for it does not currently exist.

.DS [DOUBLE SPACE MODE]

Use the .DS [Double Space Mode] control word when you want your output to be double-spaced.

.DS	

Notes:

1. This control word does not cause a break.
2. The .DS control word doubles the line spacing set by the .SL control word. When double-spacing is in effect, each space or skip caused by a .SP or .SK control word is doubled (thus, .SP 2 yields four spaces). However, if the .SP or .SK control word indicates "absolute" spaces, the space count is not doubled.
3. Additional space is placed above each output line, and is discarded if the line falls at the top of a column. This may result in columns being set short by the amount of discarded space.

Example:

```
.DS
```

Blank lines are inserted between out-

put lines below this point in the

file, as shown in these few lines.

.DU [DICTIONARY UPDATE]

Use the .DU [Dictionary Update] control word to add or delete words from an addenda dictionary. The changes to the dictionary that are specified using this control word are in effect only during the formatting of the current document.

The .DU [Dictionary Update] control word is discussed in "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.

```
.DU    { ADD }  [NAME name]   word ... word
       { DEL }
```

where:

ADD specifies that the word or words given with the control word are to be added to the addenda dictionary.

DEL specifies that the words given with the control word are to be deleted from the addenda dictionary.

NAME specifies the name of the dictionary with which the addenda dictionary is asociated. If NAME is omitted, the addenda dictionary will be associated with the first dictionary in the current sequence.

word is a string of blank delimited words.

Notes:

1. A .DU control word which requests that a word be added to, or deleted from, the addenda dictionary where that word is already in the addenda dictionary (for ADD), or not in the addenda dictionary (for DEL), will not cause an error message. The first ADD for a word will put the word in the dictionary, and all subsequent ADDs will be ignored. The first DEL for a word will delete the word from the dictionary, and all subsequent DELs will be ignored. You should be careful to avoid multiple ADD or DEL situations where a word may get added and perhaps also deleted in an imbedded file.
2. For a description of the function and use of the formatter's spelling checking and hyphenation capabilities, see the discussion in "Chapter 1. An Introduction to SCRIPT/VS" on page 1.
3. Words added to the dictionary using the .DU control word may include hyphens. In this case, the hyphens indicate potential hyphenation

points for the word. Whenever hyphenation is in effect (specified by the .HY control word) these hyphenation points will be used unless the .HW control word has been used for the specific occurrence of the word, or use of the addenda dictionary has been suppressed with the NOADD option of the .HY control word.

4. Words that contain hyphens, such as lighter-than-air, should be supplied to the .DU control word with double hyphens at these hyphen points, as described for the .HW control word.

5. Words that contain word delimiters will be added to the addenda dictionary with the delimiters intact. You can redefine these delimiters with the .DC control word.

6. Stem processing is used for verification against both the main and the addenda dictionaries when requested using the .SV control word.

7. Words may be added to the addenda dictionary even when spelling verification is off, or is in effect against the main dictionary only.

8. Only 804 entries may be made to the addenda dictionary.

9. For encoding hyphenation points specified with .DU ADD, the following rules are observed:

   - The word is divided into a maximum of four groups of three letters starting with the first vowel encountered after the first letter of the word (including the letter y).
   - Each group can contain only one hyphenation point. If more than one point is specified, the first one specified will be used.
   - The result is that a word can contain a maximum of four hyphenation points.

.EC [EXECUTE CONTROL]

The .EC [Execute Control] control word is used to cause SCRIPT/VS to execute the given line as a control word line, even if there is a macro defined with the same name, and macro substitution is ON.

.EC	**control word line**

where:

control word line is a SCRIPT/VS control word line.

Notes:

1. Use the .EC control word whenever you want to cause SCRIPT/VS to execute a control word even when a macro is defined with the same name. The .EC control word is useful within macros that have the same name as control words. Often, a macro that "redefines" a control word uses the control word function in addition to whatever other function it performs. In these cases, if the .EC function were not used, the same macro would be repeatedly invoked in a loop until SCRIPT/VS terminated it with a severe error message. Of course, macro substitution could be turned OFF, but that would prevent any other macro from being invoked until macro substitution was turned ON again.

2. The control word modifier provides an implied .EC function. It also prevents the control word separator scan on that control word line. The control word modifier may be used with any control word; it consists of a single quotation mark (') between the period and the name of the control word. (.'ce center this line).

Examples:

- To define a macro called .IM to replace the .IM control word without using the .EC control word would require the following macro definition:

```
.su off
.dm im(1) /.ty &*/
.dm im(2) /.ms off/
.dm im(3) /.im &*/
.dm im(4) /.ms on/
.su on
```

  In this example, macro substitution needs to be turned off to avoid an infinite macro substitution loop. Unfortunately, this has the effect of turning off macro substitution for the imbedded file, and all files that it imbeds. In this situation, the .EC control word should be used:

```
.dm im(1) /.ty &*/
.*.pa
.*.*.*:tnlsync.
.dm im(2) /.ec .im &*/
```

The control word modifier may be used in the same way:

```
.dm im(1) /.ty &*/
.dm im(2) /.'im &*/
```

The difference between the .EC form and the control word modifier form is that the .EC line is scanned for control word separators, while the control word modifier line is not. In this example, there is no difference between the two; the original .IM macro line will already have been scanned for separators.

- The .EC control word will issue an error message if the subject control word line is not a valid control word line. To be a valid control word, it must start with a period, followed by two characters and a blank. (A line without a period in column 1 is usually treated as text, but, as the subject of .EC, it is treated as an invalid control word.)
- The .EC control word will issue an error message if the control word line given is "valid", but refers to a nonexistent control word, even if a macro exists with the control word name given.

.EF [END OF FILE]

The .EF [End of File] control word simulates the end of the current file. When used in a profile file, the contents of the profile preceding the .EF [End of File] control word will be processed before the main document, and the remainder of the profile will be processed after the main document.

The .EF [End of File] control word is discussed in "Terminating the Formatting of a File" on page 124.

.EF	[CLOSE]

where:

CLOSE tells SCRIPT/VS not to hold your place in the current file, but to close it, so that the next time the file is imbedded, SCRIPT/VS begins processing at the top of the file, not at the line following the .EF control word.

Notes:

1. The .EF [End of File] control word causes an end of file condition to be simulated on the current input file. If the current input file is not an imbedded file (see the discussion of the .IM [Imbed] control word), all processing is terminated. If the current input file has been imbedded, the .EF control word causes input processing to continue in the outer file. In this latter case, SCRIPT/VS remembers the position of the .EF control word; if the file is imbedded again, then SCRIPT/VS begins reading at the line following the .EF control word instead of the beginning of the file, unless the CLOSE operand is used. In the case of the profile file, this will happen when all other files have been processed.

.EL [ELSE]

The .EL [Else] control word can be used in conjunction with the .IF [If] control word to process SCRIPT/VS input lines conditionally. The target line is processed only if the most recently performed .IF [If], .AN [And], or .OR [Or] control word resulted in a false condition.

The .EL [Else] control word is discussed in "The .IF Control Word Family" on page 111.

.EL	target

where:

target is any valid SCRIPT/VS input line. It may be a control word or text. If the most recently performed .IF [If] was false, the target line is processed next, with the first nonblank character after the .EL treated as the first position of the line. If the condition was true, the target line is ignored, and processing continues with the input line that follows the .EL control word line.

Notes:

1. For readability, an optional "SE" may be added to the .EL control word without intervening blank. This allows the control word to be written as ".EL" or ".ELSE".

2. The .TH [Then] and .EL [Else] control words, in conjunction with .IF [If], .AN [And], and .OR [Or], allow you to construct complex logic statements.

3. The .TH and .EL control words themselves do not cause a break or change the true/false condition; a target control word might, if it is processed. For example, the input lines

   ```
   .if &a eq &b
   .else .if &c eq &d
   .then .ty Yes.
   ```

   are equivalent to the line

   ```
   .if &a eq &b .or &c eq &d .ty Yes.
   ```

4. Multiple .TH [Then] and .EL [Else] control words may follow an .IF [If] control word; only the .TH [Then] control words will be executed if the .IF [If] resulted in a true comparison, and only the .EL [Else] control words will be executed if the .IF [If] resulted in a false comparison.

5. If there is no most recently performed comparison, the target line will not be processed.

Examples:

- The following input lines

  ```
  .if &a ne &b .ty Yes,
  .if &a ne &b .ty still.
  ```

  are equivalent to the following lines:

  ```
  .if &a eq &b
  .else .ty Yes,
  .el .ty still.
  ```

.EM [EXECUTE MACRO]

The .EM [Execute Macro] control word is used to cause SCRIPT/VS to execute the given line as a macro line even if there is a control word with the name given, and macro substitution is OFF.

.EM	**macro line**

where:

macro line is an input line that invokes a SCRIPT/VS macro.

Notes:

1. Use the .EM control word whenever you want to cause SCRIPT/VS to execute a macro when macro substitution is OFF. The .EM control word is useful when a control word must be replaced with a macro of the same name and macro substitution cannot be turned on.

2. If the .EM control word specifies a macro for which no valid macro definition exists, it is treated as an invalid control word, even if there is a control word of that name.

3. The control word modifier provides an implied .EC [Execute Control] function, and also prevents a control word line from being scanned for control word separators. If you want to prevent a macro line from being scanned for separators, you can use the control word modifier for the .EM control word:

```
.'EM .mymac A;B
```

The control word that is modified here is .EM, and this usage allows the macro 'mymac' to be executed, while preventing the data for the macro (A;B) from being misinterpreted as containing a control word separator.

.EZ [EASYSCRIPT]

EasySCRIPT is an early implementation of GML that existed in SCRIPT/370. The .EZ [EasySCRIPT] control word provides automatic formatting functions used by EasySCRIPT. These functions are available through a set of EasySCRIPT "tags" or through the .EZ control word directly. The EasySCRIPT tags are symbols that substitute to the appropriate .EZ control word. They are not true GML tags; they are delimited with the ampersand (&), not the GML delimiter (:). EasySCRIPT tags are included in SCRIPT/VS to allow documents already marked up with them to be processed by SCRIPT/VS.

.EZ	**{ ON [headnum] }**
	{ OFF }
	{ function line }

where:

ON initializes the EasySCRIPT tags that provide the EasySCRIPT numbering, paragraphing, and heading functions. The names of the tags are the same as the parameters of the .EZ control word that provide the associated function. ON also switches the head level definitions from the standard ones to another set used only while EasySCRIPT is in effect. The .DH [Define Head Level] control word operates on whichever set of head levels (standard or EasySCRIPT) is currently in effect.

headnum is the decimal number of the last heading that would have been used. EasySCRIPT uses this number to set the counter it uses for numbered headings. If not specified, 0.0.0.0 is assumed. If you specify &xref, EasySCRIPT resumes numbering where it left off when .EZ OFF was last processed. (&xref is the symbol EasySCRIPT uses to keep track of the current heading number.)

OFF cancels the EasySCRIPT tags, so that they are not recognized by SCRIPT/VS. OFF also switches the head-level definitions back to the standard set.

function is the name of the EasySCRIPT function to be invoked. The line of text data that follows the function name is processed by the built-in function requested. The functions are summarized below. The names of the functions are case sensitive. For example, there are two different bulleted list functions: the "B" function, in uppercase, starts a regular bulleted item, and the "b" function, in lowercase, starts a sub-bulleted item.

line is an input line of data. It must be separated from the function name by at least one blank.

Notes:

1. EasySCRIPT functions provide a fast, convenient way of formatting text and documents, particularly those that require decimal numbering. EasySCRIPT provides automatic numbering for heading levels and lists, if requested.

2. The names of the EasySCRIPT functions are the same as the names of the tags set up by ".ez on". For example, the "N3" function identifies a numbered list item at level 3. This function can be invoked with the control word

   ```
   .ez N3 text of the numbered item
   ```

   or with the tag

   ```
   &N3.text of the numbered item
   ```

   but the latter is enabled only after .EZ ON has been processed.

3. The symbol "&xref" contains the entire number of the current heading level, when EasySCRIPT's automatic numbering scheme is used. The symbols "&xref1", "&xref2", "&xref3", and "&xref4" contain the components of this number. For example, if "&xref" has the value "1.0", then "&xref1" will have the value "1", and "&xref2" will have the value "0".

4. The EasySCRIPT functions are summarized below. Note the differences in the uppercase and lowercase versions of a function name:

EasySCRIPT Functions	
Hx	Begins a decimal numbered heading of level x (1 through 4).
hx	Begins an unnumbered heading of level x.
P	Begins a major paragraph by resetting the current indention.
p	Begins a minor paragraph at the current indention.
Nx	Begins a numbered item of level x (1 through 4).
nx	Begins an unnumbered item of level x (1 through 4).
B	Begins a bulleted item.
b	Begins a sub-bulleted item.
toc	Generates a table of contents.

.FL [FLOAT]

Use the .FL [Float] control word to designate a block of text to be formatted and placed at the top or bottom of a subsequent output page or column. Such a block of text is called a "float".

The .FL [Float] control word is discussed in "Floats" on page 95. Figure 8 on page 56 shows the relationship of the .FL [Float] to the layout of a SCRIPT/VS output page.

```
.FL   ┌ ON   ┐ ┌ TOP    ┐ ┌ COL  ┐ ┌ ODD  ┐  [ORDER]
      │ OFF  │ │ BOTTOM │ │ PAGE │ │ EVEN │
      └ DUMP ┘ └        ┘ └      ┘ └      ┘
```

where:

ON	designates the beginning of a float.
OFF	designates the end of a float.
DUMP	causes SCRIPT/VS to place all unplaced floats in columns or on pages. As many extra columns or pages as needed will be added to place all pending floats. An automatic dump is generated at the end of the file.
TOP	designates the start of a float that will be placed at the top of a page or column.
BOTTOM	designates the start of a float that will be placed at the bottom of a page or column. If neither TOP nor BOTTOM is specified, the float will be placed at the top of a page or column.
ODD	designates the start of a page float that will be placed on an odd-numbered page. If neither ODD nor EVEN is specified, the float may be placed on any page.
EVEN	designates the start of a page float that will be placed on an even-numbered page. If neither ODD nor EVEN is specified, the float may be placed on any page.
COL	specifies that the float that is starting is to be formatted with the width of a single column of output.
PAGE	specifies that the float that is starting is to be formatted with the width of the entire page.
ORDER	specifies that after all previously ordered floats have been placed, the float is to be placed in the next available top or bottom float space in the column. If not specified, floats may be placed in columns or on pages in a different order than that in which they were defined, depending upon the space available on the page.

Default: ON, TOP, COL

Notes:

1. If a float is already in progress when the .FL control word is encountered, it is ended, as though .FL OFF had been processed, before the new .FL control word is processed. All .FL control words except .FL OFF and .FL DUMP cause a new float to be started.
2. If .FL OFF is encountered when no float is in progress, nothing happens.
3. If several floats are waiting to be placed, only one can be placed at the top or bottom of a column or page (except when a dump is being processed.) The remaining floats are placed, one at a time, on the tops and bottoms of subsequent pages and columns.
4. If there are several columns defined, each column may have a float at the top and at the bottom of it.
5. When the DUMP parameter is processed, TOP, BOTTOM, ODD, and EVEN are ignored, but ORDER, COL, and PAGE are not ignored.
6. Each column in a page has space available for one top and one bottom float. At the time the page is started, each pending ordered float is placed in the first location on the page that is available and has room for it. This process will continue until the

first ordered float that will not fit on the page is encountered.

.FM [FOOTING MARGIN]

Use the .FM [Footing Margin] control word to specify how much space to skip between the last line of text, on a full page, and the bottom titles, overriding the initial setting established for the device.

The .FM [Footing Margin] control word is discussed in "Allocating Space for Running Titles" on page 63. Figure 8 on page 56 shows the relationship of the .FM [Footing Margin] to the layout of a SCRIPT/VS output page.

.FM	$\left[\begin{array}{c} v \\ +v \\ -v \end{array}\right]$

where:

v specifies the amount of space to be skipped between the last line of text and the footings (bottom titles). If +v or -v is specified, the current value of the footing margin is incremented or decremented. If no .FM control word is used in the file, or if the .FM control word is used with no operand, the initial value is used. The minimum value that may be specified for the footing margin is 0. If a negative result is calculated for the footing margin, the value will be set to zero, and a message will be issued. The maximum value that can be used for the footing margin is equal to the bottom margin (.BM) minus the footing space (.FS).

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

1. The bottom titles are placed a specified amount of space below the last line of text. The location of the last line of text is explicitly defined by the .BM [Bottom Margin] control word, whether that line is actually filled or not.

2. This control word does not cause a break.

3. The .FM control word will take effect on the page after it is encountered.

Example:

```
fm .5i
```

A half-inch of space is left between the last line of text and the running bottom titles, if any have been defined.

.FN [FOOTNOTE]

Use the .FN [Footnote] control word to set aside lines of formatted output text to be positioned at the bottom of the current page, if possible, or at the bottom of subsequent pages.

The .FN [Footnote] control word is discussed in "Footnotes" on page 96. Figure 8 on page 56 shows the relationship of a .FN [Footnote] to the layout of a SCRIPT/VS output page.

```
.FN    { ON      }
       { LEADER  }
       { OFF     }
```

where:

ON	marks the beginning of the material in the footnote.
LEADER	allows the specification of a leader to be placed at the top of the footnotes on the page to separate the footnotes from the text of the page. The initial leader is a space of one line and a horizontal rule 16m in length.
OFF	marks the end of the footnote material.

Notes:

1. .FN ON starts a footnote. All lines until the subsequent .FN OFF control are put in the footnote. If .FN OFF is encountered when no footnote is in process, it is ignored.
2. There is no maximum size for a footnote. All text until a .FN OFF command is given is included in the footnote unless the footnote is prematurely ended by a disallowed control word.
3. The first footnote on a page is automatically started with a leader which may be redefined with .FN LEADER and .FN OFF. If the leader is larger than the page body plus the first line of the first footnote on the page, it will be ignored.
4. The .FN control word does not act as a break.
5. Footnotes will run across the page in a single column. The line length may be changed in the footnote.
6. When the footnote is started, offsets are cleared, and indention is set to the current .IN (indent) value and the column width is set to the line length. You must include an .OF control word if you want the footnote offset.

   When the footnote ends, any changes within the footnote to the indention, font, or certain other values in the formatting environment, are automatically restored to the values in effect before the footnote started. See Figure 33 on page 356 for a list of all the values that are automatically saved and restored after a footnote.
7. Widow zones are ignored in footnotes.
8. Deferred control words (such as .PI (Put Index) and .PT (Put Table of Contents)) are acted upon when the first line of the footnote is placed on a page -- regardless of where they were encountered within the footnote. The result is as though all such deferred control words were entered immediately after the .FN control word.
9. Refer to "Footnotes" on page 96 for further details on the placement of footnote callouts, footnotes, and footnote leaders; the splitting of footnotes; and the "floating" of unplaced footnotes.

.FO [FORMAT MODE]

Use the .FO [Format Mode] control word to cancel or restore concatenation of input lines and justification of output lines. The .FO control word also controls whether lines may be allowed to extend beyond the column boundary.

The .FO [Format Mode] control word is discussed in "Chapter 3. Basic Text Processing" on page 33.

```
.FO   [ ON     ]  [ EXTEND ]
      [ OFF    ]  [ FOLD   ]
      [ LEFT   ]  [ TRUNC  ]
      [ RIGHT  ]  [        ]
      [ CENTER ]  [        ]
```

where:

ON	restores default SCRIPT/VS formatting, including both justification and concatenation of lines. If the .FO control word is used with no operands, ON is assumed.
OFF	cancels concatenation of input lines and justification of output lines. Subsequent text is printed "as is." If an input line is longer than the defined line length, the line may be allowed to extend beyond the right margin, and no message will be issued.
LEFT	specifies that input lines are to be concatenated but not justified. The resulting output lines are left-aligned in the column. This format is sometimes called "ragged-right."
RIGHT	specifies that input lines are to be concatenated but not justified. The resulting output lines are right-aligned in the column with a "ragged-left" margin.
CENTER	specifies that input lines are to be concatenated but not justified. The resulting output lines are centered in the column.
FOLD	specifies that if an input line will not fit in the output column (in .FO OFF mode), it is to be broken and the remainder placed on the next output line(s). The line is broken at the last character that will fit on the column.
TRUNC	specifies that in .FO OFF mode, the line is to be truncated at the last character that will fit in the column.
EXTEND	specifies that in .FO OFF mode, if a line will not fit in the column, it is allowed to extend beyond the column width. This is the initial setting.

Initial Setting: ON EXTEND

Default: ON

Notes:

1. The .FO control word is a shorthand way to specify the two control words .CO [Concatenate Mode] and .JU [Justify Mode]. The effect is the same as if these two control words were specified, except that the .FO control word will end centering or right adjust mode, whereas the .CO and .JU control words will not. When format mode is in effect (.FO ON), lines are formed by shifting words to or from the next line (concatenation) and padding with extra space to produce an aligned right margin (justification).

2. This control word acts as a break.

3. Even when format mode is in effect, a line may exceed the current column width. This can happen if there is only one word on the line and this word is longer than the column width, and also if a word follows a tab and spans the right column boundary. The setting of the TRUNC, FOLD, or EXTEND option controls how these situations are handled.

4. Note that the TRUNC, FOLD, and EXTEND options may be specified as the only options of the .FO control word. In this case, the current formatting mode will be unchanged although a break will be done. For example, if .FO CENTER TRUNC is specified, and this is later followed by .FO EXTEND, the output will continue to be centered.

5. Options may be specified in any sequence. If contradictory options are specified, only the latest one will be used.

Examples:

- .fo off

  Justification and concatenation are
  completed for
  the preceding line or lines,
  but following
  lines are
  typed exactly as they appear
  in the file.

- .fo

  Justification and formatting are resumed with the next input line. Output from this point on in the file is justified to produce aligned left and right margins on the output page.

- .fo trunc

  If the current formatting mode is OFF, any lines that are longer than the current column width are truncated at the column boundary. If the current formatting mode is RIGHT or CENTER, any words that would extend past the right column boundary are truncated. If the current formatting mode is ON, the TRUNC option becomes meaningful only if the first word on a line or the first word after a tab would extend beyond the column width.

- .fo center fold

  Lines are concatenated and centered, and any lines that are longer than the column width are folded onto the next line. Note that the FOLD mode of operation will continue in effect until explicitly changed. For example, another .FO control word with only the OFF option will leave FOLD in effect.

.FS [FOOTING SPACE]

The .FS [Footing Space] control word allocates space from the bottom margin area for running bottom titles.

The .FS [Footing Space] control word is discussed in "Allocating Space for Running Titles" on page 63. Figure 8 on page 56 shows the relationship of the .FS [Footing Space] to the layout of a SCRIPT/VS output page.

.FS	$\left[\begin{array}{c}\mathbf{n}\\ \mathbf{+n}\\ \mathbf{-n}\end{array}\right]$

where:

n is the number of bottom title lines you want to appear on this and all subsequent output pages. This number may be from 0 to 6. If no number is given, 1 line is assumed. **n must be an integer from 0 to 6. This control word does not accept space units.** This number must be less than the bottom margin (.BM) minus the footing margin (.FM). If you specify +n or -n, the current value of the footing space is incremented or decremented accordingly. If the net result is a negative number, zero is assumed and a message is issued.

Initial Setting: 1

Default: 1

Notes:

1. The .FS [Footing Space] control word allocates space from the bottom margin for bottom titles. You only need to use this control word if you want more than one bottom title in your document. If the bottom margin is not big enough to accommodate the footing space plus the footing margin, an error message is generated.

2. This control word does not cause a break, and takes effect on the page after it is encountered.

3. The running bottom title control words merely cause a title line to be saved in a storage area for future use. Only the first bottom title (bottom title 1) is used at the bottom of output pages by default. To get more than one title at the bottom of your format-

ted output pages you must do two things: define the titles using the .RT [Running Title] control word, and then allocate space for the titles by using the .FS control.

4. If you do not want any bottom titles at all, the best way to accomplish this is to define the footing space as 0 (.FS 0). This is more efficient than setting the bottom titles to null (.RT B ////), because SCRIPT/VS does not have to process any titles to determine that none are wanted.

Example:

If you want three running bottom titles in your document, you could use the following sequence:

```
.rt b 3 /Chapter 4//&/
.rt b 2 ////
.rt b 1 $$&SYSMONTH./&SYSYEAR.$$
```

At this point, only bottom title 1, the one nearest the bottom of the page, is used on formatted output pages because the default footing space of 1 is still in effect. Now that the three title lines have been saved, the following control word causes SCRIPT/VS to print all three:

```
.fs 3
```

.GO [GOTO]

The .GO [Goto] control word causes SCRIPT/VS to branch to another part of the current SCRIPT/VS input file or macro.

.GO	[TO] label

where:

TO	is an optional keyword which is ignored if present; its only purpose is to allow the alternative forms ".GOTO" and ".GO TO".
label	is the name of a line set elsewhere in the current file or macro using the ... [Set Label] control word.

Notes:

1. Use the .GO control word to branch to another place in your SCRIPT file or macro. If the label designated on the .GO control word is not defined elsewhere in the current file or macro, an error message is issued, and processing terminates.

2. This control word does not cause an automatic break. The input line preceding the .GO control and the line at the label designated in the .GO control word are processed as though they were two sequential lines from the SCRIPT file.

3. Every .GO control word must refer to a label defined with the ... [Set Label] control word; but you may have more than one .GO referring to the same label.

4. .GO is particularly useful when performed conditionally as the subject of an IF statement. See the discussion of the .IF control word.

Example:

Suppose you had a SCRIPT file that was designed to recognize the variable SYSVAR5. In this example, if SYSVAR5 is set to SMALL, you want SCRIPT/VS to format the output at 36 lines per page and 4.2 inches per line. Otherwise, the default values are to be used. This could be done with the following control words:

```
.if &SYSVAR5 ne SMALL .go default
.pl 36
.ll 4.2i
...default
(etc.)
```

.GS [GML SERVICES]

Use the .GS [GML Services] control word to perform various services that may be required when writing GML APFs.

The .GS [GML Services] control word is discussed in "Chapter 14. GML Support in SCRIPT/VS" on page 159.

```
.GS   SCAN    symname   [string]

      { EXATT  }  [name [AS name]]   [[NOT] name ... name]
      { EXOPT  }

      { PURGE     }  [[NOT] name ... name]
      { PURGEMSG  }

      HCTR    [n]

      QATT    symname   [name ... name]

      RULES   [ =                         ] [ =                         ]
              [ (rulkey1 ... rulkey4)     ] [ (rulkey1 ... rulkey4)     ]

      SNAP    [tag ... tag]

      ARGS    [string]

      VARS    [symname1 ... symnamen]

      TAG     [ ON     ]
              [ ONNO   ]
              [ SYMBOL ]
              [ OFF    ]

      PREFIX  [ OFF  ]  [ OFF  ]
              [ =    ]  [ =    ]
              [ c[c] ]  [ c[c] ]
```

where:

SCAN specifies that the given string is to be scanned for GML attributes and residual text. The scan is performed according to the current rules for start tags. See the discussion of the various scanning rules below.

symname The residual line of text is set into the symbol "symname", and consists of everything to the right of the markup/content separator. If the NOATT or STOP rule for scanning is in effect, the residual line can be found without any markup/content separator. See the discussion of the scanning rules below. If no line is given with the .GS SCAN control word, the symbol "symname" is set to the value of the residual line remaining from prior automatic GML scanning, or if none exists, to the null string. A complete description of GML attribute scanning is contained in section "Chapter 14. GML Support in SCRIPT/VS" on page 159.

string specifies that the string given is to be scanned for attributes, starting at the beginning of the string. Any attributes found will be added to the attribute stack.

EXATT specifies that any attributes which have been found and placed in the attribute stack are to be executed. If a list of names is given, then only the attributes listed are executed. If the list of names is preceded by the keyword NOT, then all attributes except those listed will be executed. When an attribute is executed, it is no longer in the attribute stack.

EXOPT specifies that any user-defined options specified on the SCRIPT command are to be executed. (See "Chapter 2. Using the SCRIPT Command" on page 13.) If a list of names is given, then only the options listed are executed. If the list of names is preceded by the keyword NOT, then all options _except_ those listed will be executed. After an option has been executed, it is no longer available.

AS specifies explicitly the APF to be executed for a given attribute.

NOT specifies that the list of names that follows is not to be acted upon.

name specifies a list of attribute names.

PURGE specifies that any attributes which have been found and placed in the attribute stack are to be purged. If a list of names is given, then only the attributes listed are purged. If the list of names is preceded by the keyword NOT, then all attributes _except_ those listed are purged.

PURGEMSG is the same as PURGE, except that a message is issued to notify you what attributes were purged from the stack. This is useful if you want to be informed of unused attributes for a particular tag.

HCTR specifies that the decimal numbering contained in the symbol &@xref is to be either set with the value specified, or is to be incremented. If n is omitted, the symbol &@xref will be set to a null value, and numbering is reinitialized.

n specifies if the head counter is to be incremented at the level given, or is to be set with the value given. n may also be used to indicate whether numbering is to be in arabic or alphabetic form. If n is a decimal number, the counter at that level will be incremented, and the symbol &@xref will contain the counter value for that level. If n is of the form n1.n2.n3... (up to 32 levels of numbering are supported), the heading counter will be reset with the value given. If less than 32 numbers are given, those given are assumed to relate to the _leftmost_ positions. For example, if the value 1.3.2 is supplied, &@xref will be set to '1.3.2'. If n has the value A.0.0.b, the counters will be reset with alpha counters in the first and fourth positions.

Note: You can not use 0, 0.1, and so on as an initial value. The leading zero will reinitialize the counter to zero.

QATT specifies that the attribute stack is to be checked for the presence of specific attributes.

symname Any attributes that are checked and found to be absent from the attribute stack will be set in this symbol as array elements.

name specifies a list of attribute names to be checked.

RULES is used to specify the rules to be used for GML attribute scanning. You may specify two lists of rule keywords, each enclosed in parentheses. The first list sets the rules for scanning start tags, and the second list sets the rules for scanning end tags. The start-tag rules are used for scanning via .GS SCAN. The rules set by .GS RULES are used for scanning any tag that does not have its own rules set via .AA [Associate APF].

An equal sign in place of a rules list means to leave that set of rules unchanged. An empty list, that is, left and right parentheses with nothing in between, means to restore the default rules.

The recognized rule keywords are:

ATT Attributes are allowed. A regular attribute is in the form "name=value", where the name is a maximum of 8 characters long, and the first character is not a number. The other characters can be alphabetic, numeric, and the characters @, #, and $. The value can be in either of two forms: a single word consisting entirely of the same restricted character set that is allowed for names, with no embedded blanks, or a string delimited with single quotation marks. Any character is allowed within single quotation marks, except that a single quotation mark that is part of the string must be expressed as two single quotation marks. Leading and trailing spaces within the quotation marks are discarded. If you want these spaces, then you must use required blanks.

An item in the line being scanned that does not conform to this description of a valid attribute will be dealt with according to the VAT, STOP, and MSG rules that are currently in effect.

All regular attributes found during the scan are placed in the attribute stack, and they are available to the APF via .GS EXATT and .GS QATT.

NOATT No attributes are allowed on this tag. If NOATT is in effect, the other rules are immaterial. No scan is done for a tag that has the NOATT rule in effect, but what follows the tag in the input is treated as text (unless it is another tag, of course).

VAT specifies that "value attributes" are allowed for this tag. A value attribute is a single word composed of the same restricted character set that is allowed for attribute names, with no "name=" before it. Quoted strings are not allowed as value attributes. All value attributes found during the scan are placed on the APF invocation line, and are available to the APF in the macro local symbol &*.

NOVAT specifies that no value attributes are allowed for this tag. In this case, a word that would, with the VAT rule in effect, be recognized as a value attribute, is considered an invalid attribute.

STOP specifies that when an invalid attribute is found during the scan, the scan is stopped at that point, and the invalid attribute, and everything to the right of it, are treated as text.

NOSTOP specifies that an invalid attribute does not stop the scan. The invalid attribute is skipped and the scan continues.

MSG specifies that when an invalid attribute is found, a message is to be issued. If the STOP rule is in effect, then the message shows the beginning of the string that was not an attribute, and is treated as text. If the NOSTOP rule is in effect, the message shows the entire invalid attribute.

NOMSG prevents a message from being issued when an invalid attribute is found. The scan stops or continues, according to the STOP/NOSTOP rule, with no message.

The default rules for start tags are (ATT NOVAT STOP NOMSG). The default for end tags is (NOATT).

SNAP displays the current rules for start and end tag scanning. If a list of tag names is given, .GS SNAP also displays the .AA [Associate APF] association in effect for each tag in the list, along with the scanning rules for the tag.

ARGS resets the macro local symbols using the string that follows the ARGS keyword. For example, if a macro issued the control word ".GS ARGS one two three" then the symbol &* would contain the value "one two three". The symbol &*0 would contain the value "3", &*1 would contain "one", &*2 would contain "two", and &*3 would contain "three". The control word ".GS ARGS", with no string, resets &*0 to 0, and all the others to null.

VARS assigns the values of the current macro parameters, &*1, &*amp.2, ... , to the specified symbols. Array symbols are not allowed.

TAG specifies the GML tag parsing technique to be used.

ON enables Release 2 type GML scanning

ONNO enables Release 2 type GML scanning, suppress warning messages concerning unresolved tags.

SYMBOL enables Release 1 type GML scanning

OFF disables GML scanning

PREFIX used to specify a class mapping of GML tags to APFs. The mapping may be specified separately for tags (parameter 1) and end tags (parameter 2).

OFF	disables automatic mapping of tag name to APF.
=	APF mapping option is unchanged.
'c[c]'	specifies one or two characters that are to be used as a prefix to the tag name to create the mapping APF name.

Notes:

1. When GML markup is scanned (parsed), as many lines as necessary are read to obtain attributes, values and residual text (the text following the tag). After scanning is complete, the APF is executed. Scanning may be ended before all possible markup elements have been obtained by one of the following conditions:
   - Another tag is encountered
   - A control word is encountered at the beginning of an input line
   - End of input (EOF) is encountered
2. Residual text is treated as literal text; that is, special processing for leading blanks, tabs, and so on is not performed.

Examples:

- Suppose a GML tag with the following attributes is entered:

  ```
  :critter type=sloth toes=3
    name='Warren Jr.' food=leaves
    family=Bradypodidae food=fruit.
  ```

  Within the APF which processes the :CRITTER tag, the .GS EXATT function may be used to selectively process the attributes. For example,

  ```
  .gs exatt type
  ```

  results in the execution of the TYPE macro. The value of the attribute is provided to the macro as its parameters.

  The name of the macro to be executed may be given explicitly when it is not the same as the attribute name. For example,

  ```
  .gs exatt food as diet
  ```

  results in the execution of the DIET macro twice; once with the parameter "leaves", and again with the parameter "fruit".

  Once an attribute is processed, it is no longer available. For example, if the two preceding .GS control words are followed by

  ```
  .gs exatt
  ```

  the following macros will be executed:

  ```
  .TOES 3
  .NAME Warren Jr.
  .FAMILY Bradypodidae
  ```

- The head-level counter is initially set to "1.0.0.0.0....". The .GS HCTR control word may be used to change the head-level counter at any time. For example,

  ```
  .gs hctr 4.7.7.4
  ```

  sets the head-level counter to "4.7.7.4.0.0....". If the next head-level control word is .H4, it will be numbered "4.7.7.4"; if it is .H5, it will be numbered "4.7.7.4.1"; if it is .H3, it will be numbered "4.7.8".

  When only a single number is given with .GS HCTR, the cooresponding head-counter level is incremented, and all sublevels are reset. For example,

  ```
  .gs hctr 2
  ```

  will set the head-level counter to "4.8.0.0....".

- The .GS ARGS and .GS VARS control words provide a convenient means of setting a number of symbols simultaneously when the current macro parameters are not needed. For example,

  ```
  .gs args 1 7 7 6
  .gs vars x y z t
  ```

  is equivalent to

  ```
  .se x = 1
  .se y = 7
  .se z = 7
  .se t = 6
  ```

- Value attributes are presented to the APF which processes a tag in the same manner that attribute values are presented to the macro which processes an attribute.
  For example, if a tag is entered as:

```
:figure big.
```

the macro local symbol &* of the macro which processes the :FIGURE tag will be set to "big". If the tag is entered as:

```
:figure size=big.
```

the macro local symbol &* of the macro which processes the SIZE attribute will be set to "big".

.HM [HEADING MARGIN]

The .HM [Heading Margin] control word specifies the amount of space to be skipped between the running top titles and the first line of the text area, overriding the initial value established for the device.

The .HM [Heading Margin] control word is discussed in "Allocating Space for Running Titles" on page 63. Figure 8 on page 56 shows the relationship of the .HM [Heading Margin] to the layout of a SCRIPT/VS output page.

.HM	$\left[\begin{matrix} v \\ +v \\ -v \end{matrix}\right]$

where:

v specifies the amount of space to be skipped after the top title lines. If +v or -v is specified, the current value of the heading margin is incremented or decremented. If the calculated value of the heading margin is found to be negative, the value is set to zero and a message is issued. The maximum value that may be set for the heading margin is equal to the top margin (.TM) minus the heading space (.HS). If v is not specified, the default value for the logical device is restored.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

1. The last running top title line is placed a specified amount of space above the first line of text. If no .HM [Heading Margin] control word is included in the file, the default value is used, as determined for the logical output device.
2. This control word does not cause a break, and will take effect on the page after it is encountered.

Example:

```
.hm 3
```

Three lines are left between the running title lines and the first line of text. If a top margin of 6 lines is in effect, the last top title is printed two lines from the top of the page, followed by three more blank lines (the heading margin), and then the text.

.HS [HEADING SPACE]

The .HS [Heading Space] control word allocates space from the top margin area for running top titles. The .HS [Heading Space] control word is discussed in "Allocating Space for Running Titles" on page 63. Figure 8 on page 56 shows the relationship of the .HS [Heading Space] to the layout of a SCRIPT/VS output page.

.HS	[n / +n / -n]

where:

n is the number of top title lines you want on each subsequent output page. This number may be from 0 to 6. If no number is given, 1 is assumed. **The number must be an integer from 0 to 6. This control word does not accept space units.** The size of the top margin (.TM) minus the heading margin (.HM) must be large enough to accommodate the heading space specified. If +n or -n is specified, the current value for the heading space is incremented or decremented. If the net result is less than zero, the heading space is set to zero, and an error message is issued.

Initial Setting: 1

Default: 1

Notes:

1. The .HS [Heading Space] control word allocates space from the top margin for running top titles. You need to use this control word only if the default value of one top title is not adequate for your document. If the top margin is not big enough to accommodate the heading space plus the heading margin, an error message is generated.

2. This control word does not cause a break, and takes effect on the page after it is encountered.

3. The .RT [Running Title] control word merely causes a title line to be saved in a storage area for future use. Only the first top title (top title 1) is used at the top of output pages by default. To get more than one title at the top of your formatted output pages you must do two things: define the titles using .RT, and then allocate space for the titles by using the .HS control word.

4. If you do not want any top titles at all, the best way to accomplish this is to define the heading space as 0 (.HS 0). This is more efficient than setting the top titles to null (.RT T ////), because SCRIPT/VS does not have to process any titles to determine that none are wanted.

Example:

If you want three running top titles in your document, you could use the following sequence:

```
.rt t 1 $$&SYSMONTH./&SYSYEAR.$$
.rt t 2 ////
.rt t 3 /CHAPTER 4//&/
```

At this point, only top title 1 will be used on formatted output pages, because the default heading space of 1 is still in effect. Now that the three title lines have been saved, the following causes SCRIPT/VS to print all three:

```
.tm 8
.hs 3
```

.HW [HYPHENATE WORD]

Use the .HW [Hyphenate Word] control word to specify how a single occurrence of a word should be hyphenated if needed.

.HW	**text-word**

where:

text-word is the word that you want to hyphenate. It should be entered with hyphens showing where you want it broken.

Notes:

1. The .HW control word is a separate function from the hyphenation facility; it works regardless of whether hyphenation is ON or OFF (via the .HY [Hyphenate] control word).
2. The .HW control word does <u>not</u> define how a word should be hyphenated every time it is encountered. It specifies how to handle that word for this particular instance only. If you want a word hyphenated every time it occurs (if hyphenation is in use), then you must define hyphenation points for the word in the dictionary using the .DU control word.
3. If, while SCRIPT/VS is formatting the line, it is not necessary to break the word, the hyphens are compressed out, and they do not appear in the output. If you want to indicate a hyphen that should remain in a "compound-word," use two hyphens:

```
This is a
.hw com-pound--word
that may be broken in
either of two places.
```

.HY [HYPHENATE]

Use the .HY [Hyphenate] control word to control automatic hyphenation.

The .HY [Hyphenate] control word is discussed in "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.

```
.HY    { ON      }  [...]
       { OFF     }
       { SUP     }
       { ADD     }
       { NOADD   }
       { DICT    }
       { NODICT  }
       { ALG     }
       { NOALG   }
       ----------------------
       [SET]   MINPT   n
```

where:

ON begins automatic hyphenation of SCRIPT/VS output lines. Addenda dictionaries created with .DU [Dictionary Update] will be searched when a word is to be hyphenated. If the word is not found, the dictionaries specified with the .DL [Dictionary List] control word will be searched. If the word is still not found, the algorithmic hyphenator for the current language will be used, if one is available.

OFF causes hyphenation to be turned off.

SUP causes hyphenation to be suppressed temporarily. If hyphenation is OFF, then SUP does nothing, but if it is ON, then SUP turns it off until the next time a line space is generated. This allows you to suppress hyphenation at the end of a paragraph without having to turn it off and then on explicitly.

ADD specifies that addenda dictionaries created with the .DU control word are to be searched for words to be hyphenated.

NOADD specifies that addenda dictionaries are not to be searched.

DICT specifies that the dictionaries specified with the .DL [Dictionary List] control word are to be searched for words to be hyphenated.

NODICT specifies that the dictionaries specified with .DL are not to be searched.

ALG specifies that an algorithmic hyphenation routine is to be used, if one is available for the current language. An algorithmic hyphenator for English is supplied with SCRIPT/VS.

NOALG specifies that the hyphenation algorithm is not to be used.

SET is an optional parameter that indicates you are going to override the default hyphenation value, MINPT.

MINPT n is a positive number indicating the minimum hyphenation you want to allow. The initial value of MINPT is 4, which means that the first hyphenation point in a word must be at least four characters beyond the beginning of the word.

Initial Setting: OFF

Notes:

1. When SCRIPT/VS is formatting text, and the next word does not fit on the line, it ordinarily moves the word onto the next output line. When hyphenation is in effect, SCRIPT/VS attempts to break the word into two pieces: the longest piece that can fit on the line, and the remainder.

2. As many options as necessary may be specified; if contradictory options are given, the last is used.

.H0 - .H6 [HEAD LEVEL 0 - 6]

The control words .H0 through .H6 automatically format topic headings in SCRIPT/VS output. The definition of a particular head level may also result in an entry in the table of contents for that heading. The definition of a head level may be changed with the .DH [Define Head Level] control word, or a macro may be defined to perform whatever function you wish for .H0 through .H6, using the .DM [Define Macro] control word.

The .H0 - .H6 [Head Level 0 - 6] control word is discussed in "Chapter 6. Head Levels and Table of Contents" on page 73.

.Hn	text

where:

n is the number of the head level from 0 to 6.

text is the data to be formatted as a subject head and optionally placed in the table of contents.

Notes:

1. The .Hn control words provide several automated functions for you. They can provide a topic heading that is underscored or capitalized with a specified number of skips before it and line spaces after it. They can cause the unformatted topic head to be saved, along with the current page number and revision code character, in the table of contents utility file for automatic table of contents generation. They can also cause the heading to be numbered with a decimal number that reflects the level of the heading. These functions may be redefined using the .DH [Define Head Level] control word.

   Whether you use the default values or redefine them, the topic head that is generated gives you the function of a keep for the size of the space after the heading plus 3 lines. This keep is of the form ".KP v + v". See the discussion of the .KP [Keep] control word for information about which forms of keep may cancel or supersede this form.

2. These control words all cause breaks.

3. If a head level control word calls for an entry in the table of contents, the text goes into the table of contents as entered. You control how the table of contents entry is capitalized by how you enter the associated head level control word text.

4. See Figure 33 on page 359 for information about the default and EasySCRIPT default head-level definitions.

5. If you wanted to define a head level, such as .H3, to include function not within the scope of the .DH [Define Head Level] control word, there are two different methods you could use:

   a. You could define a .H3 macro that would provide all the function you wanted for .H3. Your macro would then operate whenever .H3 was encountered in the input file, assuming macro substitution was ON.

   b. You can provide a .DH macro that will create and maintain head-level macros as in Release 1, and then augment the function of that existing .H3 by adding or deleting lines from the macro that provides the function for .H3. The file DSMSTDH, provided with SCRIPT/VS, contains an example of such a .DH macro.

   In either case, see the discussion of the .DM [Define Macro] control word for information about defining macros.

6. The head level text is limited to 243 characters.

.IE [INDEX ENTRY]

The .IE [Index Entry] control word formats a single index entry. The .IE control word is normally used only by the .IX [Index] control word to format individual index entries constructed from .PI [Put Index] control words.

```
.IE    { H }  string
       { 1 }
       { 2 }
       { 3 }
```

where:

H indicates that an index heading is to be generated.

1 2 3 defines the level of the index entry as a primary, secondary, or tertiary entry. Appropriate formatting for spacing, indention, and so forth is provided for each level of index entry.

string is the text of the index entry.

Notes:

1. This control word causes a break.

2. The .IX [Index] control word creates .IE control words to format the index. The text of index entries generated by .IX [Index] consists of the index term and the page numbers on which the term appears, separated by two required blanks. The .IX [Index] control word also generates a .IE control word when the first character of an index entry differs from the previous index entry.

3. Since a .IE control word is executed for each index entry by the .IX [Index] control word, you may replace it with a macro of the same name to change the formatting provided for index entries. However, note that the .IX [Index] control word calls for the .IE control word in the form ".IE1", ".IE2", and so forth. This means, for example, that to replace the default index headings with your own, more sophisticated headings, you need supply only a ".IEH" macro; your macro will process all index headings, but level 1, 2, and 3 index entries will be processed by the .IE control word.

.IF [IF]

The .IF [If] control word allows a SCRIPT/VS input line to be processed conditionally.

The .IF [If] control word is discussed in "The .IF Control Word Family" on page 111.

.IF	**comparand1 test comparand2 target**
	SYSPAGE test { EVEN } target **{ ODD }**
	SYSOUT test { PRINT } target **{ TERM }**

where:

comparand1 is any string to be used as the first comparand. This comparand may be the value of a set symbol.

comparand2 is any string to be used as the second comparand. It too may be the value of a set symbol.

test is a 1- or 2-character code that tells SCRIPT/VS how to determine whether the comparison between the two comparands is true. The following codes are recognized by SCRIPT/VS:

Codes	Meaning
eq =	equal
ne ¬=	not equal
gt >	greater than
lt <	less than
ge >=	greater than or equal
le <=	less than or equal

target is any valid SCRIPT/VS input line. It may be a control word or text. If the condition is true, then the target line is processed next, with the first nonblank character after the second comparand treated as the first position of the subject line. If the condition is not true, the target line is ignored, and processing continues with the input line that follows the .IF control line.

SYSPAGE tests whether the page that SCRIPT/VS is currently processing is an even- or odd-numbered page.

SYSPAGE may have only one of the two values, EVEN or ODD.

SYSOUT tests whether SCRIPT/VS output is being directed to the offline printer (if the PRINT option has been specified), or to the terminal (if the TERM option, the default, is in effect).

SYSOUT may have only one of the two values, PRINT or TERM.

The SYSOUT keyword is provided for compatibility with SCRIPT/370 Version 3. In SCRIPT/VS, there is more variety possible in output formatting than can be determined with this keyword. The SCRIPT/VS system symbols '&$LDEV' and '&$PDEV' may be used to determine the actual logical and physical devices for which formatting is being done.

Notes:

1. The .IF [If] control word, in conjunction with .TH [Then], .EL [Else], .AN [And], and .OR [Or], allows you to construct complex logic statements.

2. The .IF control word itself does not cause a break; the target control word might, if it is processed.

3. Two special sets of comparands are recognized by the IF processing routine. These are SYSPAGE EVEN/ODD and SYSOUT PRINT/TERM. You may use SYSPAGE to determine whether the current page is even- or odd-numbered. The SYSOUT keyword is provided for compatibility with SCRIPT/370 Version 3, as noted above. When you use these two special comparands, you must capitalize the keywords; "SYSPAGE" is recognized, but "syspage" is

not. You may use any of the test codes with SYSPAGE and SYSOUT:

```
.if SYSPAGE eq EVEN (do this)
```

is the same as

```
.if SYSPAGE ne ODD (do this)
```

4. Each of the comparands may be up to 255 characters in length, and the shorter comparand will be extended to the length of the longer with trailing blanks.

5. If substitution is off when the .IF control word is processed, all valid symbols in the comparands will be resolved before the comparison is made. (Symbols containing imbedded blanks must be compared with substitution off so that the test to be performed and the target of the .IF can be identified.)

<u>Examples:</u>

- The target of an IF may be another IF. Suppose you wanted to imbed a file called ABC if it is monday afternoon. You could use the following:

  ```
  .se H = &SYSHOUR
  .se D = &SYSDAYOFW
  .if &H ge 12 .if &D eq 2 .im ABC
  ```

  This is the same as saying, "IF the hour is 12 or more, AND IF today is Monday, THEN imbed the file; OTHERWISE, go on to the next line."

- If you want the target line to contain more than one control word, you should use a special method. Since .IF is a control word, any control word separators on the line are detected before the .IF is processed. Thus, a control word in the form:

  ```
  .if &sval gt 32 .sk 5;.im fig7
  ```

  will process only the .sk 5 conditionally. The .IM control word is treated as a second control line. The following method can be used to get more than one control word to be conditionally processed:

  ```
  .if &sval gt 32 .cw ?;
         .cm ?.sk 5?.im fig7?.cw ;
  ```

  As in the previous example, only the part before the ; is processed conditionally. The remainder of the line is a .CM [Comment] line. If the condition is not true, the .CW control word is not processed, and the remaining line is treated as a comment. If the condition is true, the .CW is processed, and the new control word separator is recognized to allow the remaining line to be broken up into four active control words.

- If there is a possibility that one of the comparands may be a null symbol, another technique could be used:

  ```
  .if X&answer eq Xyes (do this)
  ```

  Now, if the symbol "answer" is null, the line will become:

  ```
  .if X eq Xyes (do this)
  ```

  Otherwise, if you had not included the Xs, a null symbol could shift the fields over like this:

  ```
  .if eq yes (do this)
  ```

  and "yes" is not a recognized condition. Note that the symbol is null only if so set by the .SE or .RV control words.

.IL [INDENT LINE]

Use the .IL [Indent Line] control word to indent the next output line.

```
.IL    [ 0  ]
       [ h  ]
       [ +h ]
       [ -h ]
```

where:

h specifies the amount of horizontal space to shift the next output line from the current margin. +h specifies that text is shifted to the right, and -h shifts text to the left.

Initial Setting: 0

Default: 0

Notes:

1. The .IL control word provides a way to indent only the next output line. The line is shifted to the right or the left of the current margin (which includes any indent or offset values in effect).
2. This control word acts as a break.
3. The .IL control word and the .UN [Undent] control word are opposites; thus, the control words .UN 5 and .IL -5 are equivalent.
4. The .IL control word may be useful for beginning new paragraphs.
5. When successive .IL and .UN control words are encountered without intervening text, or when positive or negative increments are specified for .IL control words entered without intervening text, the indent amount is newly set for the next output line, and any unused .IL or .UN is cancelled. Thus the lines

   ```
   .il 4
   .il 6m
   ```

   result in the next line being indented 6 em-spaces.
6. The .IL control word is triggered by the next text, skip, or space line.

Example:

```
.il 3m
```

   This line is preceded by the control word .il 3m, and it has enough text to show how the first line is indented differently from subsequent lines.

.IM [IMBED]

Use the .IM [Imbed] control word to process the contents of a specified file at this point in the current file. Processing continues as though the material in the imbedded file were part of the current file.

The .IM [Imbed] control word is discussed in "Imbedding and Appending Files" on page 119.

```
.IM    { file-id      }  [token1 ... token14]
       { (filename)   }
       { ('docname')  }
```

where:

file-id is an 8-character SCRIPT/VS name for the file to be imbedded. An 8-character name can be associated with an external file or data set with the .DD [Define Data File-id] control word. If no .DD has been executed for the name, an external file or data set name is built by SCRIPT/VS from the given name, using the rules for the current environment, as described in "Naming the Input File" on page 13. If a .DD [Define Data File-id] control word has been issued for the file or data set identified in the .IM control word, the 8-character name is used internally, regardless of whether the 8-character name or the parenthesized file or data set name was specified in the .IM control word.

filename is the real name of the file or data set to be imbedded, and must be enclosed in parentheses. If no .DD has been executed for the file or data set, SCRIPT/VS will assign an 8-character name to be henceforth associated with that file or data set.

docname is the name of a document to be imbedded. If the document name contains lowercase or special characters, it must be enclosed in single quotation marks (') and parentheses.

tokens are positional values with a maximum length of 8 characters to be passed to the file to be imbedded. The first token (word) becomes the value of the symbol &1, the second token becomes the value of the symbol &2, and so forth. The symbol &0 contains the number of tokens that were passed; up to 14 may be specified.

Notes:

1. Error messages, trace output, and identifiers provided by the NUMBER option of the SCRIPT command, all use the internal 8-byte name to describe a file. If MESSAGE(DELAY) is specified, a cross-reference list is provided at the end of the SCRIPT/VS output to show the associations of internal names to external file names that were used in that document.

2. Any SCRIPT/VS control word or text may be in an imbedded file. Files may be imbedded to a maximum nesting level of sixteen, but no more than 16 files may be active at one time. If you have many files that are open because of the .EF [End of File] control word, the nesting limit may be reduced. After .EF is processed, that file is left open, but it is not in the list of currently imbedded files.

3. The .IM and .AP control words perform similar functions, but .IM allows the contents of a second file to be inserted into the processing of an existing file, rather than appended to the end of it. Imbedding may be used to insert standard sets of control words at desired spots in a file, as well as for many other purposes.

4. The symbols &0 through &14 are reset whenever an .IM or .AP control word is processed. Whatever tokens are not given on a .IM line are reset. If you want to leave token &1 unset but set token &2, you may use a percent sign (%) in place of the token.

Example:

```
.im common chap4
```

The contents of the SCRIPT file whose file-id is COMMON are inserted into the processing sequence of the current SCRIPT file; when the end of the COMMON file is reached, processing of the current file resumes. The token "CHAP4" is set as the value of the symbol "&1." The file COMMON might have in it another imbed in the form:

```
.im &1
```

and this would be substituted as:

```
.im chap4
```

A different file could contain the control word

```
.im common CHAP5
```

so that &1 in COMMON is substituted with CHAP5 instead.

.IN [INDENT]

Use the .IN [Indent] control word to change the left margin displacement of SCRIPT/VS output.

The .IN [Indent] control word is discussed in "Changing the Margins" on page 39. Figure 8 on page 56 shows the relationship of .IN [Indent] to the layout of a SCRIPT/VS output page.

.IN	$\begin{bmatrix} \underline{0} \\ h \\ +h \\ -h \end{bmatrix}$ [[FOR] v1] [[AFTER] v2] [NOBREAK]

where:

h specifies the amount of space to be indented. If omitted, 0 is assumed, and indention reverts to the left margin. If you use +h or -h, the current left margin is incremented or decremented accordingly.

FOR is an optional keyword that signifies that the following parameter on the line specifies the vertical duration of the indention. If FOR is omitted, the number after 'h' on the line is taken as "v1".

v1 specifies the vertical duration of the indention. The indention changes temporarily for the vertical distance specified in v1, and then reverts to the original indention. If v1 is specified as 0, the new indention remains in effect until changed by another .IN control word. In this case, the new indention is "temporary," but its duration is continued until changed with another .IN control word. If v1 is specified as "*", the new indention is "permanent" until changed with another .IN control word. See the notes below for a discussion of the difference between a temporary and a permanent change to the indention. If v1 is not specified at all, "*" is assumed, and the new indention is permanent.

AFTER is an optional keyword that signifies that the following parameter on the line specifies the vertical distance until the new indention will take effect. If AFTER is omitted, the number after 'v1' on the line is taken as "v2".

v2 specifies the vertical distance until the new indention takes effect. The previous indention remains in effect until the vertical distance given has been formatted, and then the new indention takes effect. If v2 is specified as 0, the new indention takes effect immediately. If v2 is not specified at all, 0 is assumed, and the new indention takes effect immediately.

NOBREAK specifies that a break is not to be performed when the .IN control word is encountered.

Initial Setting: 0

Default: 0

Notes:

1. The .IN control word resets the current left margin. The new indention can be permanent or temporary, depending upon whether a duration was given in v1. The current indention for any line is comprised of the permanent indention plus the temporary indention. If you specify the horizontal distance to be indented with a plus (+) or minus (-) sign, the new indention is calculated by adding or subtracting your specified increment to the permanent indention. That is, before a new indention is calculated, any temporary indention is removed from the total, leaving only the permanent indention as the current value. (This is the only difference between a permanent change to the indention and one that is temporary until changed.) Any new .IN control word can change the current indention value. This indention remains in effect for all following lines (including new paragraphs and pages), until another .IN control word is encountered, or until the vertical duration has been ful-

filled. ".IN 0" cancels the indention, and output continues at the original left margin setting.

2. The value of h represents the amount of blank space left before text. Thus, ".in .5i" sets a left margin of one half-inch, and the text begins _after_ this blank margin area. The .TB [Tab Setting] and .OF [Offset] control words work in a similar manner.

3. This control word causes a break, unless the Nobreak keyword is used.

4. If a vertical distance is given in "v2", any blank skip or space lines inserted before text is formatted are not counted. Counting begins with the first nonblank text line, and thereafter includes any subsequent blank lines.

5. The value of the system symbol &$IN reflects the composite net indention for the next output line, including permanent and temporary components.

6. An attempt to set the indention to the left of the real left margin or to the right of the right margin results in an error message, and all indention is reset to zero.

7. If 'FOR' is not specified, the next number after h is taken as v1, unless preceded by 'AFTER'.

   If no v1 is given, * (permanent indention) is assumed.

8. If 'AFTER' is not specified, the next number after v1 is taken as v2, unless preceded by 'FOR'.

   If no v2 is given, 0 (immediate new indention) is assumed.

9. If v2 is specified as 0 or defaults to 0 (immediate new indention), and NOBREAK was specified, the new indention takes effect on the next line to be started. If no line is now started, then the new indention takes effect immediately.

Examples:

- .in 10

  All lines processed after this request are indented 10 character spaces from the left. This indention continues until another .IN control word is encountered.

- .in 0

  The effect of any current .IN and .OF control words is cancelled, and output is formatted flush left.

- The .OF [Offset] control word

  .of .7i

  is similar to the control word

  .in .7i 0 1
  (or .in .7i for 0 after 1)

  The difference is that if + or - is specified, the .IN control word calculates the new indention based on the permanent component of the total indention, while .OF uses the temporary component. Thus, .OF -n cannot reduce the current indention more than the temporary component, but .IN -n can.

.IR [INDENT RIGHT]

Use the .IR [Indent Right] control word to change the right margin displacement of SCRIPT/VS output.

The .IR [Indent Right] control word is discussed in "Changing the Margins" on page 39. Figure 8 on page 56 shows the relationship of .IR [Indent Right] to the layout of a SCRIPT/VS output page.

```
.IR   [ 0  ]  [[FOR] v1]   [[AFTER] v2]   [NOBREAK]
      [ h  ]
      [ +h ]
      [ -h ]
```

where:

h	specifies the amount of space to be indented. If omitted, 0 is assumed, and indention reverts to the right column boundary. If you use +h or -h, the current right margin is incremented or decremented accordingly.
FOR	is an optional keyword that signifies that the following parameter on the line specifies the vertical duration of the indention. If FOR is omitted, the number after 'h' on the line is taken as "v1".
v1	specifies the vertical duration of the indention. The indention changes temporarily for the vertical distance specified in v1, and then reverts to the original indention. If v1 is specified as 0, the new indention remains in effect until changed by another .IR control word. In this case, the new indention is "temporary," but its duration is until changed with another .IR control word. If v1 is specified as "x", the new indention is "permanent" until changed with another .IR control word. See the notes below for a discussion of the difference between a temporary and a permanent change to the indention. If v1 is not specified at all, "x" is assumed, and the new indention is permanent.
AFTER	is an optional keyword that signifies that the following parameter on the line specifies the vertical distance until the new indention will take effect. If AFTER is omitted, the number after 'v1' on the line is taken as "v2".
v2	specifies the vertical distance until the new indention takes effect. The previous indention remains in effect until the vertical distance given has been formatted, and then the new indention takes effect. If v2 is specified as 0, the new indention takes effect immediately. If v2 is not specified at all, 0 is assumed, and the new indention takes effect immediately.
NOBREAK	specifies that a break is not to be performed when the .IR control word is encountered.

Initial Setting: 0

Default: 0

Notes:

1. The .IR control word resets the current right margin. The new indention can be permanent or temporary, depending upon whether a duration was given in v1. The current indention for any line is comprised of the permanent indention plus the temporary indention. If you specify the horizontal distance to be indented with a plus (+) or minus (-) sign, the new indention is calculated by adding or subtracting your specified increment to the permanent indention. That is, before a new indention is calculated, any temporary indention is removed from the total, leaving only the permanent indention as the current value. (This is the only difference between a permanent change to the indention and one that is temporary until changed.) Any new .IR control word can change the current indention value. This indention remains in effect for all following lines (including new paragraphs and pages), until another .IR control word is encountered, or until the vertical duration has been ful-

filled. ".IR 0" cancels the indention, and output continues at the original right margin setting.

2. The value of h represents the amount of blank space just before the right margin.

3. If a vertical distance is given in "v2" describing when the new indention is to take effect, any blank skip or space lines inserted before text is formatted are not counted. Counting begins with the first nonblank text line, and thereafter includes any subsequent blank lines.

4. This control word causes a break unless the Nobreak keyword is used.

5. If 'FOR' is not specified, the next number after h is taken as v1, unless preceded by 'AFTER'.

   If no v1 is given, * (permanent indention) is assumed.

6. If 'AFTER' is not specified, the next number after v1 is taken as v2, unless preceded by 'FOR'.

   If no v2 is given, 0 (immediate new indention) is assumed.

7. If v2 is specified as 0 or defaults to 0 (immediate new indention), and NOBREAK was specified, the new indention takes effect on the next line to be started. If no line is now started, then the new indention takes effect immediately.

Examples:

- .ir .5i

  All lines processed after this request are indented one half-inch from the right hand side of the column. This indention continues until another .IR control word is encountered.

- .ir 0

  The effect of any .IR control word is cancelled, and subsequent lines are formatted to the right-hand margin.

.IT [INPUT TRACE]

The .IT [Input Trace] control word allows trace information about input lines to be displayed at your terminal or written to the same file as error messages.

The .IT [Input Trace] control word is discussed in "Tracing SCRIPT/VS Processing" on page 183.

```
.IT   [ OFF                  ]
      [ ON                   ]
      [ MAC                  ]
      [ SUB                  ]
      [ GML                  ]
      [ ALL                  ]
      [ CTL [cw cw...]       ]
      [ STEP                 ]
      [ RUN                  ]
      [ SNAP [name name...]  ]
```

where:

OFF terminates tracing.

ON traces macro and symbol substitution, and any control word that has been specified previously with CTL.

MAC causes each line coming out of a macro to be traced.

SUB causes each stage of symbol substitution to be traced for lines that contain symbols or GML tags.

GML causes various stages of GML processing to be traced. The number of different stages a GML tag goes through depends upon the attribute scanning rules that are in effect for the tag. GML tracing can show:

- The tag and the APF that will be executed
- Each line that will be scanned for attributes
- Each regular attribute that was found in the scan

- All value attributes that were found in the scan
- The residual text line that was found in the scan

ALL causes macro and symbol substitution, GML processing, and all control word lines to be traced.

CTL causes the control words specified to be traced before they are executed. If no control words are given with .IT CTL, then the list of control words to be traced is cleared. If some control words are given, they are added to the list. Nonexistent control words may be added to the list without causing an error, but they will never be traced because they will be detected as invalid control words before tracing would be done. The list remains intact when .IT OFF is executed, and is resumed if .IT ON is subsequently executed.

STEP causes SCRIPT/VS to "single step" through all control words that are being traced. If .IT ALL is in effect, all control words are traced. Otherwise, just those control words specified with .IT CTL are traced.

When .IT STEP is in effect, SCRIPT/VS displays the control word line, and then pauses to read a line from the terminal before executing it. The line you enter at this point can simply allow the control word execution to proceed, or you can enter another input line to be processed before, after, or instead of, the traced control word.

RUN cancels .IT STEP mode, while allowing all tracing to continue. (.IT OFF stops STEP mode, and also stops tracing.)

SNAP displays the current definitions for any symbol and macro that exist by the name or names given. If no names are given, the entire symbol and macro table is displayed. The SNAP is done without changing any other tracing that may be in effect.

Initial Setting: OFF

Default: OFF

<u>Notes:</u>

1. All trace information is written out as messages. If the MESSAGE (DELAY) option of the SCRIPT command is in effect, the trace information is written to the same SCRIPT/VS utility file as error messages.

2. .IT STEP mode can only take effect if messages (and trace information) are actually being displayed at your interactive terminal. Thus, STEP mode is not available in the batch environment or when the MESSAGE (DELAY) option of the SCRIPT command is in effect. When SCRIPT/VS reads a line from the terminal after tracing a control word line in STEP mode, it may have any of the following formats:

   null line - continue processing

   ? - verifies who is reading from the terminal. If you are stepping through control words and you are also using .TE [Terminal Input], it's easy to lose sight of which kind of read is being done from the terminal. While in .IT STEP mode, the single character ? is recognized by the control trace module and by the terminal input module, and the message "TERMINAL INPUT:" or "CONTROL TRACE:" is displayed, and another read is done. If the read comes from some other source, such as .RV [Read Variable] or .RD [Read Terminal], the ? is taken as ordinary data, just as it would be from terminal input when not in .IT STEP mode.

   STK 'data line' - the data line entered is stacked and processed after the traced control word has been processed

   PRE 'data line' - the data line entered is processed before the traced control word (the tracing is done before the control word is actually processed).

   REP 'data line' - the data line entered replaces the traced control word line, and is processed instead of it.

   'data line' - the data line is treated like a data line entered with the 'PRE' keyword.

   If the new line to be entered is also a control word line that is being traced, it will be traced before being processed, giving another opportunity to enter a line. If the line entered causes the original line to be reprocessed later, it may be traced again.

3. The trace function is initially OFF.

4. The SNAP parameter provides a selective printout of all currently defined set symbols and their values. It does not affect the current ON/OFF status of the trace control.

5. On all trace lines, the first three characters indicate which type of trace it is, as follows:

S	symbol substitution trace
M	macro substitution trace
G	GML trace
C	control word trace
***	symbol or macro SNAP line

If .IT ALL is in effect, control word lines may be traced several times. Each may be traced to show the various stages of symbol substitution, then traced again as a control word line after it has been completely substituted. You can tell which type of tracing a line represents by the first three characters of the line.

.IX [INDEX]

The .IX [Index] control word causes an index created from entries specified with .PI [Put Index] to be formatted.

The .IX [Index] control word is discussed in "Chapter 7. Indexing" on page 79.

.IX	$\left[\begin{array}{c}\underline{1}\\ n\end{array}\right]$ $\left[\begin{array}{l}\textbf{name}\\ \textbf{control}\\ /\end{array}\right]$

where:

n specifies the number of page numbers that may be reserved for the index. SCRIPT/VS must ensure that page numbers are the same on the second pass as they were on the first, when the index may have been empty. If the page number is explicitly set with .PA n or .PN n before any page number reference is made with .PT or .PI, then n is ignored. Page numbers may then increment sequentially after the index because the explicit .PA or .PN will cause the numbering to be the same on both passes. Typically, the index is at the back of a book, and no n need be specified.

name is an optional line to be used as the title of the index. If no name is given, the word INDEX is used. A head-level 1 (.H1) is generated at the top of the index using the name given or the word INDEX.

control is a control word or macro to be processed at the top of the index in lieu of the .H1. If this parameter begins with a period, it is assumed to be a control word, and not a name.

/ signals SCRIPT/VS not to generate any head level 1 for the index. Use this when you want no name on the index, you have no control word to be executed, and you don't want the default name INDEX to be generated.

Notes:

1. When the .IX [Index] control word is encountered, a head level 1 is processed. All entries that have been saved with previous .PI [Put Index] control words are then formatted and printed.

2. The index is formatted according to the line and page dimensions in effect at the time the .IX control word is encountered, not those in effect when the .PI [Put Index] control words were processed. Each line in the table has the page numbers that were in effect when the .PI [Put Index] control words were processed. Before the document begins, the page number is 0.

3. When the index is completely formatted, subsequent .PI control words will start a new index.

4. This control word acts as a break. It is not allowed in a keep.

5. When using .IX with the TWOPASS option of the SCRIPT command, be

sure that the expansion of the index during the second pass does not cause the document to expand in such a way that the page numbers established during the first pass become invalid.

.JU [JUSTIFY MODE]

The .JU [Justify Mode] control word turns justification of output lines on or off.

The .JU [Justify Mode] control word is provided for compatability with earlier releases of SCRIPT. The same function is provided by the .FO [Format Mode] control word.

```
.JU    [ ON  ]
       [ OFF ]
```

where:

ON restores right justification of output lines. If neither ON nor OFF is specified, ON is assumed.

OFF cancels justification of output lines. If concatenation is still in effect, .JU OFF results in ragged right output.

Initial Setting: ON

Default: ON

Notes:

1. Concatenation and justification are controlled by the .FO [Format Mode] control word. Ragged right output results from concatenation ON and justification OFF. The control word .FO LEFT provides this combination. Full formatting, with concatenation and justification both ON, is provided by .FO ON. "As is" output, with concatenation and justification both OFF is provided by .FO OFF. The only combination not covered by the .FO control word is concatenation OFF and justification ON, and if you need this combination, you can use the .JU control word to control it separately.

.KP [KEEP]

The .KP [Keep] control word allows you to designate blocks of text that must be kept together in the same column. There are several different ways of designating keeps, and each form has different functions and powers. When .KP is encountered inside another keep, it may end the first keep before starting the new one. If the new keep is of a form that can't end the current keep, it is ignored, and the text is kept together by virtue of being part of the larger keep.

The .KP [Keep] control word is discussed in "Keeps" on page 93.

```
.KP    { ON     }
       { FLOAT  }
       { DELAY  }
       { INLINE }
       { OFF    }
       { v + v  }
       { v      }
```

where:

ON starts a regular keep. The text within a regular keep is separate from the text outside of it, and no output line can be built from text part of which came from inside and part from outside the keep. A regular keep is put in this column if it will fit, and otherwise an immediate column eject is done. The regular keep appears in the output in the same relative location as it was in the input. A regular keep ends any other keep before starting. The ON parameter causes a break.

FLOAT starts a floating keep. A floating keep is put in this column if it will fit; otherwise it goes in the next column. The current column is filled using the text following the floating keep. .KP FLOAT ends any other keep before starting.

DELAY starts a delayed keep. A delayed keep is always printed in the next column, even if there is room for it in this column, and the current column is filled using the text following the delayed keep. A delayed keep, in effect, acts like a floating keep that did not fit in the current column. .KP DELAY ends any other keep before starting.

INLINE starts an inline keep. An inline keep flows with the preceding and following text. No separation of material inside and outside the keep is done, but formatting continues as though no keep were designated. All lines that contain text from within the keep are then kept together, and if column balancing is done, the entire keep is moved as a block from one column to another. .KP INLINE ends an inline keep or a keep in the form ".KP v + v" or ".KP v" before the inline keep is started, but if a regular, floating, or delayed keep is in process, .KP INLINE is ignored.

OFF marks the end of a regular, floating, delayed, or inline keep. .KP OFF also ends a keep of a designated depth, but is not required.

v + v starts a keep of a designated vertical depth. The depth of the keep is determined by adding up all the separate v's given. For example, .KP 3 + 2 would start a keep for 5 lines, and .KP 2i + 3 would start one for 2 inches plus 3 lines. (This is the only control word that allows you to add up different space units to get a single result.) This form of keep is used by the head level control words .H0 - .H6 [Head Level 0 - 6]. A keep for a designated depth need not be explicitly ended with .KP OFF. It will be ended automatically when its depth has been filled. A keep of the form .KP v + v may end another keep of the same form or a keep of the form .KP v. If an inline or higher keep is in process when .KP v + v is encountered, the .KP v + v is ignored. Any head level control word also ends a keep of the "v + v" form.

v starts a keep of a designated depth specified by v. When the designated depth has been filled, a keep of the "v" form is automatically ended. A keep of the "v" form may end another keep of the same form before starting, but if any other form of keep is in process, .KP v is ignored.

Notes:

1. Keeps started with .KP ON, .KP FLOAT, and .KP DELAY all operate with a separate environment from ordinary text. No output line may be formed by concatenating text from inside the keep to text from outside of it. When the keep is started, the current indention and certain other values are saved. Offsets and undents are cleared so that the indention at the beginning of the keep is set to the basic indention currently in effect, and the maximum column width is set to the width of the current column. When the keep is ended, the original text values are restored automatically. This means that if you change the indention, formatting mode, hyphenation, double spacing, or certain other things, you need not restore them when the keep is ended. See Figure 34 on page 360 for a list of the active environment values that are saved and restored for these keeps.

2. Keeps started with .KP INLINE, .KP v + v, or .KP v are not separated from the surrounding text. Output lines may be formed by concatenating text from inside the keep to text from outside of it. No environment values are saved or changed, and the text within the keep flows with neighboring text. Formatting continues for these keeps as though no keep had been started, but all output lines encompassed by the keep are kept together in the same column of output.

3. Certain control words are not allowed within a keep. If one of the disallowed control words is encountered, the keep is immediately ended, as though .KP OFF had been processed, and then the disallowed control word is executed. A warning message is issued, telling you what control word ended the keep. See Figure 27 on page 355 for a list of the disallowed control words.

4. If a keep is too large to be placed on the page, an error message is issued.

.LB [LEADING BLANK]

The .LB [Leading Blank] control word is generated by SCRIPT/VS and executed whenever an input line that starts with a blank is processed.

.LB	

Notes:

1. This book states in several places that a leading blank on an input line causes a break. This is actually done by generating and executing a .LB control word whenever a line with a leading blank is processed, and the function of the .LB control word is identical to that of the .BR control word.

   If you wish to have leading blanks perform some other function, you can define a .LB macro with .DM [Define Macro], and, assuming macro substitution is ON, your .LB macro will be executed whenever a leading blank is processed. Note, however, that after the .LB control word or macro is processed, the leading blank is still on the line, and it is processed as part of that text input line. In other words, you cannot use the .LB macro to remove leading blanks from a line.

2. No .LB function is performed for lines processed in literal mode (.LI [Literal]).

.LI [LITERAL]

The .LI [Literal] control word allows all input lines, including those that begin with periods, to be processed as text.

The .LI [Literal] control word is discussed in "Guidelines for Entering Text and Control Words In SCRIPT/VS" on page 50.

.LI	[<u>1</u> n **ON** **OFF** **line**]

where:

n specifies the number of lines to be treated literally. If omitted, 1 is assumed.

ON starts an open-ended literal mode, in which every line read is treated as literal text. After this control word is processed, SCRIPT/VS reads input lines looking only for ".LI OFF" beginning in column 1 on a line by itself.

OFF terminates literal mode if it was ON, or if n was given and has not been exhausted.

line is the line to be treated as literal text.

Default: 1

Notes:

1. Ordinarily, any SCRIPT/VS input line that begins with a period is interpreted as a SCRIPT/VS control word. The LITERAL control word

causes the following n lines to be processed as normal input lines even if the first character of one of the lines is a period. If .LI ON is encountered, all subsequent lines except .LI OFF (which must be recognized to cancel literal mode) are treated as literals.

2. When literal mode is in effect, null lines, lines with leading blanks, and lines with leading tabs do not cause a break. Null lines, however, do cancel continuation if the previous line ended with a continuation character.

Example:

If a text line must begin with a period:

```
Study the following control words:
.fo off
.li on
.LB [Leading Blank]
.LT [Leading Tab]
.NL [Null Line]
.LI [Literal]
.li off
```

These lines are formatted as:

```
Study the following control words:
.LB [Leading Blank]
.LT [Leading Tab]
.NL [Null Line]
.LI [Literal]
```

If formatting mode had not been turned OFF with .fo off, the same lines would be processed as:

Study the following control words: .LB [Leading Blank] .LT [Leading Tab] .NL [Null Line] .LI [Literal]

.LL [LINE LENGTH]

The .LL [Line Length] control word specifies the width of running titles, running headings, and running footings. It also changes the column width, which governs the width of text lines, if the latter has never been set explicitly with .CL [Column Line Length].

The .LL [Line Length] control word is discussed in "Chapter 4. Defining a Page Layout" on page 55. Figure 8 on page 56 shows the relationship of the .LL [Line Length] to the layout of a SCRIPT/VS output page.

.LL	$\left[\begin{matrix} \mathbf{h} \\ \mathbf{+h} \\ \mathbf{-h} \end{matrix}\right]$

where:

h specifies an output line length not greater than the output device capability. If no value is specified for h, the default value established for the device being used will be taken.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

1. The .LL control word sets the line length for running headings and footings, footnotes, and running titles. The .CL [Column Line Length] control word sets the line length for text in the body of the page, but if the column width has never been explicitly set, it has the same value as the line length.

2. This control word takes effect on the page after it is encountered. If it also performs a .CL function, however, the new column width takes effect immediately.

3. This control word causes a break.

.LT [LEADING TAB]

The .LT [Leading Tab] control word is generated by SCRIPT/VS and executed whenever an input line that starts with a tab is processed.

.LT	

Notes:

1. This book states in several places that a leading tab on an input line causes a break. This is actually done by generating and executing a .LT control word whenever a line with a leading tab is processed, and the function of the .LT control word is identical to that of the .BR control word.

   If you wish to have leading tabs perform some other function, you can define a .LT macro with .DM [Define Macro], and, assuming macro substitution is ON, your .LT macro will be executed whenever a leading tab is processed. Note, however, that after the .LT control word or macro is processed, the leading tab is still on the line, and it is processed as part of that text input line. In other words, you cannot use the .LT macro to remove leading tabs from a line.

2. No .LT function is performed for lines processed in literal mode (.LI [Literal]).

.LY [LIBRARY]

Use the .LY [Library] control word to cause symbol and macro definitions to be retrieved from a library defined with the LIB option on the SCRIPT command.

.LY	[<u>ON</u> / SYM / MAC / OFF]

where:

ON specifies that both symbol and macro definitions may be retrieved from a library. This is the default.

SYM causes unresolved symbol values to be retrieved from the library. If SYM is specified, the library will not be used to resolve undefined macros from the library (unless MAC or ON is also specified).

MAC causes unresolved macro definitions to be resolved from the library. If MAC is specified, no undefined symbol values will be resolved from the library (unless SYM or ON is also specified).

OFF indicates that use of the library for symbol values and macro definitions is to stop. This is the initial setting.

Initial Setting: OFF

Default: ON

Notes:

1. Use of the library to resolve symbol values and macro definition is expensive in processing time. This is especially true for forward referencing of symbol values where there are normally many potentially unresolved symbols. For this reason, the .LY control word is provided to control library lookup. The .LY control word allows you to tell SCRIPT/VS that, if unresolved symbols or macros are <u>used</u> in a document, to attempt to resolve these from a library.

2. Symbol values or macro definitions may be explicitly <u>set</u> from the library, regardless of the setting of the .LY control word, using the LIB option of the .SE and .DM control words.

.MC [MULTICOLUMN MODE]

The .MC [Multicolumn Mode] control word restores multiple column processing after it has been temporarily suspended by .SC [Single Column Mode].

.MC	

Notes:

1. The .MC control word cancels a temporary single-column mode that was put into effect by the .SC [Single Column Mode] control word. If there was no .SC control word preceding this control word, it has no effect, other than to cause a break.

2. This control word is not allowed in a keep.

3. The .SC control word saves the current column definition, and starts a temporary single-column processing mode. The column definition that was in effect when .SC saved it might actually have been a multiple-column definition, or it might have been a single column definition. The .MC control word is, perhaps, misnamed. What .MC actually does is to restore the column definition that was saved by .SC, however many columns that definition called for. The column definition saved by .SC and restored by .MC includes the number of columns and their positions and the column width. If two .SCs are processed without an intervening .MC, then it takes two .MCs to restore the original column definition that existed before the first .SC. The first .MC restores the single column definition that existed, by virtue of the first .SC, when the second .SC was processed.

4. The .CD [Column Definition] control word starts an entirely new column definition, and cancels any .SCs and .MCs that may be in effect.

.ME [MACRO EXIT]

Use the .ME [Macro Exit] control word to cause SCRIPT/VS to terminate processing of a macro.

.ME	[line]

where:

line is any valid SCRIPT/VS input line. If present, it will be saved until after the macro is closed, and executed in the macro caller's environment.

Notes:

1. The .ME control word has meaning only when the current input source is a macro. If the current input source is not a macro when .ME is encountered, nothing happens.

2. The .EF [End of File] control word, when encountered in a macro, causes that macro and all macros that called it to be closed, up to the parent file that first invoked the top macro in the chain. In other words, .EF ends nested sources until it has ended the current file. .ME, by contrast, ends only the current macro, and no other nested input source.

Example:

The .ME control word may be used to set a macro caller's local symbol. For example, suppose macro OUTER calls macro INNER, which contains:

```
.me .se *local = &*symbol
```

The .ME will end macro INNER and return control to macro OUTER, and will cause the .SE control word to be executed as if it were part of macro OUTER. The value of &*symbol is taken from macro INNER's local symbol, since symbol substitution is performed before control word execution. But the symbol to which it is assigned is macro OUTER's local symbol &*local, since that is where the control word is executed.

.MG [MESSAGE]

The .MG [Message] control word is used to write out a message. It may be used to provide diagnostic messages from macros.

.MG	/[mid]/[message text]/

where:

/	is any delimiter character. The first nonblank character will be taken as the delimiter character.
mid	is the message id. This string must not be longer than 16 characters and the last character must be R, I, W, E, S, or T. This final letter is used to establish the severity of the message, and the same meanings apply as for regular SCRIPT/VS messages. If a null message id is specified, the message is considered to be of type I, an information message. If the message is type R (Response), you must provide the terminal read using .RV or .TE; the Message control word will not do this for you. The message id is not printed unless the MESSAGE (ID) command option is in effect.
message text	is the text of the message. It may be any string of characters.

Notes:

1. Messages generated by .MG may cause SCRIPT/VS processing to terminate. Type S (severe) or type T (terminating) messages always terminate processing, and type E (error) messages terminate processing if the CONTINUE option of the SCRIPT command is not in effect.
2. The delimiter character between the strings may be any unique character which does not occur within the strings themselves.
3. When a message is displayed, a prefix of "+++" appears before the id or text to indicate the message was generated by .MG. If the net message is null, the prefix only is displayed. This can happen if you do not specify any message id or text, or if you specify a message id and no text, but the MESSAGE (ID) option is not in effect.
4. If the .MG line has no data at all, it is ignored.
5. If the message header is longer than 16 characters or if it does not end with one of the valid type-codes, it is considered an invalid control word parameter, and an error message is issued.

Example:

The control word:

```
.mg /msg001e/This is a message/
```

is displayed as:

```
+++MSG001E This is a message
```

if MESSAGE(ID) is in effect, or:

```
+++ This is a message
```

if MESSAGE(ID) is not in effect.

.MS [MACRO SUBSTITUTION]

Use the .MS [Macro Substitution] control word to initiate or cancel automatic macro calls during SCRIPT/VS processing.

.MS	[<u>ON</u> / OFF]

where:

ON causes SCRIPT/VS to begin searching for macro names when it encounters unrecognized control words.

OFF causes SCRIPT/VS to stop searching for macro names during processing.

Initial Setting: OFF

Default: ON

Notes:

1. SCRIPT/VS macros can be defined with the .DM control word. However, SCRIPT/VS does not ordinarily recognize and process macros unless the .MS control word has been used. When macro substitution is OFF (the initial setting for SCRIPT/VS processing), macros that have been defined via the .DM [Define Macro] control word are treated as invalid control words.

2. Even when macro substitution is OFF, a macro can be explicitly invoked via the .EM [Execute Macro] control word.

.NL [NULL LINE]

The .NL [Null Line] control word is generated by SCRIPT/VS and executed whenever a null line is processed.

.NL	

Notes:

1. Whenever SCRIPT/VS encounters a null input line, that is, a line whose length is zero, it generates and executes a .NL control word. The .NL control word does nothing, except to reset line continuation, in case the previous line ended with a continuation character.

   If you wish to have null lines perform some other function, you can define a .NL macro with .DM [Define Macro], and, assuming macro substitution is ON, your .NL macro will be executed whenever a null line is processed.

2. No .NL function is performed for lines processed in literal mode (.LI [Literal]). A null text line, however, does reset continuation if the previous text line ended with a continuation character.

3. A null line may originate from a number of sources. Because of this, you should define a .NL macro only when a specific use in a certain part of a document requires it. Null lines may originate from:

   - A source input file (not all systems in which SCRIPT/VS operates allow this).
   - From terminal input (.TE).
   - A non-null line that becomes null as a result of substitution.
   - A macro line that is null.

.OC [OUTPUT COMMENT]

Use the .OC [Output Comment] control word to place comments and carriage control characters in the output data stream. Such comments are not examined by the formatter. This control word is designed for the systems programming user of SCRIPT/VS and must be used with caution.

.OC	{ line }
	 { 'string['] }

where:

line may be anything, since it is not used in formatting the output. However, since this is a control word, the input line is scanned for control word separators. The line given will not be preceded by a carriage control character when output is being directed to a printer. The first character in the given line will be taken as a carriage control character, and you must ensure that a valid carriage control character for the output device is provided.

Output comments of the line form are not synchronized with the formatted text; they are written to the output destination immediately before the page on which the text surrounding them appears.

string may be anything delimited by single quotation marks. However, since this is a control word, the input line is scanned for control word separators. The string will be placed in the output exactly as given, in the same place relative to the input text that it was specified in the input. For this reason you should be extremely careful to control the contents of the string.

The width of output comments is considered to be zero. The depth of output comments is the same as the depth of normal line spacing on the device.

Notes:

1. The .OC control word allows comments to be placed in the output data stream. They are not examined by the formatter and thus, unless they are correctly interpreted by the output device, the output will be disrupted.

   The .OC may be used, for example, to control a printer in a certain way, such as to transmit codes recognized by certain printers.

2. In order for a string to be treated as a string, the single quotation mark must be separated from the .OC [Output Comment] control word with one and only one blank.

3. Trailing blanks in a string will be discarded if 'string' is the last entry on the input line.

.OF [OFFSET]

Use the .OF [Offset] control word to indent all but the first line of a block of text.

```
.OF    [ 0  ]
       [ h  ]
       [ +h ]
       [ -h ]
```

where:

h specifies the horizontal size of the offset. If you specify +h or -h, the old offset value is incremented or decremented the specified amount to establish the new offset size. If "h" is omitted, the new offset size is 0.

The next output line to be formatted after the .OF control word has been processed is formatted at the left margin established by the .IN [Indent] control word, with no added offset. For all subsequent lines, the left margin is established by adding the offset (.OF) to the size of the indent (.IN).

Initial Setting: 0

Default: 0

Notes:

1. A .OF control word does not take effect until after the next line is formatted. The offset remains in effect until a .IN [Indent] control word or another .OF control word is encountered.

   The .OF control may be used within a section which is also indented with the .IN control. Note that .IN settings take precedence over .OF, however, and any .IN request clears all offsets.

   If you want to start a new section with the same offset as the previous section, you need only repeat the .OF h request.

2. This control word acts as a break.

3. The .IL [Indent Line] and the .UN [Undent] control words can be used to shift only the next line to the left or right of the current margin.

4. Tabs should be used whenever possible to format numbered or bulleted lists, to ensure that the first text word on the line is even with subsequent offset lines. The items in this "Notes" section are created using offsets and tabs.

5. The .OF control word is triggered by the next text, skip, or space line.

Examples:

1. Starting an offset:

.of 7
The line immediately following the
        .OF control word is printed at the current left margin. All lines thereafter (until the next indent or offset request) are indented seven character spaces from the current margin setting. These two examples were processed with .OF control words in the positions shown.

        2. Ending an offset:

.of
The effect of any previous .OF request is cancelled, and all output after the next line continues at the current left margin setting.

.OR [OR]

The .OR [Or] control word can be used in conjunction with the .IF [If] control word to process SCRIPT/VS input lines conditionally. The result of the test performed is logically ORed to the result of the most recently performed .IF [If], .AN [And], or .OR [Or] control word to determine whether the target is to be processed.

The .OR [Or] control word is discussed in "The .IF Control Word Family" on page 111.

.OR	**comparand1 test comparand2 target**
	SYSPAGE test { EVEN } target **{ ODD }**
	SYSOUT test { PRINT } target **{ TERM }**

where:

comparand1 is any string to be used as the first comparand. This comparand may be the value of a set symbol.

comparand2 is any string to be used as the second comparand. It too may be the value of a set symbol.

test is a 1- or 2-character code that tells SCRIPT/VS what comparison to make between the comparands. The following codes are recognized by SCRIPT/VS:

Codes	**Meaning**
eq =	equal
ne ¬=	not equal
gt >	greater than
lt <	less than
ge >=	greater than or equal
le <=	less than or equal

target is any valid SCRIPT/VS input line. It may be a control word or text. If this condition or the result of the most recently performed .IF [If], .AN [And], or .OR [Or] is true, then the target line is processed next, with the first nonblank character after the second comparand treated as the first position of the line. Otherwise, the target line is ignored, and processing continues with the input line that follows the .OR control line.

SYSPAGE is a special .OR keyword that tests whether the page that SCRIPT/VS is currently processing is an even- or odd-numbered page.

See the .IF [If] control word for a description of SYSPAGE, EVEN, and ODD.

SYSOUT is a special .OR keyword that tests whether SCRIPT/VS output is being directed to a printer or to the terminal.

See the .IF [If] control word for a description of SYSOUT, PRINT, and TERM.

Notes:

1. The .AN [And] and .OR [Or] control words, in conjunction with .IF [If], .TH [Then], and .EL [Else], allow you to construct complex logic statements.
2. The .OR control word itself does not cause a break; the target control word might, if it is processed.
3. Each of the comparands may be up to 255 characters in length, and the shorter comparand will be extended to the length of the longer with trailing blanks.
4. If substitution is off when the .OR control word is encountered, all valid symbols in the comparands will be resolved before the comparison is made. (Symbols containing imbedded blanks must be compared with substitution off so that the test to be performed and the target of the .OR can be identified.)

Example:

The following input line

```
.if &a eq &b .or &c eq &d .ty Yes.
```

```
is equivalent to the input lines        .if &a eq &b
                                        .else .if &c eq &d
                                        .then .ty Yes.
```

.PA [PAGE EJECT]

Use the .PA [Page Eject] control word to force subsequent text onto a new page of output, even if the current page has not been filled.

```
.PA   [ n        ]
      [ +n       ]
      [ -n       ]
      [ NOSTART  ]
      ----------------------------
      [ { ODD  }  [ ON  ] ]
      [ { EVEN }  [ OFF ] ]
```

where:

n	specifies the page number of the next page. If n is not specified, sequential page numbering is assumed, and the next page number is one greater than the current page number. n must be an arabic number with no decimal point.
+n	specifies that the next page should have a number that is equal to the normal next sequential page number plus n. n must be a non-decimal arabic integer.
-n	specifies that the next page should have a page number that is equal to the next sequential page number minus n. If subtracting n from the next page number yields a negative number, an error message is issued, and the control word is ignored. The maximum allowed page number is 9999.
NOSTART	causes the current page to be ended, but the next page will not be started until some data causes it to be started or a control word that requires the page to be started is processed. After .PA NOSTART, the page definition (including running headings and footings), may be changed until the page is started.
ODD	causes one or two page ejects, such that the new page is odd numbered.
EVEN	causes one or two page ejects, such that the new page is even numbered.
ON	defines the start of odd or even page eject mode. This mode is ended by specifying OFF or the start of another .PA even or odd mode, or n. In odd or even page eject mode, output is formatted on odd pages only, or even pages only, whichever the case may be, and the other pages are left blank, except for running titles and headings and footings.
OFF	defines the end of odd or even page eject mode.

Notes:

1. The minimum page number is 1, and the maximum is 9999. If a .PA control word attempts to set the page number outside this range, a message is issued, and the control word is ignored.
2. Whenever a .PA control word is encountered, the rest of the current page is skipped after printing any text lines accumulated thus far. The next page is started, unless .PA NOSTART was specified. Starting a page includes formatting running headings, running footings, and running titles for the page, and establishing the page dimensions for the page. These things are then fixed for the duration of the page, and may not change until the next page is started.
3. If you use the STOP option of the SCRIPT command, SCRIPT/VS waits

for you to enter a null line (with the Return or Enter key) before starting the new page.

4. This control word acts as a break. It is not allowed in a keep.

5. If you want to change any page dimensions or define new running titles or running headings and footings for a new page, the appropriate control words must be processed before the .PA control word (except when NOSTART is specified). These control words are listed in Figure 26 on page 354. Note that at the beginning of SCRIPT/VS processing, the first page has not yet been started.

6. If .PA n (or +n or -n) is specified after .PN FRAC is specified, the page eject will occur, but the page number will not be reset. This is because the page number change to fractional pagination is pending.

7. Figure 29 on page 356 lists the control words that require a page to be started; they will cause one to start if one is not already started.

Examples:

- To start the next sequential page:

  ```
  .pa
  ```

  The rest of the current page is skipped. The top titles and page number are put in the top margin of the next page, and output resumes.

- To repeat a page number:

  ```
  .pa -1
  ```

  The new page will have the same page number as the preceding page. The calculation is done after establishing the next sequential page number.

.PF [PREVIOUS FONT]

Use the .PF [Previous Font] control word to resume the use of the font whose id was last saved using the .BF [Begin Font] control word.

The .PF [Previous Font] control word is discussed in "Using Fonts with the IBM 3800 Printing Subsystem" on page 48.

.PF	

Notes:

1. If the .PF control word is used when there is no previously saved font, the default font for the output device will remain effective.

2. The font save stack is 16 entries deep, and the stack is saved and restored by .SA [Save Environment] and .RE [Restore Environment], respectively.

.PI [PUT INDEX]

The .PI [Put Index] control word saves the specified lines for use in building an index. The .IX [Index] control word causes this index to be inserted in the document. The .PI [Put Index] control word is ignored unless enabled by the INDEX option of the SCRIPT command.

The .PI [Put Index] control word is discussed in "Chapter 7. Indexing" on page 79.

```
.PI    [ REF   ]  [KEY /key1/key2/key3/]
       [ ORDER ]
       [ START ]
       [ END   ]
       - - - - - - - - - - - - - - - - - - - - - -
       /term1[/term2[/term3[/term4[/]]]]
```

where:

REF	specifies that this control word refers to an index reference. An index reference differs from an index term in that the words 'See' or 'See also' will be prefixed to the index entry supplied and no page number will be printed in the index. 'See' will be prefixed if this is the only entry at this level under the preceding higher level entry, 'See also' will be prefixed if there are other entries.
ORDER	specifies that the page number of the entry in the index is to be placed in front of all previous page numbers for the term given.
START	specifies that the index entry being defined is the start of a reference to a range of pages. The START keyword will be ignored when the REF keyword is specified.
END	specifies that the index entry being defined is the end of a reference to a range of pages. The END keyword will be ignored when the ORDER or REF keywords are specified.
KEY	indicates that the sort keys for the index entries will be given explicitly. If KEY is omitted, the sort keys are developed from the index terms.
key1-key3	specifies the sort keys to be used for the index terms. if any key is null, that key will be developed from the index term.
term1	is the level 1, or primary, index term.
term2	is the level 2, or secondary, index term.
term3	is the level 3, or tertiary, index term.
term4	is the text to be used in the index in place of the current page number. If a null term4 is specified, no page number will be printed in the index. If no term4 is specified, the current page number will be printed in the index.
/	is any delimiter character that does not appear in any term or key.

Notes:

1. Index entries are sorted on keys developed from the index terms, except when explicitly given with the KEY parameter. The key is developed by folding the term to uppercase, removing any characters to be ignored (as specified by the IXI parameter of the .DC [Define Character] control word), and translating any characters to be considered blanks (as specified by the IXB parameter of the .DC [Define Character] control word).

.PL [PAGE LENGTH]

The .PL [Page Length] control word specifies the vertical length (depth) of output pages. The value specified overrides the standard page length which is established for each logical device.

Figure 8 on page 56 shows the relationship of the .PL [Page Length] to the layout of a SCRIPT/VS output page.

.PL	$\begin{bmatrix} \text{v} \\ \text{+v} \\ \text{-v} \end{bmatrix}$

where:

v specifies the vertical length, or depth, of output pages. If no value is specified for v, the default value for the device will be used. This number should be the same as the physical size of the paper being used. However, when formatting for a printer logical device, it may be different, as explained below. The minimum value for the page length is the sum of the top margin (.TM) and the bottom margin (.BM) plus one line. The maximum value that may be specified for the page length is as established for each logical device. If +v or -v is specified, the current page length is incremented or decremented accordingly.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

1. The .PL control word allows varying paper sizes to be used for output. (The logical device specified in the DEV option of the SCRIPT command implies a default page length, but this can be overridden with .PL.) Page length may be changed anywhere in a file, with the change effective on the page after the control word is encountered.

2. This control word does not cause a break. It is not allowed in a keep.

3. If the output is in printer format, the page length value need not be the same as the actual number of print lines on the real paper, because SCRIPT/VS will cause the printer paper to be ejected to the top of the next real page whenever a new SCRIPT/VS page is started. Thus, a SCRIPT/VS page may occupy less than a real page or more than one real page, and the output will be newly aligned to the paper each time a SCRIPT/VS page is started.

4. The previous rule notwithstanding, if you define a top margin (.TM) and a heading space (.HS) and heading margin (.HM) such that SCRIPT/VS needs to print data within the first three lines on a page, no printer page ejects can be done. Instead, SCRIPT/VS uses the page length value to find the top of the next page. It is, therefore, good practice to keep the .PL value accurate, so that it reflects the true depth of the page under SCRIPT/VS control.

5. The maximum value of the page length that may be set is governed by the value established as the maximum for the logical device for which formatting is being performed.

6. If the running headings and footings that are defined for a page fill up the page so that no room is left for text, SCRIPT/VS terminates with an error message. The depth of running headings and footings cannot be predicted at the time they are defined, because they are formatted to the current line length (.LL) when a page is started. The same running heading can occupy differing amounts of vertical space on different pages if the line length changes.

Example:

```
.pl 84
```

Page length is set to 84 lines. This is the correct size for 14 inch printer paper when printing at six lines per inch.

.PM [PAGE MARGINS]

The .PM [Page Margins] control word causes SCRIPT/VS to shift the formatted output of each page to the right and is used in conjunction with the BIND option of the SCRIPT/VS command.

The .PM [Page Margins] control word is discussed in "Changing the Page Margin" on page 57.

```
.PM   [ { h  }  [ h2  ] ]
      [ { +h }  [ +h2 ] ]
      [ { -h }  [ -h2 ] ]
```

where:

h specifies the amount of horizontal space (binding) to shift odd-numbered output pages. This overrides the binding value specified with the BIND option of the SCRIPT command. +h specifies that the pages are to be shifted to the right, and -h to the left, of the current binding (as established by the BIND option or a previous .PM control word.) If h is omitted, a default value of that specified with the BIND option on the SCRIPT command is used.

h2 specifies a binding for even-numbered pages. If h2 is omitted, h applies to all pages.

Notes:

1. The actual (or potential) page number of the output page is controlled by the .PA [Page Eject] and .PN [Page Numbering Mode] control words, which are used to specify even and odd page numbers. Consequently, you can have two or more even-numbered (or odd-numbered) pages in a row.
2. Bindings can be specified in numbers of character spaces or in space units.
3. If the BIND option is not specified, it defaults to two character spaces. This allows room for potential revision codes for the first column. (Revision codes for subsequent columns are placed in the gutter between columns.) If sufficent room is not provided for revision codes, they are discarded.
4. The .PM [Page Margins] control word takes effect on the next page.

.PN [PAGE NUMBERING MODE]

The .PN [Page Numbering Mode] control word allows you to control various aspects of page numbering, including the format of the page number, and whether it is to be shown in running titles or running headings and footings that call for it.

The .PN [Page Numbering Mode] control word is discussed in "Page Numbers" on page 64.

```
.PN   { OFF          }
      { OFFNO        }
      { ON           }
      { ARABIC       }
      { ROMAN        }
      { ALPH         }
      { FRAC         }
      { NORM         }
      { PREF string  }
      { n            }
```

where:

OFF suppresses the display of page numbers in running titles and running headings and footings, although pages are still sequentially numbered internally. Symbols set with .SE [Set Symbol] to the current page number will contain the correct number of the page on which they were processed.

OFFNO	suppresses both page number display and internal page numbering. The current page number set with .SE remains the same for all pages until .PN OFFNO is ended with .PN ON.
ON	cancels .PN OFF or .PN OFFNO, so that internal numbering of pages is resumed, and the current page number can be displayed in running titles and running headings and footings.
ARABIC	causes the following page numbers to be represented as standard arabic numerals. The ARABIC keyword may be abbreviated as AR.
ROMAN	causes page numbers to be represented as lowercase roman numerals. Page numbers greater than 3999 are not supported with the ROMAN option. The ROMAN keyword may be abbreviated as RO.
ALPH	causes alphabetic page numbering to be started. In this mode, the number 1 is converted to a, 2 to b, 26 to z, and 27 to aa. The number 1978 is represented as bxb. The ALPH keyword may be abbreviated as AL.
FRAC	causes fractional pagination to begin. The next time a page eject occurs that would normally increment from an even to an odd number, the even number (for example, 20) is saved, and numbering starts with a fractional sequence, in this case, 20.1, 20.2, 20.3, and so forth.
NORM	causes an immediate page eject to occur, and normal pagination to be resumed. In the previous example, the new page would be numbered 21. If .PN FRAC is not in effect, .PN NORM is ignored, and does nothing.
PREF	string specifies a 1- to 8-character string to be used as a prefix in front of all page numbers printed in titles, in tables of contents, or in front of set symbols set with the value of the current page number (&). The string may not contain embedded blanks. To cause the prefix to be omitted from the page number, specify ".PN pref", with no string. This clears the previously defined prefix string.
n	specifies the number of the next page. When the next page eject occurs, either naturally because of the page becoming full, or as a result of .PA, the new page will have the page number specified in n, as though this page eject had been caused by .PA n. If the next page really is started with .PA n, the number given on the .PA control word supersedes the number previously specified with .PN n.

Initial Setting: ON, ARABIC

Notes:

1. The .PN control word can be used to control SCRIPT/VS's page numbering. If the OFF operand is specified, page numbering is discontinued on output, although the page numbers continue to be incremented internally. The OFFNO operand discontinues page numbering on output and stops the internal incrementing of page numbers. When the ON operand is specified, page numbering resumes from the last internal page number.

   The actual page numbers may appear in either arabic numerals, which is the default, or roman numerals, depending upon whether .PN ARABIC or .PN ROMAN was most recent. Changes in the page numbering will take effect on the page after the .PN control was encountered.

2. The .PN OFF and .PN OFFNO control words suppress the default running top title "PAGE &." If you use the .RT [Running Title] control word and include an &, only the page number and not the text is suppressed.

3. If FRAC is specified while the page numbers are represented in ROMAN or ALPHA numerals, the page number that is printed is in lowercase roman or alpha numerals, but the fractional part is in arabic.

4. Table of contents entries generated by .H0 - .H6 [Head Level 0 - 6] or the .PT [Put Table of Contents] control words show the page numbers in the same format they appear on the page, that is, if a prefix is used, it is shown in the table of contents; if roman numbers are in effect, the contents entry has a roman numeral, and so on.

5. Whenever the page number symbol is substituted, its prefix will also be included. Care must be taken therefore when using the page number symbol as a part of an arithmetic operation on the right hand side of a .SE statement.

6. The .PN control word will take effect on the page after it is encountered.

Examples:

- .pn off

  The internal page count continues to be incremented for each page printed.

- .pn offno

  No page numbers appear on SCRIPT/VS output, and the internal page count remains at its current setting without further incrementing.

- .pn on

  Page numbering on SCRIPT/VS output resumes using the current internal page count; this count is incremented for each page printed.

- .pn roman

  The page number in the title at the bottom of the page after this one appears as a roman numeral.

  The control word

  ```
  .pn arabic
  ```

  restores arabic numbering on the next page.

.PP [PARAGRAPH START]

Use the .PP [Paragraph Start] control word to start a new paragraph.

.PP	[line]

where:

line is the text that begins a new paragraph. If line is omitted, the text from the next input line after the .PP control word begins the new paragraph.

Notes:

1. When the .PP control word is encountered, a break occurs, a skip is generated, and the next line of text is indented three character spaces to the right of the current margin. The .PP control word is equivalent to the control words:

   ```
   .sk
   .il +3
   ```

   If these values are not satisfactory for your paragraph formatting, you can redefine the .PP control word as a SCRIPT/VS macro.

Example:

The input lines:

```
.pp This line begins with
a .PP control word.
Here is some more text to show
the formatting.
```

Are formatted as:

```
   This line begins with a .PP control
word. Here is some more text to show
the formatting.
```

.PS [PAGE NUMBER SYMBOL]

The .PS [Page Number Symbol] control word allows you to change the page number symbol used in running top and bottom titles and running headings and footings. The default page number symbol is the ampersand (&) character.

This control word is provided for compatibility with earlier releases of SCRIPT. The same function is provided by the .DC [Define Character] PS control word.

.PS	[c]

where:

c specifies the character to be used as the page number symbol. It may be any character other than a blank. If it is omitted, no character is assigned as the page number symbol.

Initial Setting: Ampersand (&)

Default: Nothing. (No page number symbol.)

Notes:

1. Every occurrence of the page number symbol is replaced with the current page number in running titles, running headings, and running footings, unless .PN OFF or .PN OFFNO is in effect.

2. The .PS control word allows you to change the page number symbol currently in effect. The initial page number symbol is the ampersand (&) character. It may be necessary to change the page number symbol if the & character is not a valid character on your terminal keyboard or the & character is needed as a regular character in your title text.

3. This control word affects all running top and bottom titles and all running headings and footings, including those that have been previously defined. Thus, if a title has been set by the control word:

   ```
   .rt t ///Page &/
   ```

   and later the control word:

   ```
   .ps ?
   ```

   is encountered, the top-title must be reset to:

   ```
   .rt t ///Page ?/
   ```

   Otherwise, the current page number will not be substituted into the title.

4. Do not confuse the page number symbol with the ampersand used on the right-hand side of a .SE [Set Symbol] control word. A single ampersand in a .SE control word always means that the symbol is to be set to the current page number.

   ```
   .se currpage = &
   ```

   sets the symbol 'currpage' to the current page number, regardless of what character, if any, is defined as the page number symbol.

.PT [PUT TABLE OF CONTENTS]

Use the .PT [Put Table of Contents] control word to add lines or control words to the file which is used to generate the automatic table of contents.

The .PT [Put Table of Contents] control word is discussed in "Chapter 6. Head Levels and Table of Contents" on page 73.

.PT	**{ line }**
	{ line }

where:

line is any text line or control word line that you want in the table of contents. This line may be preceded with one or more extra leading blanks (other than the blank that delimits the control word name), and these extra blanks will be removed before the line is written into the table of contents file.

If 'line' is text, it is written to the file DSMUTTOC as part of a .SX [Split Text] control word, which causes it to be formatted as a table of contents line when DSMUTTOC is processed. (See the discussion of the .SX control word.)

If 'line' is a control word, it is written into the DSMUTTOC file directly, and it is executed when the DSMUTTOC file is processed.

If 'line' is specified with extra leading blanks, it is taken as a line of text, even if the first nonblank character is a period. The extra leading blanks are removed, and a .SX control word is built for the DSMUTTOC file, using the first nonblank character as the beginning of the data.

Notes:

1. For text lines, the .PT control word generates a .SX control word to be written into the table of contents utility file in the form:

   ```
   .SX F /text line/ ./33/
   ```

   where the page number used is the actual page number when the .PT is processed, and the delimiter used is actually hexadecimal 00. The .PT control word does not accept lines that begin with hexadecimal 00 as valid lines; such lines result in an error message.

2. This control word is especially useful for defining heading levels with the .DM [Define Macro] control word. The internal macros that process the head level control words .H0 - .H6 [Head Level 0 - 6] use .PT to write the required information into the table of contents file.

3. The .PT control word is ignored while a table of contents is actually being formatted.

Examples:

- .pt .pa

  This line places the .PA control word in the table of contents so that when the table of contents file is being processed, a page eject occurs at this point. You may do this if you want separate content sections to appear on different pages.

- .pt   .pa

  Since the line given has extra leading blanks, it is a text line, not a control word. The leading blanks are removed, and a .SX control word is built, using the characters ".pa" as the data:

  ```
  .sx f /.pa/ ./33/
  ```

  (The head level control word macros insert a leading blank in front of a line to be written to the table of contents with .PT when it is known to be text.)

- .pt .h3 this is a head level 3

  In this case, the control word .h3 is written into the table of contents file because the period appears in the first available position with no extra leading blanks. Any head level that is written into the table of contents file in this way is processed as a heading when the table of contents is actually formatted. A normal head level 3 is generated at that point in the table of contents, but no attempt is made to write any more information into the table of contents utility file. In other words, the .PT function of the macro for .H3 is ignored while the table of contents is actually being formatted.

.QQ [QUICK QUIT]

The .QQ [Quick Quit] control word causes SCRIPT/VS processing to terminate immediately, without the usual final page eject.

.QQ	

Notes:

1. Since SCRIPT/VS does not perform a final page eject after encountering the .QQ control word, some output that has been formatted may never be displayed.
2. The .QQ control word is useful when you are using the .TE [Terminal Input] control word to enter lines from the terminal, and you want to terminate processing quickly.

.QU [QUIT]

The .QU [Quit] control word causes processing to terminate with a final page eject.

.QU	

Notes:

1. The .QU control word causes a final page eject so that the last partial page of formatted text may be printed.
2. The .QU control word will cause termination no matter where or when it is encountered, including within imbedded files (see the .IM control word). All open SCRIPT files are closed before processing terminates.

.RC [REVISION CODE]

The .RC [Revision Code] control word allows you to designate a revision code marker to be printed to the left of the column.

The .RC [Revision Code] control word is discussed in "Marking Updated Material" on page 99.

.RC	**n** [**c** | **ON** | **OFF** | **ON/OFF**]
	*** c**
	Adjust 2|h

where:

n	specifies the revision code number from 1 to 9.
c	specifies the revision code character to be printed along the left margin. It may be any single character, including the blank. If not specified, a blank character is assumed.
ON	signifies the beginning of text to be marked with the code character associated with RC n.
OFF	signifies the end of text to be marked with the code associated with RC n.
ON/OFF	signifies that the next output line should be marked with the RC n code on output.

***** marks the next output line with the specified revision code (like the ON/OFF option). Unlike the ON/OFF option, the * option allows you to specify any character for this one occasion without associating it with a revision code number.

Adjust specifies the amount of horizontal space to be used to contain the revision code character and the space between it and the left column boundary.

h is the amount of horizontal space. If omitted, 2 is assumed. If a value of 0 is specified, or if the amount specified is greater than the space available, no revision codes will be printed.

Notes:

1. The .RC control word has three functions:

   a. To define a revision code symbol,

   b. To activate or deactivate the revision code, and

   c. To set the revision code adjust.

   You may have up to 9 revision codes defined at any time, and each revision code may be assigned a different character. The operands ON and OFF activate and deactivate the actual revision code marking. The operand ON/OFF has the effect of turning ON revision code n for one line only; the line that is next printed after the .RC n ON/OFF is processed.

2. By assigning different symbols to different revision code numbers, including the blank, it is possible to selectively print specific revision code markers or differentiate between various levels of revision.

3. Since the .RC control word does not cause an automatic break, revision code markings may be turned on and off within a paragraph or even a sentence without disrupting normal SCRIPT/VS formatting.

4. The revision code for the leftmost column is placed in the binding that is specified with the BIND command option. The revision code for other columns is placed in the intercolumn gutter. If there is not at least two character spaces of binding or gutter, the revision code is omitted.

5. Revision codes may be nested to a depth of 9. This is useful in circumstances where revisions are made to sections that have already been revised. If a revision code is turned ON while another is ON, the first is stacked. It is neither ON nor OFF. When the inner RC is turned OFF, the stacked RC is turned ON again. Only one RC is ON at a time.

6. If you attempt to redefine a revision code character while that revision code is ON, an error message is issued.

7. The revision code status is subject to the .SA [Save Environment] and .RE [Restore Environment] control words. If you have .RC 3 ON, then .SA, then .RC 3 OFF, then .RE, the status is <u>restored</u> as it was before the .SA (that is, the revision code is turned back on).

8. Revision codes are not applied to running titles, running headings, or running footings, but are applied to skip and space.

Example:

```
.rc 1 1
.rc 2 *
(input)
.rc 1 on
"This writeup applies to version 5 and
version 6."
.rc 1 off
```

The marker for revision code 1 is defined to be a number one (1) and the marker for revision code 2 is defined to be an asterisk (*). All other revision code markers are defined to be blank by default. The line or lines of printout that contain the sentence
1 "This writeup applies to version 5 and
1 version 6." will be noted by a number 1 printed along the left margin.

.RD [READ TERMINAL]

The .RD [Read Terminal] control word allows you to type text at the terminal during SCRIPT/VS output. SCRIPT/VS does not process this text in any way.

.RD	[<u>1</u> / n / STOP]

where:

n specifies the number of lines to be read at the terminal. If omitted, 1 is assumed.

STOP causes the terminal to stop typing anywhere in a line to allow you to type some additional text. SCRIPT/VS will resume typing after you signal ATTENTION.

Default: 1

Notes:

1. The .RD control word is meaningful only when the formatted output is actually being typed at your terminal in interactive environments. The line or lines typed are not processed by SCRIPT/VS, but they appear in the output exactly as they are typed.

2. When the STOP parameter is not specified:

   - The .RD control word causes a break and a section break. All lines read by .RD are read while SCRIPT/VS is in a single column mode. After the .RD is finished, the previous column definition is resumed.

   - If the output is not being typed at a terminal, the .RD control word causes a break and a section break, and then spaces down as many lines as the "n" value specified, but in a single column mode. In this case, .RD acts very much like ".SP n P".

   - As SCRIPT/VS reads lines from the terminal, it accounts for the space they occupy on the page. If the end of a page is reached before the number of lines to be read is exhausted, SCRIPT/VS takes control, performs a page eject to start a new page, and then continues reading the remaining lines.

3. When the STOP parameter is specified, no break occurs. The .RD STOP control word causes continuation of text lines before and after it, if any. SCRIPT/VS considers the .RD STOP to have zero width, and continues formatting lines as though the stop indicator were not there. All text entered when the typewriter terminal stops is unknown to the formatter, and it causes any formatted text to the right of the stop on the same line to be shifted to the right by the width of whatever you enter.

4. If the output is not being typed at a terminal, SCRIPT/VS does not stop in the middle of an output line to accept more text. In this case, the only effect of the .RD STOP control word is to cause continuation of the text surrounding it.

5. When using the STOP parameter under CMS, specify

   ```
   cp term attn off
   ```

   to suppress CP's attention acknowledgment.

6. The .RD [Read Terminal] control word is ignored in the ATMS-III environment.

.RE [RESTORE ENVIRONMENT]

The .RE [Restore Environment] control word restores the status of the SCRIPT/VS variables that were previously saved by the .SA [Save Environment] control word.

.RE	

Notes:

1. The .RE control word restores the status of certain SCRIPT/VS variables from the last-in-first-out stack created by the .SA control word. The .RE control word restores the SCRIPT/VS variables to values that were in effect at the time of the corresponding .SA control word. See the description of the .SA control word for additional information.
2. If there is no currently active .SA control word, the .RE control word restores the initial values. Each .RE control word effectively cancels a corresponding preceding .SA control word.
3. If the formatting mode and/or the text alignment will change as a result of the .RE [Restore Environment] control word, any current line being built will be promoted to the output before the restore is performed.

.RF [RUNNING FOOTING]

Use the .RF [Running Footing] control word to specify that the following lines of text are to be saved as a running footing for subsequent pages.

The .RF [Running Footing] control word is discussed in "Running Headings and Footings" on page 58. Figure 8 on page 56 shows the relationship of the .RF [Running Footing] to the layout of a SCRIPT/VS output page.

.RF	[**<u>ON</u>** / **OFF** / **CANCEL**]
	[**ODD** / **EVEN**] [**CANCEL** / **SUP** / **RES**]

where:

ON identifies the following lines as a running footing to be saved and placed on every subsequent page. ON is the default.

OFF indicates that the definition of the running footing is ended.

ODD specifies that the following lines are to be saved as the running footing for odd-numbered pages only.

EVEN specifies that the following lines are to be saved as the running footing for even-numbered pages only.

CANCEL may be used with the ODD or EVEN parameter, or by itself to cancel running footings defined with the ODD, EVEN, or ON parameters.

SUP may be used with the ODD or EVEN parameter, or by itself to suppress a running footing definition. A suppressed definition still exists, but it is not printed on any page until restored. A suppressed footing definition is deleted if a new one is defined with .RF ODD, EVEN, or ON. A new footing is not suppressed by a previous .RF SUP. To suppress the new definition, you must issue another .RF SUP after the definition is complete.

RES restores a running footing that was previously suppressed with the SUP parameter.

Default: ON

Notes:

1. The running footing will be placed on the page immediately above the space which is defined by the .BM [Bottom Margin] control word. It is formatted using the initial formatting environment, and all page number symbols are replaced by the current page number.

2. The running footing defined with the .RF control word will take effect on the page after the .RF control word is encountered.

3. If you start another running heading or footing definition within a footing, or use any control word which is not allowed in a running footing, the definition is terminated. The disallowed control words are the same as those that are not allowed in a keep, and are described in Figure 27 on page 355.

4. Some control words are processed once when the running footing is encountered, and others are processed every page as the footing is being formatted to be put on the page. A list of the control words that are processed immediately is given in Figure 28 on page 355.

5. The .RF control word parameters CANCEL, SUP, and RES do nothing if there is no running footing definition to cancel, suppress, or restore.

.RH [RUNNING HEADING]

Use the .RH [Running Heading] control word to specify that the following lines of text are to be saved as a running heading for subsequent pages.

The .RH [Running Heading] control word is discussed in "Running Headings and Footings" on page 58. Figure 8 on page 56 shows the relationship of the .RH [Running Heading] to the layout of a SCRIPT/VS output page.

```
.RH     [ ON     ]
        [ OFF    ]
        [ CANCEL ]
        ---------------------------
        [ ODD  ] [ CANCEL ]
        [ EVEN ] [ SUP    ]
                 [ RES    ]
```

where:

ON identifies the following lines as a running heading to be saved and placed on every subsequent page. ON is the default.

OFF indicates that the definition of the running heading is ended.

ODD specifies that the following lines are to be saved as the running heading for odd-numbered pages only.

EVEN specifies that the following lines are to be saved as the running heading for even-numbered pages only.

CANCEL may be used with the ODD or EVEN parameter, or by itself to cancel running headings defined with the ODD, EVEN, or ON options.

SUP may be used with the ODD or EVEN parameter, or by itself to suppress a running heading definition. A suppressed definition still exists, but it is not printed on any page until restored. A suppressed heading definition is deleted if a new one is defined with .RH ODD, EVEN, or ON. A new heading is not suppressed by a previous .RH SUP. To suppress the new definition, you must issue another .RH SUP after the definition is complete.

RES restores a running heading that was previously suppressed with the SUP parameter.

Default: ON

Notes:

1. The running heading will be placed on the page immediately below the space which is defined by the .TM [Top Margin] control word. It is formatted using the initial formatting environment, and all page number symbols are replaced by the current page number.
2. The running heading defined with the .RH control word will take effect on the page after the .RH control word is encountered.
3. If you start another running heading or footing definition within a heading, or use any control word which is not allowed in a running heading, the definition is terminated. The disallowed control words are the same as those that are not allowed in a keep, and are described in Figure 27 on page 355.
4. Some control words are processed once when the running heading is encountered, and others are processed every page as the heading is being formatted to be put on the page. A list of the control words that are processed immediately is given in Figure 28 on page 355.
5. The .RH control word parameters CANCEL, SUP, and RES do nothing if there is no running heading definition to cancel, suppress, or restore.
6. The function of the SCRIPT/370 Version 3 control word .HN [Headnote] is provided by the .RH control word. Thus, you cannot have a SCRIPT/370 headnote and a SCRIPT/VS running heading at the same time.

.RI [RIGHT ADJUST]

Use the .RI [Right Adjust] control word to position an output line flush with the right margin.

The .RI [Right Adjust] control word is discussed in "Positioning Lines on the Page" on page 44.

```
.RI   [ 1    ]
      [ n    ]
      [ ON   ]
      [ OFF  ]
      [ line ]
```

where:

n specifies the number of input lines to be right adjusted. If omitted, 1 is assumed. If .RI n is specified when .RI ON is in effect, right adjusting is turned off when n lines have been right adjusted, or when .RI OFF is encountered.

ON specifies that subsequent text lines are to be right adjusted.

OFF terminates right adjust mode if it was ON, or if n has been specified and has not been exhausted.

line is a line of text to be right adjusted. The line is considered to start with the first nonblank character after the .RI control word.

Notes:

1. The keyword ON or OFF, or the number of input lines to be right adjusted (n), must be the only parameter on the control word line. A string of words that happens to start with one of these is interpreted as a single line to be right adjusted. For example, the control word lines:

   ```
   .ri on top of old smokey
   .ri 555 Bailey Ave.
   ```

   are taken to be of the ".RI line" form, not requests for large numbers of lines to be right adjusted.
2. When right adjusting is in effect, no formatting is done on the line. That is, the line is right adjusted as it stands, and it is not filled from other input lines or justified. If a tab character

appears in the line to be right adjusted, the tab is resolved before the line is right adjusted.

3. This control word acts as a break.

4. If the line to be right adjusted is longer than the current column length, the excess words are right adjusted on a separate output line.

5. The .CE [Center] control word is a variant of .RI. If either of these control words is processed, the other is cancelled.

6. Contrast this control word with .FO RIGHT. The latter allows lines to be formatted by concatenating words until the line is nearly full, but then the filled line is right adjusted instead of being justified, as would be the case with .FO ON.

Example:

```
.ri 3

                           These three lines are
                               right-adjusted,
                               as you can see.
```

.RN [REFERENCE NUMBERS]

Use the .RN [Reference Numbers] control word to control output line reference numbering.

.RN	[**ON** / **OFF**]

where:

ON starts reference numbering.

OFF stops reference numbering.

Initial Setting: OFF

Default: ON

Notes:

1. This control word takes effect on the current page.

2. Lines will be numbered in increments of 1 starting at 1. Numbers will be placed about half an inch to the right of the rightmost column.

3. Only nonblank lines on the body of the page will be numbered. Lines in the running heading and footings, running titles, and footnotes are not numbered.

.RT [RUNNING TITLE]

The .RT [Running Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the top or bottom of the next page and each subsequent output page.

The .RT [Running Title] control word is discussed in "Top and Bottom Running Titles" on page 61. Figure 8 on page 56 shows the relationship of the .RT [Running Title] to the layout of a SCRIPT/VS output page.

```
.RT  | [ Top    ]  [ ALL  ]  [ 1 ]  /part1/part2/part3/
     | [ Bottom ]  [ Odd  ]  [ n ]
     |             [ Even ]
```

where:

Top	specifies that this control word refers to top titles. The TOP keyword may be abbreviated as T. This is the default.
Bottom	specifies that this control word refers to bottom titles. The BOTTOM keyword may be abbreviated as B.
Odd	specifies that the title being defined is to be printed on odd numbered pages only. The ODD keyword may be abbreviated as O.
Even	specifies that the title which is being defined is to be printed on even-numbered pages only. If neither ODD nor EVEN is specified, the title being defined will be printed on both even- and odd-numbered pages. The EVEN keyword may be abbreviated as E.
ALL	specifies that the title is to be printed on both odd- and even-numbered pages. ALL is the default.
n	is the number of the title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "top title 1" sets the same storage buffer as "bottom title 6." See the discussion of the .HS [Heading Space] and .FS [Footing Space] control words for information on how to allocate space on your output page for top and bottom titles.
part1	is the portion of the title to be left justified.
part2	is the portion of the title to be centered between the left and right margins.
part3	is the portion of the title to be right justified.
/	is any delimiter character that does not appear in part1, part2, or part3.

Initial Setting: TOP ALL 1 ///PAGE &

Notes:

1. Every occurrence of the page number symbol in part1, part2, and part3 is replaced with the current page number on each page where a title appears, unless .PN OFF or .PN OFFNO is in effect. The character designated as the page number symbol may be changed with the .PS [Page Number Symbol] control word or the .DC PS [Define Character] control word.

2. Symbol substitution and character translations set up by the .TR [Translate Character] control word are done on part1, part2, and part3 when the .RT control word is processed, not on every page.

3. The three parts of the title are used to form the actual title that is to be saved for future use, and no part may be more than 120 characters in length. This title may be printed at the top or bottom of each subsequent output page, if space has been allocated for it using the .HS or .FS control words.

4. The specific location of the top titles on the page is controlled by the .TM [Top Margin] and .HM [Heading Margin] control words; the number of top titles to be used on each page is controlled by the .HS [Heading Space] control word.

5. The specific location of the bottom titles on the page is controlled by the .BM [Bottom Margin] and .FM

[Footing Margin] control words; the number of bottom titles to be used on each page is controlled by the .FS [Footing Space] control word.

6. Any title may be changed by including another .RT control word later in the file.

7. The default top title, printed on each page of output after page one, is

```
    PAGE &
```

   which is right-justified at the top of the page. This title may be suppressed with the .PN OFF control word.

8. This control word will take effect on the page after it is encountered.

9. The length of the title will be that of the line length as set by the .LL control word.

10. The parameters may be specified in any order, and if contradictory options are specified, only the latest one will be used. The first character that is not recognized as an option will be taken as a delimiter.

Example:

```
 .rt t 'heading''PAGE &'
```

The heading and the current page number will be printed at the top of all subsequent pages, unless the heading space has been set to zero.

.RV [READ VARIABLE]

The .RV [Read Variable] control word is similar to the .SE [Set Symbol] control word, except that the value of the symbol is read from the terminal.

.RV	**symname [[=] ']**

where:

symname is the name of the symbol to be set. It may be any name that would be allowable on the left side of the equal sign in a .SE [Set Symbol] control word.

[=] ' indicates that the value set into the named symbol is to be treated as a quoted string. If you do not specify the single quotation mark, SCRIPT/VS provides the equal sign automatically, and processes whatever string is entered according to the rules for the value on the right-hand side of the equal sign in .SE [Set Symbol] control words. In this case, any value that requires single quotation marks must have the quotation marks explicitly supplied as part of the value entered from the terminal.

Notes:

1. When the .RV control word is encountered, a line is read from your terminal. This line is used as the right-hand side of the equal sign to set the value of the symbol named in the .RV control word. Any expression that would be allowable as the value in a .SE control word is allowable here. If no name is given on the .RV control word, it is ignored, and no line is read from the terminal.

2. The .RV control word does not cause an automatic break.

3. No message is displayed before the terminal is unlocked to accept the input line. You may use the .TY [Type on Terminal] control word to issue a prompting message before the .RV control word issues its terminal read.

4. The .RV [Read Variable] control word is not supported in the CICS/ATMS-III environment, and will result in a null value. In batch environments, .RV will be ignored unless the file DSMTERMI can be read.

Example:

A symbol called "name" could be set with the following control word:

```
 .se name = 'John Doe'
```

The same symbol could also be set this way:

```
 .rv name = '
```

At this point, SCRIPT/VS issues a read to your terminal, and you may enter the material to be used as the value of the symbol. In this example, you would enter:

```
 John Doe
```

You must use single quotation marks in the same circumstances where they would be required in a .SE control word, unless the .rv name =' form is used.

.SA [SAVE ENVIRONMENT]

The .SA [Save Environment] control word saves the SCRIPT/VS formatting environment, which consists of the values and dimensions of certain control words. If any of these control words is processed, the environment is changed accordingly. The .RE [Restore Environment] control word restores all the environment values to the settings that were in effect before the .SA control word was issued.

.SA	

Notes:

1. The .SA control word saves environments in a stack. The .RE [Restore Environment] control word restores the SCRIPT/VS environment to the values that were in effect at the time of the most recent .SA control word.

2. The .SA control word only saves a copy of the values of these SCRIPT/VS variables, it does not change any of these variables.

3. Since .SA does not change any of the SCRIPT/VS variable settings, all variables should be explicitly set to the values appropriate unless the current settings are known. For example, you can explicitly set indention to 0, and then restore it to whatever it was previously.

4. The control word values in the saved environment are listed in Figure 34 on page 360.

5. The environment saved by .SA is divided into three parts, the "active environment," "page control," and translate tables. The active environment is automatically saved and restored for some keeps (see the discussion of the .KP [Keep] control word) and for footnotes. It is not necessary to use .SA and .RE within keeps and footnotes unless you want to save and restore values that are not in the active environment, such as .TR [Translate Character] specifications.

.SC [SINGLE COLUMN MODE]

The .SC [Single Column Mode] control word saves the current column definition and starts a temporary single column format. The .MC [Multicolumn Mode] control word restores the column definition that was saved by .SC.

.SC	

Notes:

1. The .SC [Single Column Mode] control word temporarily starts formatting in a single column that is the same width as the current .LL [Line Length] specification. The .MC [Multicolumn Mode] control word restores the column definition that was in effect before the .SC was processed.

2. More than one .SC control word may be processed without an intervening .MC. Each .MC clears one .SC, and, until the first .SC in the list is cleared, the column definition restored by each .MC is a single column definition that was set up by an earlier .SC.

3. The .CD [Column Definition] control word starts an entirely new column definition, and clears all .SC's and .MC's that may be in effect.

4. This control word is not allowed in a keep.

5. The .SC control word starts a new section. Therefore, skips inserted by the .SK [Skip] control word are discarded, since they would appear at the top of a column.

.SE [SET SYMBOL]

The .SE [Set Symbol] control word allows you to define and assign values to symbols or arrays of symbols. Using the .SE control word, you can give a symbolic name to a page number, a word, or even a string of SCRIPT/VS control words. The .SE control word itself, or any of its parameters, can be a symbol.

The .SE [Set Symbol] control word is discussed in "Chapter 12. Symbols in Your Document" on page 129.

.SE	**symname [([n])] 'qstring[']**
	symname [([n])] = [**symvalue** **&** **SUBSTR string [start [length]]** **INDEX string1 [string2]**]
	symname [([n])] OFF
	symname LIB

where:

symname	is the name to which you want to assign a symbolic value to be substituted during SCRIPT/VS processing. It may contain a maximum of 10 non-blank characters which may be upper- and lowercase alphabetic, numeric, and the characters @, #, and $.
n	You may specify a line number in parentheses for array symbols, except when you use the LIB parameter. An array line number is also called an element number or a subscript.
qstring	may be any string preceded by a single quotation mark. If the string has trailing blanks, the string may be terminated by a quotation mark. The string may itself contain quotation marks, which need not be doubled as in the 'symvalue' form below.
symvalue	assigns a value to the symbol name; it may be a character string or an arithmetic expression.
&	assigns the symbol name a value equal to the current page number string.
SUBSTR	obtains the specified characters (substring) from a given string and assigns them to the symbol provided.
string	is the string from which the substring is to be extracted.
start	is a positive integer that defines the position of the beginning character in the string which is to be assigned to the symbol. If both start and length values are omitted, the symbol will be assigned the entire value of the string.
length	specifies the number of characters to be extracted from the string, in other words, the length of the substring. If length is omitted, the remainder of the string, from the specified start to the end, will be assigned to the symbol.
INDEX	searches the string "string1" to see if it contains the string "string2". If it does, the symbol value is set to the position of the starting character of "string2" within "string1". If it does not, the symbol value is set to "0".
string1	is a string that is to be searched to see if it contains the string "string2".

string2 is a string that is to be searched for in the string "string1". If string2 is omitted, or has a null value, the symbol will be set to 0.

OFF unsets the named symbol so that, to SCRIPT/VS, it was never set. An entire array symbol will be set off if no subscript is provided. However, only the specified element will be set off if a subscript is provided.

LIB causes the symbol to be set by retrieving its value from a library. The name of this library may be defined using the LIB option on the SCRIPT command. If the LIB option is used to set a symbol, the value retrieved from the library will replace the current value. If no entry with the symbol name given exists in the library, the symbol will be undefined. Since symbol names in the library are in uppercase only, the same member of the library will be used to define all symbols of the same name that differ only in the case of the characters used. Subscripted symbol names may not be used with the LIB option. The LIB option may be used regardless of the most recent specification of the .LY control word.

Symbol Names:

The character '*' has a special meaning as the first character of a symbol name because it denotes that the symbol is a local symbol. This means that it will only have the value that is set at the current level of macro nesting, and every macro that is called has its own set of local symbols for the duration of that macro's execution.

During SCRIPT/VS processing, a symbol name is recognized when it is preceded by an ampersand (&) and followed by a blank or a period:

```
&symname
```

If the symbol name appears in any of the following forms:

```
symname()
symname(n)
symname(&symbol)
```

it is an array symbol.

SCRIPT/VS also recognizes symbol names that are preceded by the GML delimiter, initially and by default, the colon (:). The name of a symbol defined with .SE as a GML tag must be all in uppercase characters, and it cannot be an array symbol. The GML delimiter causes SCRIPT/VS to search for a symbol name that is all uppercase, regardless of the case of the name in the input line.

Symbol Values:

If a symbol value is set to a character string that contains any embedded blanks or any special characters, it must be enclosed in single quotation marks. For example,

```
.se dog = cat
.se end = '.qu'
.se sentence = 'This is a sentence.'
```

are all valid character strings. If you want a character string to contain a single quotation mark ('), you must enter two of them, for example

```
.se title = 'Mrs. O''Grady''s Cat'
```

unless you use the form:

```
.se title 'Mrs. O'Grady's Cat'
```

If you want to use the INDEX or SUBSTR parameter of .SE to operate on a portion of the string "Mrs. O'Grady's Cat," which is the value of the symbol &title, you should turn substitution OFF with the .SU [Substitute Symbol] control word before issuing the .SE control word. If substitution is ON, the control word line

```
.se syma = index &title Cat
```

does not work properly, because substitution is performed on the line before the .SE control word processes it. The substituted line already has the string "Mrs. O'Grady's Cat" on the right side of the equal sign, and the .SE control word misinterprets the internal blanks as delimiters between parameters. If substitution is OFF, the .SE control word receives the line in its unsubstituted form, and the parameter on the right side of the equal sign is the character string "&title". Even though substitution is OFF, the .SE control word can retrieve symbol values when it recognizes symbol names on the right side of the equal sign. In this case, the .SE control word knows that the entire value of the symbol &title constitutes the "string1" parameter for this .SE control word.

If the symbol value is an arithmetic expression, it must be in the form:

[op1] n [op2 n op2 n op2 n...]

where:

op1 is a unary + or - sign.

op2 is an arithmetic operator:

- + addition
- - subtraction
- * multiplication
- / division

n is a valid integer of length less than 9 digits. Lengths greater than this may produce unpredictable results. The integers may have been assigned their values as a result of a symbol substitution (including the page number symbol).

For example,

```
.se nextpage = & + 1
.se current = -100
.se addit = &current + 25
.se answer = 15 - 42
```

are all valid arithmetic expressions.

Notes:

1. In symbol names, uppercase and lowercase letters are considered to be different, thus the symbols symbol1, Symbol1, and SYMBOL1 are three distinct symbols. Symbols whose names start with the dollar sign ($) are system symbols, and they exist only in an uppercase form. The reserved system symbols which may not be set by the user, such as &$RET, are in this category. Although you can set a symbol whose name starts with $ if that name is not in use as a read-only system symbol, this is discouraged; confusion can occur due to the name folding. Symbols preceded by the GML delimiter can be recognized only if a symbol (GML tag) has been defined with the specified name all in uppercase. During substitution, a symbol name that begins with $ or a symbol preceded by the GML delimiter is folded to uppercase before being resolved.

2. An iterative substitution, as described in the .SU [Substitute Symbol] control word discussion, is automatically performed on all character string symbol values.

3. If the symbol value is omitted, the symbol's value is set to a null character string (length zero).

4. The symbol for the current page number, &, remains the same even if the page number symbol that is used in running titles and running headings and footings is changed with the .PS [Page Number Symbol] or .DC PS [Define Character] control word.

5. If you set a symbol name equal to the current page number (.SE refer = &) within a keep or float, the symbol is actually set twice. The page where the keep will finally be located is not known until the keep is ended and measured. When a .SE control word sets a symbol to the current page number, the symbol is set immediately, and then it is set again when the page number of the keep is known. If you refer to this symbol before the second setting, the number may be inaccurate.

6. Arithmetic expressions in set statements are evaluated strictly from left to right, and no operator takes precedence over another. For example, the expression:

   ```
   .se x = 1 + 2 * 4 + 6
   ```

   will set the symbol x to the value 18.

7. See the discussion of the .SU [Substitute Symbol] control word for more information about symbol substitution.

8. The LIB option of the .SE control word allows a symbol to be explicitly retrieved from the library. The .LY [Library] control word allows a symbol value to be retrieved from the library when it is <u>used</u> in a document and when a value for it does not currently exist. When a symbol value is once retrieved from the library, it is stored in the symbol table for future use.

9. You should be careful when using local symbols and page number symbols in arithmetic set statements with the multiplication operand (*). The expression .se a = &*3+1 will be taken as a request to add 1 to the value of the local symbol &*3, <u>not</u> as a request to multiply the page number by 3 and then add 1. To achieve the latter effect, the page number symbol must be delimited (.se a = &.*3+1).

10. When symbol substitution is ON, a .SE control word line is completely substituted before it is processed. When substitution is OFF, the .SE control word can still perform individual substitutions on symbolic values in the control word. See the next two notes for examples.

11. Be careful of the effects of substitution on arithmetic set statements when symbols that contain negative numbers are used. For example,

    ```
    .se a = -3
    ```

```
.se b = 5+&a
```

will result in an invalid expression if substitution is on, as the line will be substituted as:

```
.se b = 5+-3
```

which is invalid. However, if substitution is OFF, the .SE control word processor can see that you want to add 5 to -3, and can do it correctly.

12. The string assignment form of the .SE control word:

    ```
    se a 'string'
    ```

    is not substituted when substitution is turned off.

13. Substitution also has an effect on the .SE control word if strings are to be set which are longer than 16 characters. SCRIPT/VS will treat a single character string without special characters or blanks as a character string even if it is not enclosed in quotation marks, if it is not more than 16 characters long. If the string is longer than this, an error will result. For example:

    ```
    .se a = '12345678901234567890'
    .se b = index &a 1
    ```

    will result in an error if substitution is on because the symbol &a is not enclosed in quotation marks. The error will not happen if substitution is off.

.SK [SKIP]

Use the .SK [Skip] control word to generate blank vertical space before the next text output line, except at the top of a column or page.

The .SK [Skip] control word is discussed in "Vertical Space" on page 43.

.SK	$\left[\begin{matrix}\underline{1}\\ v\end{matrix}\right]$ **[C]** **[A]** **[P]**

where:

v is the amount of space to be inserted in the output. If no number is given, 1 line is assumed. If the size in "v" is not qualified as any of the other space units (inches, picas, ciceros, or millimeters), it is a request to skip a number of lines. In this case, unless A is specified, the size of the request is multiplied by the appropriate factor if double spacing or multiple spacing is in effect.

C indicates conditional skips. These skips depend upon what follows them in the output column. If conditional skips are followed by a line of text, they appear in the column as requested. If they are followed by another skip or space request, the two skip or space requests are compared, and only the larger of the two remains in the column.

A indicates absolute skips. If the vertical size of the skip given in "v" is expressed in inches, picas, ciceros, or millimeters, it is already an absolute number, and the actual requested depth will be skipped, to the closest approximation possible on the current logical device. In this case, A need not be specified.

P indicates page skips. These skips will generate skip space across the full width of the page, even when formatting in multiple columns. Since this type of skip causes a section break, it is not allowed in a keep.

Default: 1

Notes:

1. No blank space is generated if it would be the first to be printed at the top of a column of output. The top of a column may be at the top of the page or after a section break. This may result in columns being set short by the amount of discarded blank space.

   If the blank space would not fall at the top of a column, the .SK control word is identical to the .SP [Space] control word.

   If the column is partially filled, and a .SK control word is encountered requesting more space than remains in the column, only enough space to fill the column is generated, and then all the rest (at the top of the next column) is ignored.

2. Page skips (the P parameter) are ignored if they fall at the top of a page, but not if they fall elsewhere on the page.

3. If double spacing is in effect, the number of skips generated is multiplied by the line spacing amount, unless absolute spacing is specified.

4. This control word acts as a break.

5. If the skip request is in lines (unqualified space units), the size of each line is as defined with the .SL [Set Line Space] control word.

6. Conditional skip processing is not performed across the boundaries of keeps, floats, footnotes, running headings and running footings.

.SL [SET LINE SPACE]

This control defines the vertical distance from the baseline of the current line to the baseline of the following line.

.SL	[vsize]

where:

vsize is the the vertical size of all following formatted output lines until redefined with another .SL.

Initial Setting: One logical device print line.

Default: One print line.

Notes:

1. The vertical size of formatted output lines is set by .SL to the nearest approximation of the requested size that is possible on the current logical device.

2. The .SL value is used for formatted lines and for requests in lines for the following control words:

   .CC [Conditional Column Begin]
   .CP [Conditional Page Eject]
   .SK [Skip]
   .SP [Space]

   In all other control words that can have a vertical dimension expressed in lines, such as .PL [Page Length] and .TM [Top Margin], the size of the request is based on the size of a print line on the current logical device. For example, if the logical device were an 8 line per inch printer, the control word .TM 4 would set the top margin to 4 lines, or one-half inch, regardless of the .SL value.

3. Additional space is placed above each line, and is discarded if the line falls at the top of a column.

4. The "A" parameter of the .SP [Space] and .SK [Skip] control words does not affect spacing set with .SL.

.SO [STAIRS/VS OUTPUT]

Use the .SO [STAIRS/VS Output] control word to set the STAIRS paragraph code. The .SO control word will only take effect when output is being produced for STAIRS.

The .SO [STAIRS/VS Output] control word is discussed in "Chapter 17. Producing Input for STAIRS/VS" on page 179.

```
.SO    { DOC name      }
       { PID nxx       }
       { OPR number    }
       { RPW password  }
       { DPW password  }
```

where:

DOC specifies a document name to be placed in the STAIRS/VS CTF blocks. The name may be up to 12 characters.

PID specifies the identification to be given to the next paragraph. Blocks are numbered in increments of 1 starting with the low order position. The first position must be numeric (0-9). The second and third positions are alphameric in the ascending order: blank (X'40'), A-Z, 0-9. Lowercase alphabetic characters will be folded to uppercase. A blank may be used in the second position only if the third position is blank.

OPR specifies an operator number to be placed in the STAIRS/VS CTF blocks. The number may be any number up to 32,767.

RPW specifies a read password to be placed in the STAIRS/VS CTF blocks. The password may be up to five characters, and the default is blanks.

DPW specifies a delete password to be placed in the STAIRS/VS CTF blocks. The password may be up to five characters, and the default is the read password.

Notes:

1. If the .SO [STAIRS/VS Output] control is omitted, blocks will be numbered starting with 0 followed by two blanks.
2. When concatenation is off, each line will have a separate identification code.
3. The .SO control word causes a break, and the DOC parameter starts a new page.

Examples:

- Some valid uses of the .SO [STAIRS/VS Output] control word are:

```
.so 34c
.so 0b
.so 20b
```

- Some examples of the numbering sequence are:

```
0      0AY    0A8
0A     0AZ    0A9
0AA    0A0    0B
0AB    0A1    0BA
0AC    0A2    0BB
 .      .      .
 .      .      .
```

.SP [SPACE]

Use the .SP [Space] control word to generate blank vertical space before the next text output line.

The .SP [Space] control word is discussed in "Vertical Space" on page 43.

.SP	$\begin{bmatrix}\underline{1}\\ v\end{bmatrix}$ [C] [A] [P]

where:

v is the amount of space to be inserted in the output. If no number is given, 1 line is assumed. If the size in "v" is not qualified as any of the other space units (inches, picas, ciceros, or millimeters), it is a request to space a number of lines. In this case, the size of the request is multiplied by the appropriate factor if double spacing or multiple spacing is in effect, unless A is specified.

C indicates conditional spaces. These spaces depend upon what follows them in the output column. If conditional spaces are followed by a line of text, they appear in the column as requested. If they are followed by another skip or space request, the two skip or space requests are compared, and only the larger of the two remains in the column.

A indicates absolute spaces. If the vertical size of the space given in "v" is expressed in inches, picas, ciceros, or millimeters, it is already an absolute number, and the actual requested depth will be spaced, to the closest approximation possible on the current logical device. In this case, A need not be specified.

P indicates page spaces. These spaces will generate space across the full width of the page, even when formatting in multiple columns. Since this type of space causes a section break, it is not allowed in a keep.

Default: 1

Notes:

1. If double spacing is in effect, the number of spaces generated is multiplied by the line spacing amount, unless absolute spacing is specified.
2. This control word acts as a break.
3. If the space request is in lines (unqualified space units), the size of each line is as defined with the .SL [Set Line Space] control word.
4. If the space request exceeds the remaining column depth, the space is placed at the top of the next column. If the space request exceeds the maximum column depth, the excess space is discarded.
5. Spacing via .SP for large amounts of space may produce undesirable results if column balancing is in effect. This is because the space will not be split across columns.

.SS [SINGLE SPACE MODE]

Use the .SS [Single Space Mode] control word to cancel a previous .DS [Double Space Mode] or .LS [Line Spacing] control word, and to resume single-spacing of output.

.SS	

Notes:

1. This control word does not cause a break.

2. Output following the .SS [Single Space Mode] control word is single spaced. Since this is the normal output format, .SS is needed only to cancel a previous .DS [Double Space Mode] or .LS [Line Spacing] control word.

.SU [SUBSTITUTE SYMBOL]

Use the .SU [Substitute Symbol] control word to cause SCRIPT/VS to stop substitution of defined set symbols or to restore substitution.

.SU	[**1** / **n** / **ON** / **OFF** / **line**]

where:

n specifies the number of lines to be scanned for set symbols to be substituted. If omitted, 1 is assumed.

ON turns on an open-ended substitution mode. ON is the initial setting.

OFF turns off substitution mode if it was ON, or if n was given and is not yet exhausted.

line is a line containing symbols that you want SCRIPT/VS to substitute with values previously set. Symbols may be set via the .SE, .RV, .IM, or .AP control words, or by a macro call.

Initial Setting: ON

Default: 1

Notes:

1. The .SU control word causes a specified number of the following input lines, control words as well as text, to be scanned for defined set symbols. If the argument ON is in effect, every line up to a subsequent .SU OFF will be scanned. Substitution ON is the initial mode of operation, but it is reset to OFF with .SU OFF or with .SU n, after n lines have been read. Use of ".SU line" will not cause resetting of the count of the number of lines to be substituted.

2. When an input line is substituted, each complex symbol may go through several stages of substitution until no further substitution can be done. Any "symbol name" for which no definition exists is left in the input line as text.

3. The substitution of set symbols may increase or decrease the length of the text line. If the line's length reduces to zero, it becomes a "null line."

4. The TWOPASS option of the SCRIPT command may result in defining symbols during the first pass that can be substituted during the second, even though these symbols are defined physically later in the SCRIPT file. If the length of the symbol value and the length of the symbol name are grossly different, the formatting may come out slightly differently in the two passes.

.SV [SPELLING VERIFICATION]

Use the .SV [Spelling Verification] control word to cause spelling checking to start and stop. This control word must be enabled by the SPELLCHK option of the SCRIPT command. If the SPELLCHK option is not in effect, the .SV control word is ignored.

The .SV [Spelling Verification] control word is discussed in "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.

.SV	[<u>ON</u> OFF]
	[NOADD] [NOSTEM] [NUM]

<u>where:</u>

ON	specifies that spelling verification is to be started. Any addenda dictionary will be searched before the user and main dictionaries, and stem processing will be performed, but numbers will not be checked.
NOADD	The addenda dictionary will not be searched, stem processing will be performed, and words that contain numeric characters will not be checked. Turns on verification if it was off and inhibits use of the addenda dictionary for spelling checking. This dictionary is created using the .DU [Dictionary Update] control word. If NOADD is specified, only the main dictionary will be used for word verification.
NOSTEM	turns on verification if it was off and stops the spelling checking function from performing stem processing on words to be verified. Stem processing is described in more detail in "Chapter 16. Automatic Hyphenation and Spelling Verification" on page 171.
NUM	turns on verification if it was off and indicates that spelling verification is to be started for words which contain numeric as well as alphabetic characters. This option allows text that contains numbers to be verified. If ON instead of NUM is specified, words that contain alphabetic characters only will be checked.
OFF	stops spelling checking.

Initial Setting: ON

Default: ON

<u>Notes:</u>

1. Each time the .SV control word is used, the settings that control spelling verification are all reset. For example:

   ```
   .sv noadd
   ```

   will stop spelling checking against the addenda dictionary. If this is followed later in the document by:

   ```
   .sv num
   ```

   spelling verification will now start for numbers, and will be <u>resumed</u> from the addenda dictionary.

.SX [SPLIT TEXT]

The .SX [Split Text] control word is used to split a string of text between the left and right column margins, with a filler in between the two.

The .SX [Split Text] control word is described in "Positioning Lines on the Page" on page 44.

```
.SX   [ F ]  /lpart/fill/rpart[/]
      [ C ]
```

where:

F allows the left part of a split line to be folded if it will not fit in the column. The folding is done according to rules appropriate for creating table of contents entries. The fill string and the right part are never folded; they must fit in the column.

C specifies that the middle part of the string, usually the fill string, is to be centered, and not replicated.

/ is any delimiter character. The first nonblank character will be taken as the delimiter character.

lpart is the string to be placed against the current left margin.

fill is a string of up to eight characters to be used to fill the space between lpart and rpart. If no fill is specified, blanks are used. The beginning position of the fill string is a multiple of its length from the left margin. Multiples of the fill string are inserted until they would overlap the right part. If the fill string is longer than the space between the strings, it is not used. The fill string may not contain tabs or backspaces.

If the "C" parameter was specified, the fill string is centered between the left end of the left part and the right end of the right part, and not replicated. When the "C" parameter is used, the fill string is not limited to eight characters, but may not overlap either the left or right parts.

rpart is the string to be placed against the current right margin.

Notes:

1. The delimiter character between the strings may be any unique character that does not occur within the strings themselves.
2. Any of the three parts of the line may be null.
3. The final split line is printed in the font that is in effect when the .SX control word is encountered.
4. This control word causes a break.
5. The .PT [Put Table of Contents] control word writes .SX control words into the the table of contents file to be processed when the table of contents is formatted. The delimiter used for these internally generated .SX control words is hexadecimal 00.

Examples:

- Split text with null fill string:

```
.sx /left part//right part/
```

```
left part                   right part
```

- A foldable split text, as used in tables of contents:

```
.of 1
.sx f /An example...ne/ ./323/
```

```
An example of a folded split
 text line  . . . . . . . .   323
```

- Split text with null left and right parts:

```
.sx //-+//
```

```
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

.SY [SYSTEM COMMAND]

The .SY [System Command] control word is only supported in the interactive environments of CMS and TSO. In CMS, SCRIPT/VS passes a line to CMS for processing as a CMS or CP command line. In TSO, the line is stacked until the end of SCRIPT/VS processing.

.SY	**line**

where:

line is a CMS, CP, or TSO command line. In CMS, if line is omitted, CMS subset is entered.

Notes:

1. Use the .SY control word if you want to perform some CMS or CP command when your SCRIPT file is processed, or, in TSO, if you want some command to be performed after formatting is complete.

2. The .SY control word does not cause a break.

3. No CMS command or user program is allowed that requires the use of the same area of storage that is being used by SCRIPT/VS. CMS commands that are valid in CMS SUBSET are valid on the .SY command. An invalid SUBSET command results in a return code of -2.

4. To test whether a command executed successfully in a SCRIPT file, you can use the .IF control word to test the value of the reserved symbol &$RET. For example:

   ```
   .if &$ret ne 0 .qu
   ```

   causes SCRIPT/VS to terminate processing if the return code from the last executed CMS command is not zero.

5. If the command does not exist or was not executed at all, &$RET is set to a negative value. This would be the case for nonexistent commands in CMS, and for all commands in environments other than CMS.

6. In the TSO environment, if the .SY control word is used more than once, the commands will be executed in the order in which they were encountered.

7. No command or program should be used that will free the storage controlled by SCRIPT/VS (for example, XEDIT in CMS).

Example:

The .IM [Imbed] control word issues an error message if the designated file is not found. The CONTINUE option of the SCRIPT command allows SCRIPT/VS to continue processing after this error. The following .CIM [conditional imbed] macro would allow SCRIPT/VS to test for the existence of a file in CMS before attempting to imbed it, and only imbed it if it is available:

```
.dm cim() /.sy state &*1 * *
.dm cim() /.if &$ret eq 0 .im &*
```

The macro would be invoked as follows:

```
.cim filename
```

.TB [TAB SETTING]

Use the .TB [Tab Setting] control word to define how tab characters (hexadecimal 05) are to be resolved. They may be changed into a number of blanks or to a string of another character.

The .TB [Tab Setting] control word is discussed in "Using Tabs In SCRIPT/VS" on page 36.

.TB	[<u>SET</u> / ADD / CLR / DEL]	[[f/]h [f/]h ... [f/]h]

where:

ADD specifies that the tab displacements given are to be added to those currently defined.

DEL or **CLR** specifies that the tab displacements given are to be removed from those currently defined.

SET specifies that all the old tab stops are to be cleared and a new set of tab stops is to be defined.

h specifies the horizontal displacements of the tab stops. SCRIPT/VS displaces to the next stop by padding with blanks or other fill character. The sequence for any single .TB control word must consist of increasing positive values separated by one or more blanks. However, a .TB ADD control word can insert new tab stops between existing ones.

If no parameters are specified with .TB SET, the default tab settings are restored.

If .TB SET 0 is specified, all tab settings are cleared.

f/ specifies the fill character to be used in displacing through position h. If the fill character is to be the blank, it need not be specified.

Default: 5, 10, 15, 20, 25, 30, 35, 40, 45, 45, 50, 55, 60, 65, 70, 75, and 80

Notes:

1. This control word acts as a break.
2. The tab settings on any single .TB control word must be increasing. Tab settings on any single .TB control word that are not so ordered result in an error message.
3. Tab characters that are found beyond (to the right) of the last defined tab stop are converted to a single blank.
4. The fill character is formatted in the current font when the fill string is being formatted.
5. If the space to the next tab stop is less than the width of one fill character, the tab stop after the next is used.
6. On the 3800, fill characters are only supported with monospaced fonts If you use fill characters with proportionally spaced fonts, vertical misalignment may result.
7. Backspaces after a tab have the effect of reducing the tab position for non-3800 logical devices, but the distance to be tabbed is never reduced to less than one character space.
8. No more than 99 tab stops may be set at one time.
9. Tabs beyond the column margin set by .CL [Column Line Length] are ignored.

Examples:

- .tb 10 20 */30 40

  Tab positions are interpreted as character positions 10, 20, 30, and 40. If a tab character is processed between positions 20 and 30 of a line, the positions from the current position up through and including position 30 are filled with asterisks (*) instead of blanks. The next character goes in position 31. For example, using the system symbol &$TAB to generate tab characters, the line,

  ```
  &$TAB.text&$TAB.text&$TAB.text
  ```

  results in:

```
        text      text******text          Tab positions revert to default
                                          values of 5, 10, 15, etc.
•   .tb
```

.TC [TABLE OF CONTENTS]

The .TC [Table of Contents] control word causes the automatically generated table of contents to be imbedded and printed. Entries may be placed in the table of contents by head level control words .H0 - .H6 [Head Level 0 - 6] and by the .PT [Put Table of Contents] control word.

The .TC [Table of Contents] control word is discussed in "Chapter 6. Head Levels and Table of Contents" on page 73.

.TC	[**1** / **n**] [**name** / **control** / **/**]

where:

n is the number of page numbers to be reserved for the table of contents. If omitted, 1 is assumed. This operand is meaningful when the table of contents is at the front of the document, and the TWOPASS option is used to process it. On the first pass, the table of contents is empty, but on the second pass, it may occupy several pages. If the page numbers in the table of contents are to be accurate, every entry in the table of contents must have the same page number on both passes. After .TC, if .PA n, or .PN n, explicitly sets the page number before .PT sets anything into the table of contents, then all the pages from the table of contents to the explicit .PA n, or .PN n will be sequentially numbered. If not, the n value on the .TC control word will be used to determine the number of the page after the table of contents.

name is an optional line to be used as the title of the table of contents. If no name is given, the word CONTENTS is used. A head-level 1 (.H1) is generated at the top of the table of contents using the name given or the word CONTENTS.

control is a control word or macro to be processed at the top of the table of contents in lieu of the .H1. If this parameter begins with a period, it is assumed to be a control word, and not a name.

/ signals SCRIPT/VS not to generate any head level 1 for the table of contents. Use this when you want no name on the table of contents, you have no control word to be executed, and you don't want the default name CONTENTS to be generated.

Default: 1

Notes:

1. When .TC is encountered, a head level 1 is processed. A page eject is done if not already at the top of a page, but no entry is placed in the table of contents for the head. All table of contents entries that have been saved in the utility file DSMUTTOC are then formatted and printed. The entries come from the head level control words whose definitions call for table of contents entries (by default, the control words .H0 through .H3 cause these entries) and from any explicit .PT control words in the source file which have been executed prior to the .TC, either on the current or previous pass.

2. The table of contents is formatted according to the line and page dimensions in effect at the time the .TC control word is encountered, not those in effect when the head level was processed. Each line in the table has the revision code and the page number that were in effect when the head level was processed.

3. When the table of contents is completely formatted, another page eject is done, and the new page is

numbered as though sequential page numbering had occurred and the table of contents had occupied exactly n pages. If the table takes other than n pages, there will be either a gap or an overlap in pagination. If TWOPASS is in effect, the pagination may be allowed to run sequentially if the page number is explicitly reset before the first table of contents entry following .TC. In this case, it doesn't matter what was specified for n.

4. This control word acts as a break. It is not allowed in a keep.

5. If the .TC control word is used at the beginning of a document you must be careful that the resolution of symbols during the second pass does not cause the document to expand or contract in such a way that the page numbers established during the first pass are caused to be invalid.

Example:

See the table of contents of this document for an example of an automatically generated table of contents.

.TE [TERMINAL INPUT]

Use the .TE [Terminal Input] control word when you want to enter text or control lines during the processing of the input file.

.TE	[<u>1</u> / n / **ON** / **OFF** / **line**]

where:

n specifies the number of lines that will be accepted from the terminal. If omitted, 1 is assumed.

ON starts an open-ended terminal input mode.

OFF turns off terminal input mode if it was ON, or if n was given and has not yet been exhausted.

line is an input line to be processed. The "line" form is available with .TE because it is a Type 1 control word, but it actually does not read anything from the terminal. The control word:

```
.te read this line
```

causes the line "read this line" to be processed as an ordinary input line, but SCRIPT/VS obviously does not read it from the terminal, because it already has the line.

Default: 1

Notes:

1. When the .TE control word is encountered, your terminal keyboard is unlocked to accept input lines. The input lines may be text or control words and are processed as if they had been read from an imbedded file (see the .IM [Imbed] control word). The only exceptions to this are the .GO [Goto] and ... [Set Label] control words, which are not allowed during terminal input. If a numeric operand was specified, terminal input is ended after reading n lines. If no operand was specified, only one line is read from the terminal. If ON was specified, input is accepted from the terminal until ended with .TE OFF. When terminal input is ended, processing reverts back to the line following the .TE control word in the file. If the TWOPASS option of the SCRIPT command is in effect, .TE control words in the input file will be processed on both passes.

2. If you use .TE while the formatted output is being displayed at your terminal, the input and output may be interspersed. This can be useful for testing or experimentation, but is not usually appropriate for final output.

3. The .RD [Read Terminal] control word merely unlocks the keyboard to allow you to type lines in the midst of the normal terminal output. It does not process what you type. The .TE control word, on the other hand, may be used to enter control words or cause text

input to be formatted and to appear in the output when the output is written to a device other than the terminal. The .TE control is in effect an imbed, where the "file" imbedded is your keyboard.

4. Use the .TY [Type on Terminal] control word immediately before the .TE control word to display prompting messages.

5. If .TE ON was specified, the number of lines to be read is open ended. It can be ended by .TE OFF, but since your keyboard is a simulated imbed file, .EF, .QQ, or .QU will also end it.

6. The .TE control word may be used to enter control words to specify a particular processing of the input file, such as revision codes or conditional sections.

7. Terminal input may be read from a disk file if the terminal input file name DSMTERMI has been associated with the file or data set name with the .DD [Define Data File-id] control word. See the discussion of .DD for more information.

8. In the ATMS-III environment, a null line is returned. In batch environments, the file DSMTERMI is read.

.TH [THEN]

The .TH [Then] control word can be used in conjunction with the .IF [If] control word to process SCRIPT/VS input lines conditionally. The target line is processed only if the most recently performed .IF [If], .AN [And], or .OR [Or] control word resulted in a true condition.

The .TH [Then] control word is discussed in "The .IF Control Word Family" on page 111.

.TH	**target**

where:

target is any valid SCRIPT/VS input line. It may be a control word or text. If the most recently performed .IF [If] was true, the target line is processed next, with the first nonblank character after .TH treated as the first position of the subject line. If the condition was not true, the target line is ignored, and processing continues with the input line that follows the .TH control line.

Notes:

1. For readability, an optional "EN" may be added without an intervening blank to the .TH control word. This allows the control word to be written as ".TH" or ".THEN".

2. The .TH [Then] and .EL [Else] control words, in conjunction with .IF [If], .AN [And], and .OR [Or], allow you to construct complex logic statements.

3. The .TH and .EL control words do not cause a break or change the true/false condition; the target control word might, if it is processed. For example, the input lines

```
.if &a eq &b
.else .if &c eq &d
.then .ty Yes.
```

are equivalent to the line

```
.if &a eq &b .or &c eq &d .ty Yes.
```

4. Multiple .TH [Then] and .EL [Else] control words may follow an .IF [If] control word; only the .TH [Then] control words will be executed if the .IF [If] resulted in a true comparison; only the .EL [Else] control words will be executed if the .IF [If] resulted in a false comparison.

5. If there is no most recently performed comparison, the target will be processed.

Examples:

- The following input lines

```
.if &a eq &b .ty Yes,
.if &a eq &b .ty still.
```

are equivalent to the following lines:

```
.if &a eq &b
.then .ty Yes,
.th .ty still.
```

.TI [TRANSLATE INPUT]

Use the .TI [Translate Input] control word to translate the input text from one input representation to another. **This control word should be used with caution, since this translation will occur before any other processing is done.**

The .TI [Translate Input] control word is discussed in "Character Manipulation" on page 87.

.TI	**[s t] ...**

where:

s is the source character to be translated. It may be a single character, or a 2-character hexadecimal code.

t is the desired output representation of the source character. It may be a single character, or a 2-character hexadecimal code.

Default: Restores the initial input translate table.

Notes:

1. Multiple pairs of translate characters may be specified with a single .TI control word.
2. Translate-character specifications remain in effect until explicitly respecified.
3. A .TI control word with no operands causes the translation table to be reinitialized and all previously specified translations to be reset.
4. The .TI control word does not cause a break.

.TM [TOP MARGIN]

The .TM [Top Margin] control word specifies the amount of vertical space to be reserved above the text and running heading on output pages, overriding the initial value established for the device.

The .TM [Top Margin] control word is discussed in "Changing the Page Length" on page 57. Figure 8 on page 56 shows the relationship of the .TM [Top Margin] to the layout of a SCRIPT/VS output page.

.TM	$\left[\begin{array}{c} v \\ +v \\ -v \end{array}\right]$

where:

v specifies the amount of vertical space to be reserved at the top of output pages. If no value is specified for v, the default value for the logical device will be used. v must be large enough to accommodate the heading margin and the heading space, both of which are allocated from the top margin area. The top and bottom margins may not fill the page so that there is no room left for formatted text. If the value specified for the top margin is so large that there would not be at least one line available for text, the old top margin is left unchanged, and an error message is issued.

+v or -v increases or decreases the existing top margin by the amount given. The calculated top margin value must fall within the allowed range, or an error message will be issued.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

1. When the .TM control word is processed, a new top margin is set for future pages, but it is too late for the new value to take effect on

the current page. If you want to increase the heading margin (.HM) or the heading space (.HS) beyond what the current top margin will accommodate, you must change the top margin value first.

2. This control word does not cause a break, and it takes effect on the page after the control word is encountered.

3. If you specify .TM 0, the heading margin and the heading space are also made zero automatically. Any other top margin specification that is smaller than the current size of the heading margin plus the heading space cannot be satisfied, and results in an error message.

4. An error message is also issued if you try to set the top and bottom margins so that they fill the entire page, without at least one line left for text.

5. An extra blank page will precede the first formatted page for 1403 output if you request, through some combination of .TM, .HS, and .HM, that the first line be printed above the fourth line of the page. The blank page occurs because SCRIPT/VS assumes that the channel one punch of the carriage control tape is at the fourth line of the page.

.TR [TRANSLATE CHARACTER]

The .TR [Translate Character] control word allows you to specify the output representation of each character in the source text.

The .TR [Translate Character] control word is discussed in "Character Manipulation" on page 87.

.TR	**[s t] ...**

where:

s is the source character to be translated. It may be a single character, or a 2-character hexadecimal code.

t is the desired output representation of the source character.

More than one pair of source and intended output codes may be specified with a single .TR control word.

Default: Restores the initial output translate table.

Notes:

1. The .TR control word is primarily of use when the final output device uses a different character set than was used to create the source SCRIPT file.

2. The text associated with running title lines (.RT) is translated under control of the translations in effect at the time that the .RT control word was processed. If you change the translations after the running title has been saved for future use, it is too late to affect that running title.

3. Since control words are only processed internally, they are never translated by the .TR control word. However, text data associated with a control word (as in running titles and typed messages) can be translated.

4. Translate-character specifications remain in effect until explicitly respecified.

5. A .TR control word with no operands causes the translation table to be reinitialized and all previously specified translations to be reset.

6. By using the .IF, .CS, or .TE control words, you may specify different output character sets for different runs with different output devices.

7. The .TR control word does not cause a break.

8. The hexadecimal codes for each printable character for the various character sets and fonts used by SCRIPT/VS are shown in "Appendix A. SCRIPT/VS Summary" on page 341.

9. During the time a translation is in effect, every occurrence of the character is translated to the designated output character in formatted text. You should therefore take care not to translate characters that will be needed during that range. The actual translation is done at various times in the

formatting process, depending on the requirements of the logical device for which the document is being formatted. The latest time when a translation can be done is when a line is finished, and is placed in a column. You should assume, therefore, that a translation will be needed until the next break is done, whether this happens naturally because a line is full, or is forced by a control word that causes a break.

Examples:

- .tr 0 b0 1 b1 ... 9 b9

  This causes the characters 0, 1, ..., 9 to print as their corresponding superscript characters if they are available in the current font. For example, the formula:

  X2+Y2=Z3

  prints as:

  $X^2+Y^2=Z^3$

- .tr 40 ?

  This causes all blanks in the file to be translated to question marks (?) on output.

.TS [TRANSLATE STRING]

Use the .TS [Translate String] control word to translate an input character to a string. **This control word should be used with caution, since this translation will occur after input translation but before any other processing is done.**

The .TS [Translate String] control word is discussed in "Character Manipulation" on page 87.

.TS	**s**	**/string[/]** **OFF** **IGNORE**

where:

s	is the source character to be translated. It may be a single character or a 2-character hexadecimal code.
/	is any delimiter character.
string	is the desired output representation of the source character. It may be any string from 0 to 255 characters in length.
OFF	specifies that string translation is to be stopped for the specified character.
IGNORE	specifies that the given character string is to be retained in the text and included in the output, but not measured during formatting. That is, an IGNOREd character is treated as a zero width character.

Notes:

1. A character may be set to a null string by either specifying adjacent delimiters or omitting the string specification altogether.

2. Unlike other delimited strings (for example, on the .RT [Running Title] control word), the .TS string is not scanned for internal delimiters. Thus the end delimiter is only required if you want trailing blanks or the string ends with the character that you have chosen as the delimiter.

3. String translation is only performed when symbol substitution is in effect.

4. String translation specifications remain in effect until explicitly respecified.

5. Unlike the .TI [Translate Input] and .TR [Translate Character] control words, you must explicitly stop each string translation by specifying the OFF option. For example:

   .ts % /<percent sign>

   will cause all occurrences of % in the input to be replaced by the string <percent sign> until you enter:

   .ts 6c off

Note that it can be difficult to use the .TS [Translate String] control word unless you turn substitution off or you specify the value in hexadecimal (be careful if you have translated one of the characters used as a hexadecimal digit).

6. String translations specified with the .TS [Translate String] control word are stored in the symbol table. Since string translation occurs at the same time as symbol substitution, the substituted string is not subject to further symbol substitution.

7. The .TS control word does not cause a break.

.TU [TRANSLATE UPPERCASE]

The .TU [Translate Uppercase] control word allows you to specify the output representation of characters when uppercase output has been requested with the .UC [Underscore and Capitalize], .UP [Uppercase], or .BF [Begin Font] control words or the UPCASE option on the SCRIPT command.

The .TU [Translate Uppercase] control word is discussed in "Character Manipulation" on page 87.

.TU	[s t] ...

where:

s is the source character to be translated. It may be a single character, or a 2-character hexadecimal code.

t is the desired output representation of the source character.

More than one pair of source and intended output codes may be specified with a single .TU control word.

Default: Restores the initial output translate table.

Notes:

1. The .TU control word is primarily of use when the default translate table designed for English is not appropriate.

2. Translate specifications remain in effect until explicitly respecified.

3. A .TU control word with no operands restores uppercase translation to the default.

4. The default uppercase translation has the same effect as the following .TR [Translate Character] control word:

```
.tr a A b B ... z Z
```

5. The .TU control word does not cause a break.

.TY [TYPE ON TERMINAL]

The .TY [Type on Terminal] control word causes one line of information to be displayed at your terminal, or written into the file DSMTERMO, no matter where the SCRIPT/VS formatted output is going.

.TY	text

where:

text is the line to be typed. It is used only for this message. It does not become part of your document unless the document output is also being typed at your terminal.

Notes:

1. The .TY [Type on Terminal] control word will cause a file with the id DSMTERMO to be created in the background environment.

2. When the .TY control word is processed, the text line given is typed at the terminal. This line is not part of the document. SCRIPT/VS does not process the line for output; the line is not justified, or formatted in any way. However, the line is scanned for control word separators and symbols are substituted. The text to be typed is translated according to the .TR [Translate Character] translations currently in effect.

3. You may use the .TY control word to issue a prompting message before a .TE [Terminal Input] or .RV [Read Variable] control word.

4. The information line printed is not counted as part of the normal output. Thus, if the formatted output is being typed on the terminal, the paper positioning may become incorrect and require manual adjustment. In general, the .TY control word should be used for document-driven messages when the formatted output is going to a printer or to a disk file.

5. Contrast this control word with .MG [Message]. The .MG control word allows you to issue a true SCRIPT/VS message. A true message may have any of several different degrees of severity, it may terminate SCRIPT/VS processing, and its destination and final form are controlled by the MESSAGE command option. The .TY control word merely types out a line without any of the function of a true message.

6. The .TY control word is ignored in the ATMS-III environment.

Example:

```
.ty Do you want 2 column output?
.rv answer
.if x&answer ne xyes .go by2col
.cd 2 0 46
.cl 43
...by2col
```

.UC [UNDERSCORE AND CAPITALIZE]

The .UC [Underscore and Capitalize] control word automatically underscores and capitalizes one or more input lines.

The .UC [Underscore and Capitalize] control word is discussed in "Underlining and Capitalizing" on page 47.

.UC	[<u>1</u> / n / ON / OFF / line]

where:

n specifies the number of lines to be underscored and capitalized. If omitted, 1 is assumed. If .UC n is specified when .UC ON is in effect, .UC is turned off when n lines have been underscored and capitalized, or when .UC OFF is encountered.

ON specifies that subsequent text lines are to be underscored and capitalized.

OFF terminates underscore and capitalization mode if it was ON, or if n has been specified and has not been exhausted.

line is a single text line to be capitalized and underscored.

Initial Setting: OFF

Default: 1

Notes:

1. Use the .UC control word whenever you have a line of data that is to be formatted in capital letters and underscored. This control word provides the combined function of .US [Underscore] and .UP [Uppercase].

2. The .UC control word does not cause an automatic break; single words in a sentence may be underscored and capitalized.

3. By default, capitalization is performed by translating a-z to A-Z. The .TU [Translate Uppercase] control word may be used to extend capitalization for languages other than English.

Examples:

- Underscoring and capitalizing a single word:

```
 This sentence has
 .uc one
 word processed by .UC.
```

  results in

  This sentence has <u>ONE</u> word processed by .UC.

.UD [UNDERSCORE DEFINITION]

Use the .UD [Underscore Definition] control word to specify whether blank characters should be underscored whenever automatic underscoring is done using the .US [Underscore] and .UC [Underscore and Capitalize] control words or when a font that calls for underscoring is in effect.

.UD	[**ON** / **OFF**]

where:

ON specifies that blanks are to be underscored.

OFF specifies that blanks are not to be underscored.

Initial Setting: ON

Default: ON

Notes:

1. This control word does not cause a break.
2. In order that documents containing .UD [Underscore Definition] specifications supported by SCRIPT/VS Version 1.0 will be processed, no error messages will be issued if the .UD [Underscore Definition] control word is used in the form previously supported.

.UN [UNDENT]

Use the .UN [Undent] control word to cause the next line to be shifted. The current indention is changed for the next line only, then restored to its previous value for subsequent lines.

.UN	[**0** / **h** / **+h** / **-h**]

where:

h specifies the amount of horizontal space by which the indention is to be altered for the next output line only. A SCRIPT/VS "undent" is a negative indent. If -h is specified, the .UN control word is effectively the same as the .IL [Indent Line] control word. If omitted, 0 is assumed, and the indention is not changed.

Initial Setting: 0

Default: 0

Notes:

1. The .UN control word provides function similar to that provided by the .OF [Offset] control word. The choice between using .UN and .OF is usually a matter of personal preference. They may also be used at the same time to control margins that shift both right and left.
2. This control word acts as a break.
3. The value specified in a .UN control word is subtracted from the current indention (indent value plus offset value) to determine where to format the next line. If the .UN amount exceeds the current indention amount, an error message results.
4. If successive .UN or .IL control words with positive or negative specification for h are encountered without intervening text lines, the .UN value is reset to the latest specified value each time.
5. The .UN control word is triggered by the next text, skip, or space line.

Example:

```
    .in 3p
    .un 3p
```

If an indention of 3 picas is in effect (as in these lines), the next line is undented to the left margin; all following lines have the normal indention of 3 picas from the left margin.

.UP [UPPERCASE]

The .UP [Uppercase] control word automatically capitalizes one or more input lines.

The .UP [Uppercase] control word is discussed in "Underlining and Capitalizing" on page 47.

.UP	[**1** / **n** / **ON** / **OFF** / **line**]

where:

n specifies the number of lines to be capitalized. If omitted, 1 is assumed. If .UP n is specified when .UP ON is in effect, capitalization is turned off when n lines have been capitalized, or when .UP OFF is encountered.

ON specifies that subsequent text lines are to be capitalized.

OFF terminates capitalization mode if it was ON, or if n has been specified and has not been exhausted.

line is the line to be capitalized.

Initial Setting: OFF

Default: 1

Notes:

1. Use the .UP control word whenever you have a line of data that is to be formatted in capital letters. If your entire document is to be in capital letters, use the UPCASE option of the SCRIPT/VS command line.
2. The .UP control word does not cause an automatic break. Single words in a sentence may be capitalized.
3. Another method of capitalizing a single word is to use the uppercase attribute symbol &u' that is recognized by the symbol processor.
4. By default, capitalization is performed by translating a-z to A-Z. This translation can be extended with the .TU [Translate Uppercase] control word for languages other than English.

Examples:

- Capitalizing a single word:

```
    This sentence has
    .up one
    capitalized word.
```

  results in:

  This sentence has ONE capitalized word.

- Capitalizing a single word using the symbol processor's uppercase attribute:

  This sentence has &u'one capitalized word.

  results in:

  This sentence has ONE capitalized word.

.US [UNDERSCORE]

The .US [Underscore] control word automatically underscores one or more input lines.

The .US [Underscore] control word is discussed in "Underlining and Capitalizing" on page 47.

.US	[1 / n / ON / OFF / line]

where:

n specifies the number of lines to be underscored. If omitted, 1 is assumed. If .US n is specified when .US ON is in effect, .US is turned off when n lines have been underscored, or when .US OFF is encountered.

ON specifies that subsequent text lines are to be underscored.

OFF terminates underscoring if it was ON, or if n has been specified and has not been exhausted.

line is a single text line to be underscored.

Initial Setting: OFF

Default: 1

Notes:

1. Use the .US control word whenever you have a line of data that is to be underscored.
2. The .US control word does not cause an automatic break; single words in a sentence may be underscored.
3. The .UD [Underscore Definition] control word controls whether blank characters (X'40') are underscored or not by the .US [Underscore] control word.

Examples:

- Underscoring a single word:

```
This sentence has
.us one
underscored word.
```

results in:

This sentence has <u>one</u> underscored word.

.UW [UNVERIFIED WORD]

The .UW [Unverified Word] control word is generated by SCRIPT/VS and executed whenever unverified and potentially misspelled words are found.

The .UW [Unverified Word] control word is discussed in "SPELLCHK: Verify Spelling" on page 182.

.UW	word word ...

where:

word is the list of unverified words found for one input line.

Notes:

1. Whenever misspelled words are found in an input line, the .UW control word is executed with the misspelled words as parameters.

Normally this control word issues an error message to tell you that those words were not verified.

However, if you wish to have some function performed when a misspelled word is encountered, you can define a .UW macro with .DM [Define Macro], and, assuming macro substitution is ON, your .UW macro will be executed whenever misspelled words are found. Note, however, that after the .UW control word or macro is processed, the misspelled words are still on the line, and are processed as part of that text input line. In other words, you cannot use the .UW macro to correct or remove such words from a line.

2. When unverified words are found, you may want to add them to an addenda dictionary using the .DU [Dictionary Update] control word so that only the first occurrence is detected, or write the words to a file using the .WF [Write To File] control word so that you can review the file created for possible permanent addition to your dictionary.

.WF [WRITE TO FILE]

Use the .WF [Write To File] control word to cause lines of text or control words to be written to an output file with the id DSMUTWTF.

.WF	[<u>1</u> / n / ON / OFF / line / IMBED / ERASE]

<u>where:</u>

n specifies that the next n lines are to be written into the DSMUTWTF file.

ON specifies that the following text and control words are to be written into the DSMUTWTF file until .WF OFF is encountered.

OFF stops writing text and control words to the DSMUTWTF file, whether ON was specified, or a number of lines in 'n' that has not yet been exhausted. The .WF OFF control word must occur on a line by itself.

line is a line of text or control words to be written to the file.

IMBED causes the DSMUTWTF file to be imbedded.

ERASE causes the DSMUTWTF file to be erased.

<u>Notes:</u>

1. All the text and control words between the .WF ON and OFF control words will be written into the DSMUTWTF file. No .WF control word is written to the file. Any .WF other than .WF OFF is ignored when .WF is writing lines to the file.

2. If symbol substitution is ON, the lines that are written to the file will have been substituted. If substitution is ON when the file is imbedded, the lines will be substituted again if any unresolved symbols remained from the first substitution.

3. The file-id DSMUTWTF may be associated with different file or data set names using the .DD [Define Data File-id] control word. See the discussion of .DD for more information. Different groups of information can be written to different actual files when .DD is used.

4. The .WF control word cannot write into a file that is currently in use for .AP [Append] or .IM [Imbed]. If an imbedded or appended file is ended with the .EF [End of File] control word it is still "in use" unless the CLOSE option was specified.

5. The data written to a file will be added after any existing data in the file, unless DSMUTWTF has been associated with a member of a partitioned dataset. In this case, the data replaces any existing member.

.WZ [WIDOW ZONE]

The .WZ [Widow Zone] control word specifies that single-line widows in body text are to be suppressed.

.WZ	[<u>ON</u> / OFF]

where:

ON specifies that widows are to be suppressed. This is the default and the initial setting.

OFF specifies that widows are not to be suppressed.

Initial Setting: ON

Default: ON

Notes:

1. This control word causes a break.
2. Widow processing is not performed if the depth of the body of the page is less than 8 lines.
3. All controls that cause a break effectively end any widow zone and the next line of text will start a new widow zone. This may result in what appears to be a single line widow but which is actually a single line paragraph.
4. Widow zone processing is not performed in footnotes.

.ZZ [DIAGNOSTIC]

The .ZZ [Diagnostic] control word provides the system programmer with the ability to dump selected internal SCRIPT/VS control blocks. This control word is ignored unless enabled by the DUMP option of the SCRIPT command.

.ZZ	{ **ON** }
	{ **OFF** }
	{ **PROG** }
	{ **DUMP** nn [nn ...] }

where:

ON allows dump data that is specified in message definitions to be printed. This is the initial setting if the DUMP option of the SCRIPT command is specified.

OFF prevents dump data that is specified in message definitions from being printed.

PROG causes a program exception, regardless of whether the DUMP option was specified on the SCRIPT command. This is useful when used in an interactive environment which provides dynamic debugging tools.

DUMP causes immediate dumping of the data areas indicated by the code numbers given in "nn". The valid range of values is 9 through 64. Any number of values may be specified; if none is specified, all are set on. The valid code numbers are as follows:

9	Active save areas (most recent first) - DSMSAVD
10	Global area - DSMSGLB
11	Language processor common area - DSMSECT
16	Logical Device table - DSMSLDT
17	TRS work area - DSMTRSC

Notes:

1. The .ZZ control word may be used several times within the source text to provide selective dump information.
2. The information is dumped to the same destination to which error messages are written.
3. The control will only be active if the DUMP option was specified on the SCRIPT command.

Figure	Page	Description
21	342	A picture of the SCRIPT/VS page layout.
22	343	List of the utility files that SCRIPT/VS creates or uses.
23	343	Summary of the options of the SCRIPT command.
24	345	Summary of the SCRIPT/VS control words and their parameters.
25	354	List of the control words that always result in a break between input lines of text.
26	354	List of the control words that always take effect on the next output page.
27	355	List of the control words that are not allowed within a keep, float, footnote, or running heading or footing.
28	355	List of the control words that are executed during the definition phase of a running heading or footing.
29	356	List of control words that start the page.
30	356	List of obsolete control words.
31	357	List of the control words whose initial values are based on the logical output device.
32	358	List of logical devices defined to SCRIPT/VS.
33	359	Summary of default head-level characteristics.
34	360	List of paramaters contained within the formatting environment.
35	361	Summary of SCRIPT/VS system symbols.
36	363	List of the attributes of a symbol.
37	363	List of the characters that delimit words for purposes of spelling verification.
38	364	The IBM 1403 Printer's TN print train character set.
39	364	List of the fonts provided with SCRIPT/VS for use with the IBM 3800 Printing Subsystem.
40	365	List of the fonts provided with the IBM 3800 Printing Subsystem.

Figure 20. Index to SCRIPT/VS Summary

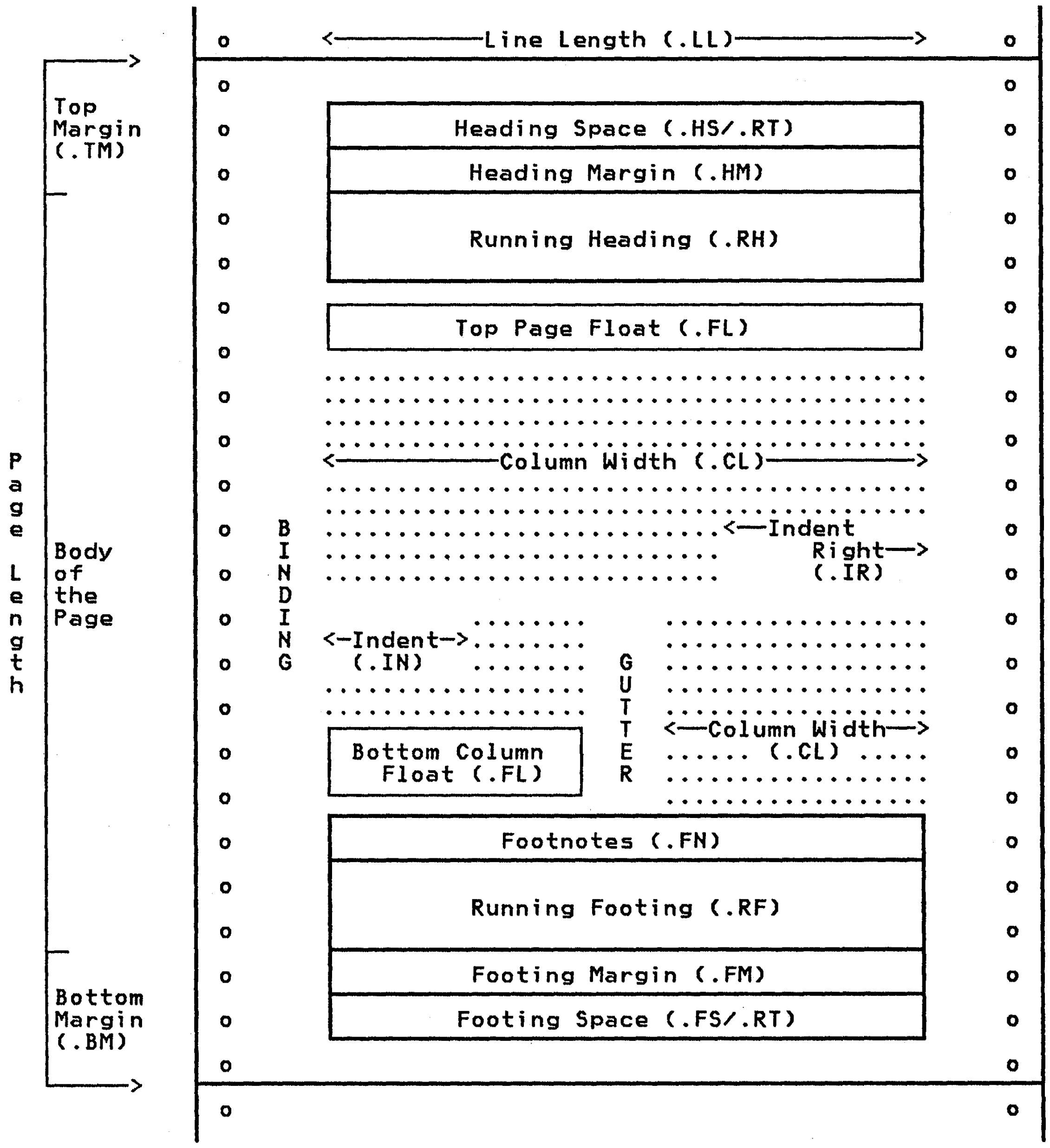


Figure 21. SCRIPT/VS Terms for Parts of the Page: Note that Top Margin and Bottom Margin include all the space on the paper that is accessible to SCRIPT/VS. For terminals and 1403-type printers, this includes the entire page. For 3800-type devices, Top and Bottom Margin do not include 1/2 inch on each side of the interpage perforation. This space is reserved by the 3800 Printing Subsystem for accelerating and decelerating the paper when it is necessary to halt the paper.

SCRIPT/VS Utility Files			
File-id	Description	Control Word	Option
DSMTERMI	Terminal input file	.TE, .RV	
DSMTERMO	Terminal output file	.TY	TERM
DSMUTCTF	STAIRS/VS CTF file	.SO	CTF
DSMUTDIM	Delay imbed file	.DI	
DSMUTMSG	Error messages file	.MG	MESSAGE(DELAY)
DSMUTTOC	Table of Contents file	.TC, .PT	
DSMUTWTF	Write to File file	.WF	

Figure 22. File-id's of SCRIPT/VS Utility files: SCRIPT/VS uses or creates these files as a result of the control words or options indicated. Any of these files may be redefined with the .DD [Define Data File-id] control word.

Option	**Parameters**	**Description**
BIND	(bind) (obind ebind)	Shift the page image to the right.
CHARS	(font1 ... font4)	Specify up to four fonts.
CONTINUE		Continue processing after a nonsevere error occurs.
CTF		Prepare output in STAIRS/VS Condensed Text Format.
DEST	(station-id)	Specify a remote output station. (Valid only in TSO.)
DEVICE	(devtype)	Specify a logical output device.
DUMP		Enables the .ZZ [Diagnostic] control word.
FILE	[(fileid)]	Specify a disk file for output. (Not valid in ATMS-III.)
INDEX		Enable the .PI [Put Index] control word.
LIB	(libname ...) (opnum ...)	Specify symbol and macro libraries. (Only one in TSO; up to eight in CMS and ATMS-III.)
MESSAGE	([DELAY] [ID] [TRACE])	Control message printing.
NOPROF		Suppress the profile.
NOSPIE		Prevent entering SPIE exit routines. (Valid only in CMS and TSO.)
NOWAIT		Prevent prompting for paper adjustment. (Valid only for typewriter terminals in CMS and TSO.)
NUMBER		Print file name and line number.

Figure 23. Summary of SCRIPT Options (Part 1 of 2)

Option	Parameters	Description
OPTIONS	[(fileid)]	Specify a file that contains SCRIPT options. (Valid only in CMS and ATMS-III.)
PAGE	[([PROMPT] [[FROM] p [TO] q] [[FROM] p FOR n] [[FROM] p ONLY])]	Selectively print pages. (PROMPT is valid only in CMS and TSO.)
PRINT	[(copies,class, fcb,ucs)]	Produce printer output. (Suboptions are valid only in TSO.)
PROFILE	[(fileid)]	Specify a profile. (A file to be imbedded before the primary input file is processed.)
QUIET		Suppress formatter identifier message.
SEARCH	(libname) (opnum ...)	Specify a library. (Not valid in CMS.)
SPELLCHK		Enable the .SV [Spelling Verification] control word.
STOP		Print separate pages at the terminal. (Valid only for typewriter terminals in CMS and TSO.)
SYSVAR	(n value ...)	Set symbol values for &SYSVARn.
TERM		Display the output at a user's terminal.
TLIB	(libname ...)	Specify spelling checking and hyphenation libraries. (Valid only in CMS.)
TWOPASS		Prepare with two formatting passes, and produce output on the second pass.
UNFORMAT		Print all input lines without formatting.
UPCASE		Fold lowercase letters to uppercase before printing.
@user-option	[(sub-options...)]	User-defined options, which must begin with the character "@". (Valid only in CMS.)

Figure 23. Summary of SCRIPT Options (Part 2 of 2)

Control Word	Parameters	Description
...	label [input line]	Set Label: Inserts a line that can be used as the target of a .GO control word.
.AA	tag [apf [(rules)]] [apf [(rules)]]	Associate APF: Maps a GML tag to the macro or control word which processes it, and gives the attribute scanning rules for the tag.
.AN	x test y input-line "test" can be: lt le eq ne gt ge < <= = ¬= > >=	And: Tests the relationship between "x" and "y". When the test is satisfied and the most recently executed .IF, .AN, or .OR was also satisfied, SCRIPT/VS processes the "input-line".
.AP	{name|(file-id)} [token1 ... token14]	Append: Allows an additional file to to be appended to the file just processed
.BC	[ON|OFF]	Balance Columns: Causes SCRIPT/VS to attempt to balance the columns when a page eject occurs or when the column definition is changed.
.BF	[font-id]	Begin Font: Causes SCRIPT/VS to use a new font. Based on logical device.
.BM	[v|+v|-v]	Bottom Margin: Specifies the amount of space in the bottom margin area. Causes break. SCRIPT/VS system symbol: &$BM
.BR		Break: Prevents the concatenation of the following text line with preceding text.
.BX	[NEW|OFF|CAN|SET] [d1 [/] d2 ...] [CHAR name]	Box: Draws horizontal and vertical lines around subsequent output text. Causes a break.
.CB		Column Begin: Causes an eject to the next column (or next page). Causes a break.
.CC	[v]	Conditional Column Begin: Causes a column eject if less than a specified amount of space remains in the column. Causes a break. .CC causes a column eject unless there is no data in the current column.
.CD	n [p1 p2 ... p9]	Column Definition: Specifies the number of columns on the page and position of each column. Causes a break. Initial value: One column at position 0.
.CE	[1|n|ON|OFF| input-line]	Center: Centers text lines between the current left and right margins. Causes a break.
.CL	[h|+h|-h]	Column Width: Specifies the width of each column (all columns are the same width). Causes a break. SCRIPT/VS system symbol: &$CL Default: Line length
.CM		Comment: Identifies a comment line.

Figure 24. SCRIPT/VS Control Word Summary (Part 1 of 9)

Control Word	Parameters	Description
.CO	[ON|OFF]	Concatenate Mode: Causes output lines to be formed by concatenating input lines. Causes a break.
.CP	[v]	Conditional Page Eject: Causes a page eject if less than a specified amount of space remains on the page. Causes a break. Default: .CP causes a page eject unless there is no data on the current page.
.CS	n [ON|OFF] n [INCLUDE|IGNORE]	Conditional Section: Allows conditional inclusion of input in the formatted output. Initial value: All conditional sections included.
.CT	[input-line]	Continued Text: Appends the "input-line" to the previous input text without an intervening word space.
.CW	[character]	Control Word Separator: Defines the control word separator character. SCRIPT/VS special character: &$CW Initial value: ; (semi-colon)
.DC	[option char ...|OFF]	Define Character: Defines the characters for special functions. Options: ASEP: Array element separator characters CONT: Line continuation character (&$CONT) CW: Control word separator (&$CW) GML: GML tag delimiter (&$GML) IXB: Index term blanks IXI: Index term nulls STOP: End-of-sentence characters PUNC: Punctuation characters PS: Page number symbol (&$PS) RB: Required blank (&$RB) WORD: [See Figure 37 on page 363] Initial Values: ASEP: , 40 GML: : : : PS: & CONT: none STOP: . ! ? RB: 41 CW: ; PUNC: - '
.DD	name [LIB|DD|DSN file-id]	Define Data File-id: Specifies the file-id of a file to be used with the .IM [Imbed], .AP [Append] or .WF [Write To File] control words.
.DF	font-id [US|UP|UC|STOP| OS RPT n|OS CHAR c| BOX cname|FONT name]	Define Font: Define internal fonts composed of external font characteristics and line formatting functions.
.DH	n [options]	Define Head Level: Defines the format and characteristics of the section headings produced by the .Hn control words. Default: Restores initial settings.
.DI	[1|n|ON|OFF| input-line]	Delay Imbed: Delays the processing of input lines until the next page eject occurs. Causes a break.
.DL	name ... name	Dictionary List: names spelling checking and hyphenation dictionaries.

Figure 24. SCRIPT/VS Control Word Summary (Part 2 of 9)

Control Word	Parameters	Description
.DM	name(n) / [x|OFF|input-line] name [x|OFF|LIB]	Define Macro: Defines a macro using text, SCRIPT/VS control words, and special symbols.
.DS		Doublespace Mode: Causes subsequent output lines to be doublespaced.
.DU	{ADD|DEL} word ...	Dictionary Update: Adds words to or deletes words from an addenda dictionary, which is used to supplement a SCRIPT/VS main dictionary for spelling verification and hyphenation.
.EC	input line	Execute Control: Execute the input line as a control word even if there is a macro of the same name.
.EF	[CLOSE]	End of File: Simulates an end of file condition.
.EL	[input-line]	Else: Processes the "input-line" if the most recently executed .IF, .AN, or .OR control word was not satisfied.
.EM	input line	Execute Macro: Execute the input line as a macro even if macro substitution is off.
.EZ	ON|OFF|tag	EasySCRIPT: Enables or disables the EasySCRIPT processing functions.
.FL	[ON|OFF|DUMP] [TOP|BOTTOM] [COL|PAGE] [ODD|EVEN] [ORDER]	Float: Delimits a group of lines to be kept together and placed at the top or bottom of a column or page.
.FM	[v|+v|-v]	Footing Margin: Specifies the amount of space between the last line of text in the page's body and the first bottom title line. SCRIPT/VS system symbol: &$FM Default: Based on logical device type.
.FN	{ON|OFF|LEADER}	Footnote: Saves formatted text and prints it at the bottom of the page in single-column format.
.FO	[ON|OFF| LEFT|RIGHT|CENTER] [EXTEND|TRUNC|FOLD]	Format Mode: Controls concatenation and justification of input lines. Default (for .FO OFF): EXTEND
.FS	[n|+n|-n]	Footing Space: Specifies the number of lines in the bottom margin area that can contain bottom title lines. SCRIPT/VS system symbol: &$FS Default: Based on logical device type.
.GO	label	Go To: Causes SCRIPT/VS to locate the input line identified with "label" and resume processing with that input line.
.GS	[options]	GML Services: Provides a variety of functions helpful in writing APFs, including attribute scanning and symbol manipulation.

Figure 24. SCRIPT/VS Control Word Summary (Part 3 of 9)

Control Word	Parameters	Description
.HM	[v|+v|-v]	Heading Margin: Specifies the amount of space between the top title lines and the first line of text (or running heading) on the body of the page. SCRIPT/VS system symbol: &$HM
.HS	[n|+n|-n]	Heading Space: Specifies the number of lines in the top margin area that contain top title lines. SCRIPT/VS system symbol: &$HS
.HW	text-word	Hyphenate Word: Specifies hyphenation points for a word that might need to be hyphenated during formatting.
.HY	[ON|OFF|SUP|NOADD] [SET MINPT n] [SET THRESH n]	Hyphenate: Controls the SCRIPT/VS automatic hyphenation function. Initial setting: OFF, MINPT=4, THRESH=7
.Hn	[text-line]	Head Level n: Formats a section heading according to default characteristics supplied for the heading.
.IE	{H|1|2|3} string	Index Entry: Generated by the .IX control word to format an index entry.
.IF	x test y input-line "test" can be: lt le eq ne gt ge < <= = ¬= > >=	If: Tests the relationship between "x" and "y". When the test is satisfied, SCRIPT/VS processes the "input-line." Otherwise, SCRIPT/VS ignores the "input-line." "x test y" can also be: SYSPAGE eq|ne EVEN|ODD SYSOUT eq|ne PRINT|TERM
.IL	[0|h|+h|-h]	Indent Line: Indents the next output line the specified amount of horizontal space. Causes a break.
.IM	{name|(file-id)} [token1 ... token14]	Imbed: Processes the named file at this point.
.IN	[0|h|+h|-h] [FOR v1] [AFTER v2] [NOBREAK]	Indent: Specifies the amount of space subsequent output lines are to be indented from the current left margin. May cause a break. SCRIPT/VS system symbol: &$IN
.IR	[0|h|+h|-h] [FOR v1] [AFTER v2] [NOBREAK]	Indent Right: Specifies the amount of space subsequent input lines are to be indented from the current right margin. May cause a break. SCRIPT/VS system symbol: &$IR
.IT	[ON|OFF|ALL| CTL|GML|MAC|SUB| SNAP|STEP|RUN] Response to STEP: (null) PRE input-line REP input-line STK input-line	Input Substitution Trace: Provides a trace of processing for each SCRIPT/VS control word and macro, as well as symbol substitution. When .IT STEP is in effect, the user responds interactively. Initial value: No input tracing.

Figure 24. SCRIPT/VS Control Word Summary (Part 4 of 9)

Control Word	Parameters	Description
.IX	n [name|/]	Index: Generates an index from terms previously specified by with the .PI control word. Causes a break.
.JU	[<u>ON</u>|OFF]	Justify Mode: Causes left and right justification of output lines as needed to make the end of each line even with the current right margin. Causes a break.
.KP	[ON|FLOAT|DELAY|INLINE|v|v + v|OFF]	Keep: Ensures that a group of output lines are kept together in the same column. SCRIPT/VS system symbol: &$KP
.LB		Leading Blank: Is processed whenever an input line with a blank as the first character is encountered. Causes a break.
.LI	[<u>1</u>|n|ON|OFF|input-line]	Literal: Ensures that input lines are treated as text lines by SCRIPT/VS (used when a text input line begins with a period).
.LL	[h|+h|-h]	Line Length: Specifies the length of each subsequent output line. SCRIPT/VS system symbol: &$LL
.LT		Leading Tab: Is processed whenever an input line with a tab as the first character is encountered. Causes a break.
.LY	[<u>ON</u>|OFF|SYM|MAC]	Library: Specifies whether a library is to be used to resolve symbol and macro definitions. Use the LIB option to identify the libraries.
.MC		Multicolumn Mode: Restores column definition saved by a previous .SC [Single Column Mode] control word. Causes a break.
.ME	[input-line]	Macro Exit: Ends a macro and returns control to the macro's caller. The "input-line" is processed as though it were part of the macro's caller.
.MG	/[id]/text/	Message: Produces a message similar in format to the SCRIPT/VS error messages.
.MS	[<u>ON</u>|OFF]	Macro Substitution: Causes SCRIPT/VS to recognize and process macros. Initial value: OFF
.NL		Null Line: Is processed whenever an input line that contains no characters is encountered.
.OC	{line|'string'}	Output Comment: Specifies data that is to be inserted into the output document as it is, as an output comment.

Figure 24. SCRIPT/VS Control Word Summary (Part 5 of 9)

Control Word	Parameters	Description
.OF	[0|h|+h|-h]	Offset: Causes a hanging indention (a paragraph in which the indention of the first line is unchanged and subsequent lines are indented to the offset value.) Causes a break. SCRIPT/VS system symbol: &$OF
.OR	x test y input-line "test" can be: lt le eq ne gt ge < <= = ¬= > >=	Or: Tests the relationship between "x" and "y". When the test is satisfied or the most recently executed .IF, .AN, or .OR was also satisfied, SCRIPT/VS processes the "input-line".
.PA	[ODD|EVEN] [ON|OFF] [+0|n|+n|-n] [NOSTART]	Page Eject: Causes a page eject, and can set the page number of the new page.
.PF		Previous Font: Causes the last stacked font to become the current font.
.PI	[START|END|ORDER|REF] [KEY /k1/k2/k3/] /t1[/t2[/t3[/pn]]]	Put Index: Puts a mutli-level term in the index.
.PL	[v|+v|-v]	Page Length: Specifies the amount of space, including top and bottom margins, for each output page. SCRIPT/VS system symbol: &$PL
.PM	[obind [ebind]]	Page Margins: Overrides page binding set with BIND option of the SCRIPT command.
.PN	[n|ON|OFF|OFFNO| ARABIC|ROMAN|ALPHA| NORM|FRAC| PREF string]	Page Numbering Mode: Controls external and internal page numbering. Initial value: Arabic numerals from 1.
.PP	[input-line]	Paragraph Start: Begins formatting the output line as the start of a paragraph after a skip.
.PS	character	Page Number Symbol: Sets a page number symbol. SCRIPT/VS system symbol: &$PS Initial value: & (ampersand)
.PT	input-line	Put Table of Contents: Places the input line (which may be a control word, macro, GML tag, symbol, or line of text) into the file used to accumulate table of contents entries (DSMUTTOC).
.QQ		Quick Quit: Causes SCRIPT/VS processing to terminate immediately without completing the current page.
.QU		Quit: Causes SCRIPT/VS processing to terminate after completing the current page.
.RC	n s n [ON|OFF|ON/OFF] * s ADJUST h	Revision Code: Specifies a revision code symbol that is to be printed to the left of the output line that contains updated material.

Figure 24. SCRIPT/VS Control Word Summary (Part 6 of 9)

Control Word	Parameters	Description
.RD	[1|n|STOP]	Read Terminal: Allows user to type in one or more text lines while a file is being formatted. Causes a break.
.RE		Restore Status: Restores environment that has been previously saved with the .SA [Save Environment] control word.
.RF	[ON|OFF|CANCEL] [ODD|EVEN] [SUP|RES]	Running Footing: Specifies input lines that are to be saved as a running footing and processed at the bottom of each appropri- ate page. Initial value: No running footing.
.RH	[ON|OFF|CANCEL] [ODD|EVEN] [SUP|RES]	Running Heading: Specifies input lines that are to be saved as a running heading and processed at the top of each appropriate page. Initial value: No running heading.
.RI	[1|n|ON|OFF| input-line]	Right Adjust: Produces output lines that are unconcatenated input lines aligned with the right-hand margin. Causes a break.
.RN	[ON|OFF]	Reference Numbers: serially numbers output lines in body of page.
.RT	[TOP|BOTTOM] [ALL|ODD|EVEN] [1|n] /left/center/right/	Running Title: Defines running title lines for the top and bottom of even, odd, or all output pages. Initial value: .RT TOP ALL 1 ///PAGE &/
.RV	symbolname [='']	Read Variable: Allows user to assign a val- ue to a symbolname by entering it at the terminal in response to an interactive request made while SCRIPT/VS is processing the input file.
.SA		Save Status: Saves the current values and parameters of the formatting environment.
.SC		Single-Column Mode: Causes SCRIPT/VS to save the current column definition and format subsequent input lines in a single column. Causes a break.
.SE	symname[(n)] [LIB|OFF] [= value] [= SUBSTR str n1 n2] [= INDEX str1 str2]	Set Symbol: Defines a symbol name and assigns a value to it.
.SK	[1|v] [A] [C] [P]	Skip Lines: Specifies the amount of space to insert before the next text output line. No lines are inserted if the .SK occurs at the top of a page or column. Causes a break.
.SL	[v]	Set Line Spacing: Specifies the vertical distance between baselines of output lines. Default: Based on logical output device.

Figure 24. SCRIPT/VS Control Word Summary (Part 7 of 9)

Control Word	Parameters	Description
.SO	[DOC name] [PID naa] [OPR number] [RPW password] [DPW password]	STAIRS Output: Specifies information for STAIRS/VS CTF output.
.SP	[1|v] [A] [C] [P]	Space Lines: Specifies the amount of space to insert before the next text output line. The specified number of lines are inserted even when the .SP occurs at the top of a page or column. Causes a break.
.SS		Single-Space Mode: Causes subsequent output lines to be single-spaced.
.SU	[1|n|ON|OFF| input-line]	Substitute Symbol: Controls the substitution of symbols with their previously assigned values. SCRIPT/VS system symbol: &$SU Initial value: ON
.SV	[ON|OFF] [NOADD] [NOSTEM] [NUM]	Spelling Verification: Defines the start and functions of the SCRIPT/VS spelling verification function. Enabled with the SPELLCHK option.
.SX	[F|C] /left/fill/right/	Split Text: Produces an output line of three parts: "left" is aligned with the current left margin; "right" is aligned with the current right margin; "fill" is centered or repeated between the two strings. Causes a break.
.SY	input-line	System Command: SCRIPT/VS passes the input line to the host system for processing. SCRIPT/VS system symbol: &$RET
.TB	[ADD|CLR|DEL|SET] [h h h ...] [f/h f/h ...]	Tab Setting: Specifies the tab settings to be used when the input file is formatted. Causes a break. Default: 5 10 15 20 ... 75
.TC	n [name|/]	Table of Contents: Imbeds the table contents file (DSMUTTOC), which consists of table of contents entries automatically generated by the .Hn control words, and entries inserted by using the .PT [Put Table of Contents] control word. Use the TWOPASS option if the table of contents is not at the back of the document. Causes a break.
.TE	[1|n|ON|OFF]	Terminal Input: Allows user to enter lines interactively from the terminal when the file is formatted.
.TH	[input-line]	Then: Processes the "input-line" if the most recently executed .IF, .AN, or .OR control word was satisfied.
.TI	[s t ...]	Translate Input: Specifies character translations to be performed on input lines before SCRIPT/VS processing begins. Default: Identity

Figure 24. SCRIPT/VS Control Word Summary (Part 8 of 9)

Control Word	Parameters	Description
.TM	[v|+v|-v]	Top Margin: Specifies the amount of space in the top margin area. SCRIPT/VS system symbol: &$TM
.TR	[s t ...]	Translate Character: Specifies character translations to be performed on output. Default: Identity
.TS	s [/string/|OFF|IGNORE]	Translate String: translates a single character into a string.
.TU	[s t ...]	Translate Uppercase: Specifies character translations to be performed for capitalization. Default: a-z is mapped to A-Z.
.TY	input-line	Type On Terminal: Types the input line on the user's terminal during formatting.
.UC	[1|n|ON|OFF| input-line]	Underscore and Capitalize: UNDERSCORES AND CAPITALIZES one or more subsequent input lines.
.UD	{ON|OFF}	Underscore Definition: Determines whether blanks will be underscored when the .UC and .US control words are used.
.UN	[0|h|+h|-h]	Undent: Causes the next output line's indention to change: it is moved to the left of the current left margin. Causes a break.
.UP	[1|n|ON|OFF| input-line]	Uppercase: Prints one or more subsequent input lines in UPPERCASE characters.
.US	[1|n|ON|OFF| input-line]	Underscore: Prints one or more subsequent input lines with underscored characters.
.UW	[word ...]	Unverified Words: Generated by spelling verification for unverified words.
.WF	[1|n|ON|OFF| IMBED|ERASE| input-line]	Writes one or more input lines to the output file DSMUTWTF. IMBED: imbeds file DSMUTWTF. ERASE: erases file DSMUTWTF.
.WZ	[ON|OFF]	Widow Zone: Turns widow processing on or off.
.ZZ	[ON|OFF|PROG] nn ...	Diagnostic: Turns on or off the diagnostic trace function, and selects the type of data to be traced. Enabled with the DUMP option.

Figure 24. SCRIPT/VS Control Word Summary (Part 9 of 9)

.BR [Break]
.BX [Box]
.CB [Column Begin]
.CC [Conditional Column Begin] [1]
.CD [Column Definition]
.CE [Center]
.CL [Column Line Length]
.CO [Concatenate Mode]
.CP [Conditional Page Eject] [1]
.EP [Even Page Eject]
.FI [Fill Mode]
.FO [Format Mode]
.HN [Headnote]
.H1 [Head Level 1]
.H2 [Head Level 2]
.H3 [Head Level 3]
.H4 [Head Level 4]
.H5 [Head Level 5]
.H6 [Head Level 6]
.IE [Index Entry]
.IL [Indent Line]
.IN [Indent] [2]
.IR [Indent Right] [2]
.IX [Index]
.JU [Justify Mode]
.LB [Leading Blank]
.LL [Line Length]
.LT [Leading Tab]
.MC [Multicolumn Mode] [1]
.NB [No Balancing]
.NC [No Concatenation]
.NF [No Formatting]
.NJ [No Justification]
.OF [Offset]
.OP [Odd Page Eject]
.PA [Page Eject]
.PM [Page Margins]
.PP [Paragraph Start]
.QU [Quit]
.RD [Read Terminal]
.RF [Running Footing]
.RH [Running Heading]
.RI [Right Adjust]
.SC [Single Column Mode]
.SK [Skip]
.SO [STAIRS/VS Output]
.SP [Space]
.SX [Split Text]
.TB [Tab Setting]
.TC [Table of Contents]
.UN [Undent]
.WZ [Widow Zone]

[1] The break occurs only if the control word performs its function. These control words may do nothing if the function is not needed.

[2] These control words ordinarily cause a break. Their functions will be performed without a break if the NOBREAK parameter is specified.

Figure 25. Control Words That Cause a Break: When concatenation is on (see the .FO [Format Mode] control word), words from input lines are rearranged on output lines to make each column line as full as possible. This process is inhibited for the current line if any of these control words is encountered.

.BM [Bottom Margin]
.FM [Footing Margin]
.FS [Footing Space]
.H1 [Head Level 1] [1]
.HM [Heading Margin]
.HS [Heading Space]
.LL [Line Length]
.PL [Page Length]
.PN [Page Numbering Mode]
.RF [Running Footing]
.RH [Running Heading]
.RT [Running Title]
.TM [Top Margin]

[1] .H1 causes a page eject by default. The .DH [Define Head Level] control word allows you to redefine the meaning of the .H1 control word.

Figure 26. Control Words That Take Effect On the Next Page: These control words take effect on the next output page to be started. If no data has yet been placed on the first page of the document, or the previous page was ended with a .PA NOSTART control word, the first, or next, page has not yet been started, and these control words can take effect on this page.

.BM [Bottom Margin]	.IX [Index]
.BT [Bottom Title]	.KP [Keep]
.CB [Column Begin]	.LL [Line Length]
.CC [Conditional Column Begin]	.MC [Multicolumn Mode]
.CD [Column Definition]	.OB [Odd Page Bottom Title]
.CP [Conditional Page Eject]	.OP [Odd Page Eject]
.DI [Delay Imbed]	.OT [Odd Page Top Title]
.EB [Even Page Bottom Title]	.PA [Page Eject]
.EF [End of File]	.PL [Page Length]
.EP [Even Page Eject]	.PN [Page Numbering Mode]
.ET [Even Page Top Title]	.RD [Read Terminal]
.FL [Float]	.RN [Reference Numbers]
.FM [Footing Margin]	.RT [Running Title]
.FN [Footnote]	.SC [Single Column Mode]
.FS [Footing Space]	.SK [Skip]
.FT [Footing]	.SP [Space]
.HE [Heading]	.TC [Table of Contents]
.HM [Heading Margin]	.TM [Top Margin]
.HN [Headnote]	.TT [Top Title]
.HS [Heading Space]	

Figure 27. Control Words That End a Keep, Float, Running Heading or Footing, or Footnote: If found, a message is issued and the Keep, Heading or Footing, or Footnote is terminated before the control word is processed.

Note: .RF and .RH are disallowed in keeps, floats, and footnotes. .KP, .FL, and .FN are disallowed in running headings and footings. Only the 'P' (Page) option of the .SK [Skip] and .SP [Space] control words ends a keep.

.AA [Associate APF]	.MS [Macro Substitution]
.AN [And]	.OR [Or]
.AP [Append]	.PI [Put Index]
.CM [Comment]	.PN [Page Numbering Mode]
.CS [Conditional Section]	.PP [Paragraph Start]
.CW [Control Word Separator]	.PT [Put Table of Contents]
.DC [Define Character]	.QQ [Quick Quit]
.DD [Define Data File-id]	.QU [Quit]
.DF [Define Font]	.RF [Running Footing]
.DL [Dictionary List]	.RH [Running Heading]
.DM [Define Macro]	.RV [Read Variable]
.DU [Dictionary Update]	.SE [Set Symbcl]
.EL [Else]	.SU [Substitute Symbol]
.FL [Float]	.SV [Spelling Verification]
.GO [Goto]	.SY [System Command]
.GS [GML Services]	.TE [Terminal Input]
.IF [If]	.TH [Then]
.IM [Imbed]	.TY [Type on Terminal]
.IT [Input Trace]	.UD [Underscore Definition]
.LI [Literal]	.WF [Write To File]
.LY [Library]	

Figure 28. Control Words Within a Running Heading or Footing: These control words are processed only once, during a running heading or footing definition. All other control words are saved as part of the heading or footing definition, and processed each time the definition is formatted for a new page.

.BX [Box]	.IX [Index]
.CB [Column Begin]	.KP [Keep]
.CC [Conditional Column Begin]	.PI [Put Index]
.CD [Column Definition]	.PP [Paragraph Start]
.CP [Conditional Page Eject]	.PT [Put Table of Contents]
.CT [Continued Text]	.RD [Read Terminal]
.FL [Float]	.SK [Skip]
.FN [Footnote]	.SO [STAIRS/VS Output] DOC
.HW [Hyphenate Word]	.SP [Space]
.H0 - .H6 [Head Level 0 - 6]	.SX [Split Text]
.IE [Index Entry]	.TC [Table of Contents]

Figure 29. Control Words That Start the Page: If the page is not already started, either because no text has yet been formatted for page one, or because the previous page was ended with .PA NOSTART, any of these control words will cause the page to be started.

Obsolete Control Word	SCRIPT/VS Equivalent Control Word
.BT	.RT [Running Title] BOTTOM
.CO	.FO [Format Mode]
.CW	.DC [Define Character] CW
.EB	.RT [Running Title] BOTTOM EVEN
.EP	.PA [Page Eject] EVEN
.ET	.RT [Running Title] TOP EVEN
.FI	.FO [Format Mode] ON
.FT	.RT [Running Title] BOTTOM
.HE	.RT [Running Title] TOP EVEN
.HN	.RH [Running Heading]
.JU	.FO [Format Mode]
.LS	.SL [Set Line Space]
.NB	.BC [Balance Columns] OFF
.NC	.CO [Concatenate Mode] OFF
.NF	.FO [Format Mode] OFF
.NJ	.JU [Justify Mode] OFF
.OB	.RT [Running Title] BOTTOM ODD
.OP	.PA [Page Eject] ODD
.OT	.RT [Running Title] TOP ODD
.PS	.DC [Define Character] PS
.SF	.BF [Begin Font]
.TT	.RT [Running Title] TOP

Figure 30. Obsolete Control Words: SCRIPT/VS continues to recognize and support these control words, but their functions have been subsumed by more general control words as indicated.

Device Type	Initial Values .TM	.HS	.HM	.BM	.FS	.FM	.LL
1403	6	1	2	6	1	2	6i
2741	6	1	2	6	1	2	6i
3270	6	1	2	6	1	2	6i
3800	3	1	2	3	1	2	6i

.BM [Bottom Margin]	.HS [Heading Space]
.FM [Footing Margin]	.LL [Line Length]
.FS [Footing Space]	.PL [Page Length] [1]
.HM [Heading Margin]	.TM [Top Margin]

[1] For 3800-type devices, Page Length does not include 1/2 inch at the top and bottom of each page. This area is inaccessible.

Figure 31. Control Word Values Based On the Logical Device: The initial and default values for these control words vary, depending upon the specified or implied logical output device.

Logical Device Type	Real Device Type	Lines per Inch	Page Size (inches)		Line Length[1]	Page Length[2]
			Width	Depth	(bytes)	(lines)
TERM	([3])	6	8-1/2	11	60/132	66/144
2741	2741	6	8-1/2	11	60/132	66/144
3270	3270	6	8-1/2	11	60/132	66/144
1403N6	1403	6	8-1/2	11	60/85	66/144
1403N8	1403	8	8-1/2	11	60/85	88/192
1403W6	1403	6	13-1/2	11	60/132	66/144
1403W8	1403	8	13-1/2	11	60/132	88/192
1403W6S	1403	6	13-1/2	8-1/2	60/132	51/144
1403W8S	1403	8	13-1/2	8-1/2	60/132	68/192
1403SW [4]	1403	6	8-1/2	11	72/90	66/102
STAIRS	1403	6	13-1/2	11	60/132	66/144
3800N6	3800	6	8-1/2	11	60/85	60
3800N8	3800	8	8-1/2	11	60/85	80
3800N12	3800	12	8-1/2	11	60/85	120
3800W6	3800	6	13-1/2	11	60/136	60
3800W8	3800	8	13-1/2	11	60/136	80
3800W12	3800	12	13-1/2	11	60/136	120
3800N6S	3800	6	11	8-1/2	60/110	45
3800N8S	3800	8	11	8-1/2	60/110	60
3800N12S	3800	12	11	8-1/2	60/110	90
3800W6S	3800	6	13-1/2	8-1/2	60/136	45
3800W8S	3800	8	13-1/2	8-1/2	60/136	60
3800W12S	3800	12	13-1/2	8-1/2	60/136	90

[1] Line lengths are given as "default/maximum" in 10-pitch characters. For the IBM 3800 Printer, 12-pitch and 15-pitch fonts have values 20% and 50% greater, respectively. The potential maximum line length includes binding. The text fonts (as shown in Figure 39 on page 364) contain 10, 12, and 15 pitch blank characters. If these fonts are used, you should allow for an output record length, in bytes, that is 15 times the length (in inches) of your longest print line (including the binding).

[2] Default and maximum page lengths are identical for 3800 devices.

[3] The physical device type corresponding to the TERM logical device may be either 2741 or 3270, depending upon the actual terminal type.

[4] This is a 12-pitch device, as opposed to the normal 10-pitch 1403.

Figure 32. SCRIPT/VS Logical Device Characteristics

.Hn Control Word							
keys	H0	H1	H2	H3	H4	H5	H6
SKBF	0	0	3	3	3	1	1
SPAF	0	5	2	2	2	0	0
TCIN	0	0	0	2	4	6	8
TO	X						
TC	X	X	X	X			
TS		X					
US		X	X		X	X	X
UP		X	X	X		X	
OJ		X					
PA		X					
SECT		X					
BR		X	X	X	X		

EasySCRIPT Head Levels							
keys	H0	H1	H2	H3	H4	H5	H6
SKBF	0	0	3	3	3	1	1
SPAF	0	5	3	3	3	0	0
TCIN	0	0	0	2	4	6	8
TO	X						
TC	X	X	X	X	X		
TS		X					
US		X	X		X		X
UP		X	X	X		X	
OJ		X					
PA		X					
SECT		X					
BR		X	X	X	X		

where the "keys" are:

SKBF:	number of line skips before the head.
SPAF:	number of line spaces after the head.
TCIN:	amount of indention for table of contents entry.
TO:	table of contents entry only; no heading in text.
TC:	table of contents entry.
TS:	line space before table of contents entry.
US:	head is underscored.
UP:	head is capitalized.
OJ:	head is outjustified.
PA:	page eject before head.
SECT:	section break around head.
BR:	break after head.

Figure 33. Summary of Head Level Characteristics: This table lists the default characteristics of the .Hn [Head Level n] control words and EasySCRIPT &Hn tags. The .DH [Define Head Level] control word allows you to redefine any of these Head Levels to suit your needs.

Active Environment			
Parameter	Control Word	Initial Setting	Symbol
Capitalization	.UC, .UP	OFF	
Column balancing	.BC	ON	
Continuation Character	.DC CONT	(null)	&$CONT
Control Word Separator	.DC CW, .CW	";"	&$CW
Current font	.BF, .DF, .PF	([1])	
Column definition	.CD	Single column	
Centering [2]	.CE	OFF	
Column width	.CL	Line length	&$CL
Concatenation	.FO	ON	
Conditional sections	.CS	INCLUDE	
Font save stack	.BF, .PF	empty	
Line spacing	.SS, .DS, .SL	Single spacing	
Format mode	.FO	ON	
GML tag delimiter	.DC GML	":"	&$GML
Indention [3]		0	&$IN
Justification	.FO	ON	
Page number symbol	.DC PS, .PS	"&"	&$PS
Revision code, adjust	.RC	OFF, 2	
Right adjustment	.RI	OFF	
Right indention	.IR	0	&$IR
Spelling verification	.SV	OFF	
Tab setting	.TB	5 10 15 ... 80	
Terminal input [2]	.TE	OFF	
Underscoring	.UC, .UD, .US	OFF	

Page Environment			
Parameter	Control Word	Initial Setting	Symbol
Bottom margin	.BM	([1])	&$BM
Footing margin	.FM	([1])	&$FM
Footing space	.FS	([1])	&$FS
Heading margin	.HM	([1])	&$HM
Heading space	.HS	([1])	&$HS
Hyphenation	.HY	OFF	
Page length	.PL	([1])	&$PL
Page numbering mode	.PN	Arabic	
Macro substitution [2]	.MS	OFF	
Symbol substitution [2]	.SU	ON	&$SU
Top margin	.TM	([1])	&$TM

Translate Tables			
Parameter	Control Word	Initial Setting	Symbol
Input translation	.TI	Identity	
Output translation	.TR	Identity	

[1] These parameters' initial settings are based upon the logical output device.

[2] The number of lines remaining, or ON or OFF, is saved.

[3] The composite current indention is determined from the .IN, .IR, .IL, .UN and .OF control word values. These values are individually saved.

Figure 34. The SCRIPT/VS Formatting Environment: The .SA [Save Environment], .KP [Keep], .FL [Float], and .FN [Footnote] control words preserve the active environment. The .SA [Save Environment] control word also preserves the page environment and translate tables.

Date and Time [1]		
Symbol	Description	Value
&SYSYEAR	Year of the century	00-99
&SYSMONTH	Month of the year	01-12
&SYSDAYOFM	Day of the month	01-31
&SYSDAYOFW	Day of the week	1-7 ("1" is Sunday)
&SYSDAYOFY	Day of the year	001-366
&SYSHOUR	Hour of the day	00-23
&SYSMINUTE	Minute of the hour	00-59
&SYSSECOND	Second of the minute	00-59

Output Device Characteristics		
Symbol	Description	Value
&$LDEV	Logical output device [2]	1-8 characters
&$OUT	Output destination	TERM, PRINT, FILE
&$PDEV	Physical output device	1403, 2741, 3270, 3800

SCRIPT Command Options		
Symbol	Description	Value
&$BE	Even bind [3] [4]	0-
&$BO	Odd bind [3] [4]	0-
&$CHAR(n)	Fonts [5]	1-4 characters
&$INDX	Indexing [6]	0, 1
&$LIB	Macro library available [6]	0, 1
&$PARM	Command options [7]	8-256 characters
&$SYS	Environment	CMS, TSO, VS2, VS1, DOS, CICS
&$TWO	TWOPASS option in effect [6]	0, 1
&$UNF	Unformatted output [6]	0, 1

[1] These symbols may contain leading zeros. They can be eliminated with a .SE [Set Symbol] control word: ".se SYSHOUR = &SYSHOUR + 0".

[2] Set by the DEVICE option of the SCRIPT command.

[3] Set by the BIND option of the SCRIPT command and the .PM [Page Margins] control word.

[4] The system symbol values are represented in character spaces, regardless of the space units used in setting them. The maximum value depends upon the logical output device.

[5] Set by the CHARS option of the SCRIPT command. This is a symbol array; element 0 contains the number of fonts specified and elements 1, 2, ... contain the names of the fonts specified.

[6] "1" indicates the command option was specified; "0" indicates it was not specified.

[7] This is the SCRIPT command options list. In CMS, the command options list is tokenized (divided into eight character fields separated by blanks and parentheses) and truncated at 32 tokens (256 characters).

Figure 35. SCRIPT/VS System Symbol Names (Part 1 of 2)

Page Characteristics		
Symbol	Description	Value
&$BM	Bottom margin (.BM) [8]	0-
&$CL	Column width (.CL) [9]	0-
&$FM	Footing margin (.FM) [8]	0-
&$FS	Footing space (.FS)	0-6
&$HM	Heading margin (.HM) [8]	0-
&$HS	Heading space (.HS)	0-6
&$IN	Left indention [9]	0-
&$IR	Right indention [9]	0-
&$LC	Internal line counter [8] [10]	0-
&$LL	Line length (.LL) [9]	0-
&$OF	Offset [9]	0-
&$PL	Page length (.PL) [8]	0-
&$TM	Top margin (.TM) [8]	0-

SCRIPT/VS Formatter Parameters		
Symbol	Description	Value
&$BS	Backspace Character	hexadecimal 16
&$CONT	Continuation character [11]	one character
&$CW	Control word separator [11]	(default: ";")
&$C256	Identity vector	256 characters
&$EGML	GML end-tag delimiter [11]	(default: "::")
&$FNAM	Current input file name	eight characters
&$GML	GML tag delimiter [11]	(default: ":")
&$KP	Keep in effect	ON, OFF
&$LNUM	Last line number read	0-
&$MCS	GML markup/content separator	(default: ".")
&$PN	Page number [12]	1-
&$PS	Page number symbol [11]	(default: "&")
&$RB	Required Blank [11]	(default: hexadecimal 41)
&$RET	Return code from .SY [13]	0-
&$SU	Symbol substitution enabled	ON, OFF
&$TAB	Tab Character	hexadecimal 05
&$TAG	Name of last GML tag found	(any valid tag name)
&$TAGD	Delimiter of last tag found	(&$GML or &$EGML)

[8] These values are represented in line spaces, regardless of the space units used in setting them. The maximum value depends upon the logical output device.

[9] The values of these symbols are represented in character spaces, regardless of the space units used in setting them. The maximum value depends upon the logical output device.

[10] The value of the symbol &$LC is the number of lines remaining in the current column, excluding unplaced keeps, floats, footnotes, and widow zones.

[11] Set by the .DC [Define Character] control word.

[12] &$PN contains the numeric portion of the current page number. The page number as substituted can be obtained with the control word ".se x = &".

[13] In CMS, any possible return code value. In TSO, "0" to indicate the command was stacked for execution after SCRIPT/VS terminates. In ATMS-III, "0" to indicate the control word was ignored. In batch, "-3" to indicate that the .SY [System Command] control word is not supported.

Figure 35. SCRIPT/VS System Symbol Names (Part 2 of 2)

Attribute	Function
&a'	Converts a numeric character string[1] to a "base-26" lowercase alphabetic "number."
&A'	Converts a numeric character string[1] to a "base-26" uppercase alphabetic "number."
&E'	Verifies the existence of a symbol; the value is 1 if the symbol has been set; 0 if not.
&L'	Yields the length of a character string[1].
&r'	Converts a numeric character string[1] into a lowercase roman numeral.
&R'	Converts a numeric character string[1] into an uppercase roman numeral.
&T'	Yields the type of the current value of a symbol. The type is either "N" for numeric or "C" for character.
&U'	Converts a lowercase character string to uppercase.
&V'	Yields the current value of a symbol.
&X'	Yields the hexadecimal string represented by the character string[1].

[1] The character string may be the value of a symbol.

Figure 36. Attributes of a Symbol's Value

Code	Character	Code	Character	Code	Character
05	Tab	4E	+	6C	%
11	Special Blank [1]	4F	|	6D	_
12	Special Blank [1]	5A	!	6F	?
13	Special Blank [1]	5B	$	7A	:
16	Backspace	5C	*	7E	=
40	Blank	5D	)	7F	"
41	Required Blank [2]	5E	;	8B	{
4B	. (Period)	5F	¬	9B	}
4C	<	61	/	AD	[
4D	(	6B	,	BD	]

[1] Special Blanks are used for justification in documents formatted for the 3800 Printing Subsystem.

[2] The required blank is a blank which cannot have space added to it during justification. Its value may be changed with the .DC [Define Character] control word.

Figure 37. Characters that Delimit Words for Spelling Verification: These default characters can be changed with the .DC [Define Character] control word, which accepts either single characters or 2-digit hexadecimal character codes.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00																	0F
10																	1F
20																	2F
30																	3F
40											¢	.	<	(	+	|	4F
50	&										!	$	*	)	;	¬	5F
60	-	/										,	%	_	>	?	6F
70											:	#	@	'	=	"	7F
80		a	b	c	d	e	f	g	h	i		{	≤	(	+	+	8F
90		j	k	l	m	n	o	p	q	r		}	□	)	±	■	9F
A0	—	°	s	t	u	v	w	x	y	z		└	┌	[	≥	•	AF
B0	0	1	2	3	4	5	6	7	8	9		┘	┐	]	≠	—	BF
C0		A	B	C	D	E	F	G	H	I							CF
D0		J	K	L	M	N	O	P	Q	R							DF
E0			S	T	U	V	W	X	Y	Z							EF
F0	0	1	2	3	4	5	6	7	8	9							FF
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Figure 38. TN Translate Table For the 1403 Printer

Text Fonts	Highlight Fonts	Special Fonts
GT10 Gothic (10-pitch) GT12 Gothic (12-pitch) GT15 Gothic (15-pitch)	GB10 Gothic Bold GB12 Gothic Bold GI12 Gothic Italic	GR10 Gothic Reverse GP12 Proportional
ST10 Serif (10-pitch) ST12 Serif (12-pitch) ST15 Serif (15-pitch)	SI10 Serif Italic SI12 Serif Italic SB12 Serif Bold	RT10 Roman Text SO12 Serif Overstruck

Figure 39. Complete 3800 Fonts Provided With SCRIPT/VS: Each font is a complete set of special, upper- and lowercase characters. Any two of these fonts may be specified with the CHARS option of the SCRIPT command.

10-pitch Fonts	
GS10	Gothic
GF10	Gothic Folded
GU10	Gothic Underscored
TU10	Text Underscored [1]

12-pitch Fonts	
GS12	Gothic
GF12	Gothic Folded
GU12	Gothic Underscored

15-pitch Fonts	
GS15	Gothic
GSC	Gothic Condensed
GF15	Gothic Folded
GFC	Gothic Folded Condensed
GU15	Gothic Underscored
GUC	Gothic Underscored Condensed
DUMP	Condensed DUMP [2]

Format Fonts	
FM10	Format 10-pitch
FM12	Format 12-pitch
FM15	Format 15-pitch

10-pitch Katakana Fonts	
2773	Gothic and Katakana
2774	Gothic and Katakana [2]
KN1	Gothic and Katakana [2]

3211 Print Trains	
A11	Gothic 10-pitch
G11	Gothic 10-pitch
H11	Gothic 10-pitch
P11	Gothic 10-pitch
T11	Text 10-pitch [1]

1403 Print Trains	
AN	Gothic 10-pitch
GN	Gothic 10-pitch
HN	Gothic 10-pitch
PCAN	Gothic 10-pitch
PCHN	Gothic 10-pitch
PN	Gothic 10-pitch
QN	Gothic 10-pitch
QNC	Gothic 10-pitch
RN	Gothic 10-pitch
XN	Gothic 10-pitch
YN	Gothic 10-pitch
SN	Text 10-pitch [1]
TN	Text 10-pitch [1]

10-pitch OCR Fonts	
AOA	Gothic and OCR-A
AOD	Gothic and OCR-A
AON	Gothic and OCR-A
OAA	Gothic and OCR-A
ODA	Gothic and OCR-A
ONA	Gothic and OCR-A
BOA	Gothic and OCR-B
BON	Gothic and OCR-B
OAB	OCR-B
ONB	Gothic and OCR-B

[1] This is an upper- and lowercase font which closely resembles the ST10 SCRIPT/VS font. It counts as two fonts when combined with other fonts in the CHARS option of the SCRIPT command.

[2] This font contains more than 64 characters. It counts as two fonts when combined with other fonts in the CHARS option of the SCRIPT command.

Figure 40. Fonts Supplied With the 3800 Printing Subsystem: These are all uppercase-only fonts, unless otherwise marked. Any four fonts (except those otherwise marked) of <u>identical pitch</u> may be specified with the CHARS option of the SCRIPT command.

APPENDIX B. DEVICE AND FONT TABLE MAINTENANCE

The logical devices and font tables provided with SCRIPT/VS are intended to be comprehensive enough to serve the needs of most general users. When no existing logical device or font table definition seems suitable, the procedures described in this appendix can be used to modify the supplied tables, or to create new ones.

LOGICAL DEVICE TABLE MAINTENANCE

SCRIPT/VS bases its formatting of a document on the characteristics of a specified (or implied) logical output device. SCRIPT/VS takes into account the characteristics of the physical device, as well as dynamically changing characteristics: font, lines per inch, and form size.

The combination of these fixed (physical) characteristics and changeable characteristics is called the logical output device, which corresponds to the "setup" of a physical output device.

The module DSMLPLDT contains one logical device table (LDT) for each logical output device. Each LDT is created by an LDT macro and is mapped by DSMSLDT.

Updating a Logical Device Table (LDT)

Use the following procedure to add or change an LDT entry:

1. Obtain a listing of the DSMLPLDT module from your library.

2. Obtain the source code for the DSMLPLDT module and add a new LDT macro that describes the new logical output device

   - or -

   Modify an existing LDT specification macro that describes the logical output device whose characteristics you want to change. The macro's field names are described in detail in "LDT Field Descriptions."

3. Assemble the DSMLPLDT module to include the changes you've made.

4. Link the newly-assembled version of the DSMLPLDT module with the rest of SCRIPT/VS, as described in the Program Directory.

LDT Field Descriptions

Each logical device table (LDT) macro is specified as:

LDT	LD=name,	Logical device name
	PD=device name,	Physical device identifier
	DF=font name	Default font identifier
	[,DLL=device units|60]	Default line length
	[,DOTxx=value]	Page margin values[48]
	[,DPL=device units|66]	Default page length
	[,FSS=font space|1]	Font storage size
	[,HS=device units|1]	Horizontal space unit
	[,LPI=6|8|12]	Lines per inch
	[,MLL=device units|60]	Maximum line length
	[,MPL=device units|66]	Maximum page length
	[,P=10|12|15]	Pitch
	[,VS=device units|1]	Vertical space unit

Page dimensions (MPL, DPL, MLL, DLL) and space units (HS and VS) are represented as device units: the printer's physical unit of resolution in the relevant direction. Vertically, this is one line for all printers. Horizontally, this is one character space for all printers except the 3800 Printing Subsystem. The 3800 Printing Subsystem's unit of horizontal resolution is the pel, a space of 1/180th of an inch. The HS space unit for the 3800 Printing Subsystem, therefore, could be 18 (for a 10-pitch horizontal space), 15 (for a 12-pitch space), or 12 (for a 15-pitch space).

The content and meaning of each LDT field that you can specify with the LDT are shown below.

LDT Field	Description (Required Parameters)
LD	Logical device name: 1 to 8 alphameric characters.
PD	Physical device name: 4 to 8 alphameric characters. The values recognized are 1403, 2741, 3270, and 3800.
DF	Name of the default font, 1 to 4 alphameric characters The default font is used when the CHARS option of the SCRIPT command is not specified.

LDT Field **Description (Optional Parameters)**

DLL The default line width in device units. This value is used until reset with the .LL [Line Length] control word. DLL cannot be greater than MLL. Default = 60.

DLL and DPL establish the default page size, which cannot be larger than the form size.

DOTxx These keywords can be used to override the default page margin parameters:

Keyword	Overrides Default
DOTBM	Bottom margin
DOTTM	Top margin
DOTHM	Heading margin
DOTFM	Footing margin
DOTHS	Heading space
DOTFS	Footing space

Note: For 3800-type logical devices, DOTBM and DOTTM are ignored. Instead,

Bottom margin = Footing margin + Footing space

Top margin = Heading margin + Heading space

DPL The default page length in device units. This value is used until reset with the .PL [Page Length] control word. DPL cannot be greater than MPL. Default = 66.

FSS The font storage size of the device. This value is used to ensure that all fonts specified with the CHARS option of the SCRIPT command will fit in the physical device. Refer to the FS parameter of the FIB macro. Default = 1.

HS The number of device units in one "horizontal space unit" (the width of a character in the default font). This value is used to resolve a control word's parameter that specifies a horizontal space or displacement, but does not specify a unit of measurement (for example, .IN 2).

---

[48] xx = TM (Top Margin), HS (Heading Space), HM (Heading Margin), BM (Bottom Margin), FS (Footing Space), or FM (Footing Margin). DOTxx can be specified repeatedly (for example: "DOTTM=3,DOTBM=3").

LPI	Lines per inch. A decimal number: 6, 8, or 12. Default = 6.
MLL	The maximum number of horizontal device units in the printable portion of the page. Default = 60. MLL and MPL establish the form size for the logical output device (that is, the size of its physical page.)
MPL	The maximum number of vertical device units in the printable portion of the page. Default = 66.
P	Pitch, or number of equal-width characters per inch. For monospaced devices only. A decimal number: 10, 12, or 15. Default = 10.
VS	The number of device units in one "vertical space unit" (that is, the vertical space of a print line). This value is used to resolve a control word's parameter that specifies a vertical space or displacement, but does not specify a unit of measurement (for example, .SP 2). The blank-character codes are used for line justification when the output is in a font that includes pseudo-blanks. All fonts to be used in a mixed-pitch environment must include the proportional-spaced blank character codes in their character arrangement tables.

Default Values for Logical Output Devices

When you specify values for the various fields of the LDT, SCRIPT/VS will use those values to derive the following defaults:

Parameter	**Default derived from**
Top margin	6 times VS (6 vertical line spaces) or, for the 3800 Printing Subsystem, 3 times VS or 0, if page length is less than 15.
Heading Space	VS (1 vertical line space) or 0, if page length is less than 15.
Heading Margin	2 times VS (2 vertical line spaces) or 0, if page length is less than 15.
Bottom Margin	6 times VS (6 vertical line spaces) or for the 3800 Printing Subsystem, 3 times VS or 0, if page length is less than 15.
Footing Space	VS (1 vertical line space) or 0, if page length is less than 15.
Footing Margin	2 times VS (2 vertical line spaces) or 0, if page length is less than 15.

FONT TABLE MAINTENANCE

When formatting documents for the 3800 Printing Subsystem, SCRIPT/VS makes use of font tables for each of the fonts named in the CHARS option of the SCRIPT command. Each font table describes the font in terms of its name, pitch, and the width of each character.

The module DSMAFFIB contains one font information block (FIB) for each known font. The FIBs are created by the FIB macro and are mapped by DSMFIBD.

```
*             0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
* 00-3F:
GP12WT DC   H'15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15' ----------------
       DC   H'15,18,15,12,15,15,15,15,15,15,15,15,15,15,15,15' ----------------
       DC   H'15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15' ----------------
       DC   H'15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15' ----------------
* 40-7F:
       DC   H'15,15,15,15,15,15,15,15,15,15,15,12,15,12,15,12'  ---------¢.<(+|
       DC   H'18,15,15,15,15,15,15,15,15,15,12,15,15,12,12,15' &---------!$*);-
       DC   H'15,18,15,15,15,15,15,15,15,15,15,12,18,18,15,15' -/---------,%_>?
       DC   H'15,15,15,15,15,15,15,15,15,15,12,18,18,12,15,15' ----------:#@'="
* 80-BF:
       DC   H'15,15,15,15,15,15,12,15,15,12,15,12,15,12,12,15' -abcdefghi------
       DC   H'15,12,15,12,18,15,15,15,15,15,15,12,15,12,15,12' -jklmnopqr------
       DC   H'12,15,15,12,15,15,18,15,15,15,15,15,15,12,15,12' --stuvwxyz------
       DC   H'12,12,12,12,12,12,12,12,12,12,15,15,15,12,15,15' ----------------
* C0-FF:
       DC   H'15,15,15,15,15,15,15,15,15,12,15,15,15,15,15,15' -ABCDEFGHI------
       DC   H'15,12,15,12,18,15,15,15,15,15,15,15,15,18,15,15' -JKLMNOPQR------
       DC   H'18,15,15,15,15,15,18,15,15,15,15,15,15,15,15,15' --STUVWXYZ------
       DC   H'15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15' 0123456789------
*
*             0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
```

Figure 41. Example of a Font Width Table: GP12 is a 12-pitch proportional-spaced font with psuedo-blanks.

Updating the Font Table (FIB)

Use the following procedure to add or change a font table:

1. Obtain a listing of the DSMAFFIB module from your library.
2. Code a FIB macro that describes the new font.

   - or -

   Modify the FIB macro that describes the font whose characteristics you want to change. (The macro's field names are described in detail in "FIB Field Descriptions," which follows.)
3. Assemble the DSMAFFIB module to include the changes you've made.
4. Link the newly-assembled version of the DSMAFFIB module with the rest of SCRIPT/VS. (See the Program Directory for details.)

FIB Field Descriptions

Each font information block (FIB) macro is specified as:

```
FIB  font-name,              Font name
     width-table,            Character width table
     box-vector              Box character vector
     [,PITCH=10|12|15]       Pitch
     [,CODE=M|MB|PB]         Font type code
     [,FS=font-size|1]       Font size
```

The content and meaning of each font table (FIB) field that you can specify with the FIB macro are shown below:

FIB Field	Description (Required parameters)
font-name	The font name: 1 to 8 characters.

width-table The address of the font's width table. The width table contains a halfword entry for each of the 256 character codes. Each entry specifies the width of the character as a binary number of device units. See Figure 41 for an example of the width table.

box-vector The address of the box character set vector. The vectors defined are:

DSMBXAPL:	│	─	┼	┌	┐	└	┘	┬	├	┤	┴
DSMBXGPC:	│	─	┼	┌	┐	└	┘	─	│	│	─
DSMBXSWC:	│	─	┼	┌	┐	└	┘	┬	├	┤	┴
DSMBXTNC:	│	─	┼	┌	┐	└	┘	─,┐	│,┌	│,┐	─,└
DSMBXTST:	│	-	+	1	2	3	4	A	B	C	D
DSMBXVAN:	│	-	+	+	+	+	+	+	+	+	+
DSMBX32A:	│	─	┼	┌	┐	└	┘	┬	├	┤	┴
DSMBX32T:	│	─	┼	┌	┐	└	┘	┬	├	┤	┴
DSMBX38C:	│	─	┼	┌	┐	└	┘	┬	├	┤	┴

Note that the intersection characters in the DSMBXTNC vector are composites, formed by overstriking the two characters shown.

FIB Field **Description (Optional parameters)**

PITCH The font's pitch (characters per horizontal inch); either 10, 12, or 15.

CODE A 1- or 2-character code to specify the font type:

M Monospace font

MB Monospace font with special blanks

PB Proportional-spaced font with special blanks

Special blanks are described in "Appendix E. Formatting Considerations for the 3800 Printing Subsystem" on page 385.

FS Font size. The sum of the values of the FS parameters of all fonts specified with the CHARS option of the SCRIPT command may not exceed the value of the FSS parameter of the logical device specified with the DEVICE option.

Fonts Provided With SCRIPT/VS

The fonts provided with SCRIPT/VS for use with the 3800 Printing Subsystem are listed and illustrated in "Appendix D. Fonts Supplied with SCRIPT/VS" on page 377.

3800 Printing Subsystem Fonts Supported By SCRIPT/VS

The fonts provided with the 3800 Printing Subsystem are listed in Figure 39 on page 364. For details on creating a character arrangement table and its corresponding character set, see IBM 3800 Printing Subsystem Programmer's Guide.

APPENDIX C. STEM PROCESSING

During stem processing, SCRIPT/VS removes prefixes and suffixes in order to obtain the stem of a word that it is trying to hyphenate or verify the spelling of. The following descriptions summarize, by language the prefixes and suffixes SCRIPT/VS checks for during this process.

ENGLISH PREFIXES AND SUFFIXES

SCRIPT/VS checks for the following prefixes during stem processing:

ANTI	ANY	BACK	COUNTER	CROSS	DE
DIS	DOWN	EN	FORE	IN	INTER
INTRA	KILO	MACRO	MEGA	MICRO	MILLI
MINI	MIS	MULTI	NON	OUT	OVER
PRE	PRO	RE	SEMI	SOME	SUB
SUPER	TELE	TRANS	UN	UNDER	UP

SCRIPT/VS also checks for these seven suffixes:

' (apostrophe)	S	ED	AL	ALLY
ING	ION			

FRENCH PREFIXES AND SUFFIXES

There are two types of French[49] prefixes that SCRIPT/VS checks during stem processing. They are: contractions that are the result of elision processing; and grammatical prefixes. The following are the contractions that SCRIPT/VS checks for:

D' (DE)	J' (JE)	L' (LE/LA)	S' (SE/SI)
N' (NE)	M' (ME)	JUSQU' (JUSQUE)	LORSQU' (LORSQUE)
QU' (QUE)	T' (TE)	PUISQU' (PUISQUE)	QUOIQU' (QUOIQUE)

SCRIPT/VS also checks for these grammatical prefixes:

INTER	ENTRE	CONTRE	TRANS	SUR	ANTI
DE(S)	EN	EM	IN	IM	RE
REM	REN	RES	REDE		

The French suffixes that SCRIPT/VS checks for are:

ERAI	ERAS	ERA	ERONS	EREZ
ERONT	ERAIS	ERAIT	ERIEZ	ERIONS
ERAIENT	IRAI	IRAS	IRA	IRONS
IREZ	IRONT	IRAIS	IRAIT	IRIONS
IRAIENT	OSITION(S)	ATION(S)	ATEUR(S)	ATRICE(S)
ATIF(S)	ATIVE(S)	ATIVEMENT	IVE(S)	IVEMENT
IONS	SIONS	TIONS	INS	INT
I/NMES	I/NTES	INRENT	IS	IT
I/MES	I/TES	IRENT	US	UT
U/MES	U/TES	URENT	ISSE	ISSES
ISSENT	ISSEMENT	ELLE(S)	ELLEMENT	EUSE(S)
EUSEMENT	ENT(S)	ENTES	ENCE(S)	ANT(S)
ANTE(S)	ALE(S)	AUX	ALS	ALEMENT
E	EMENT	X	AIS	AIT
AITS	IEZ	AIENT	ABLE(S)	ABLEMENT
AI	AS	A	A/MES	A/TES
ABILITE(S)	IBILITE(S)	E	S	ER(S)
IEUR	EUR	EURS	RONT	

[49] These prefixes and suffixes are also checked for Canadian French.

DUTCH PREFIXES AND SUFFIXES

The following Dutch prefixes are processed by SCRIPT/VS during stem processing:

AAN	AARDAPPEL	ACHTEN	ACHTER
AF	ANTI	ATOOM	AUTO
AVERIJ	BANK BOEREN	BASIS BOOM	BEDRIJFS
BELEIDS	BE	BIER BUITEN	BIJEEN
BIJ	BINNEN	BLAAS	BODEM
BOUW	BOVEN	BRUTO	BUREAU
BURGER	BURO	CLUB	COMMANDO
CONTRA	DAAR	DAK	DEPOSITO
DISCONTO	DOOR	DRAAD	DRAAI
DRIE	DRIEE:N	DRUK	EENEN
EI	EIND	EXPORT	FABRIEKS
FILM	FOTO	GAS GIRO	GE GROEI
GELD	GROND	HALF	HAVEN
HER	HOEK	HOOFD	HUIS
HOEK	HUUR	IN INCASSO	JAAR
KABEL	KALK	KANTOOR	KAPITAAL
KLUB	KOMMANDO	KONTRA	KOOP
KOSTEN	LABORATORIUM	LEIDING	LOS
MAATSCHAPPIJ	MASSA	MEDE MILIEU	MEE
MICRO	MIKRO	NA	NEGENEN
NETTO	NIEUWBOUW	NIVEAU	NIVO
OCTROOI	OKTROOI	OM	ONDER
ONT	ON	OP	OVER
PERS	PLAN	POMP	POST
PRIJS	PRODUCTIE	PRODUKTIE	PROGRAMMA
PSYCHO	RADIO	RECLAME	REKLAME
RESEARCH	RIJ	RISICO	RISIKO
SALDO SPROEI	SAMEN	SCHAKEL	SCHEEPS
SLIB	STAATS	STOF	STROOM
STUDIE	TEGEN	TELEFOON	TELEGRAAF
TELEGRAM	TENTOON	TERREIN	TERUG
TOE	TOUW	TRANSITO	TROUW
TUSSEN	UIT	VAST	VEEL
VALUTA VLIEGTUIG	VENDU VOORT	VER	VIEREN
VERZEKERINGS VRACHT	VIJFEN	VLOEI	VLOEISTOF
VOOR	VRACHTEN	VRIJ	WAAR
WEER	WEG	WERP	WONING
ZAND	ZEE	ZELF	ZEVENEN
ZIJ	ZINK		

SCRIPT/VS also checks for the following Dutch suffixes:

ELIJKE	LIJKE	IJE	JE	ENDE
DENDE	ENDE	KTE	PTE	BARE
LOZE	E	ERE	ELE	AGE
IE:LE	DE	INGEN	IINGEN	ELIJKEN
LIJKEN	ELIJKHEDEN	LIJKHEDEN	HEDEN	SOREN
TOREN	EREN	ELEN	BAREN	LOZEN
EN	DEN	VAN	IEN	BAAR
ER	DER	LOOS	S	ELIJKHEID
LIJKHEID	HEID	END	DEND	IEND
ING	DING	IING		

ITALIAN PREFIXES AND SUFFIXES

There are two types of Italian prefixes that SCRIPT/VS checks for during stem processing. They are: contractions that are formed during elision processing; and grammatical prefixes. The following list summarizes the contractions that SCRIPT/VS checks for:

L'	ALL'	ANCH'	BELL'	COLL'	D'
DALL'	DEGL'	DELL'	GL'	NELL'	QUELL'
QUEST'	SULL'	UN'	NEANCH'	NESSUN'	NEINT'
QUAL'	QUALCOS'	QUALCUN'	QUAND'	QUANT'	SENZ'
C'	V'	DAGL'	S'	M'	T'
BUON'	COM'	DEV'	AGL'	SUGI'	COS
TUTT	GRAND				

The following list summarizes the grammatical prefixes that SCRIPT/VS checks for:

AUTO	ANTI	APPAR	BIS	CAPO	CENTRO
CON	CONTRO	DIS	DE	EX	EXTRA
FILO	FOTO	IN	IPER	INTER	RI
SEMI	SODDIS	SOM	SOPRA	SOS	SOTTO
SUPER	TELE	TRAS	TRAT	ULTRA	SUB
VICE	PRE	SOM	STRA	'SOVRA	'S
'INTRA	'CONTRA	'SOTT			

SCRIPT/VS processed the suffixes of Italian verbs differently than it does suffixes for Italian nouns. The Italian noun suffixes that SCRIPT/VS checks for are:

A	E	I	O	AZIONE	AZIONI
ANZA	ANZE	ATORE	ATORI	ATRICE	ATRICI
ATURA	ATURE	IZIONE	IZIONI	ISTA	ETTO
ETTO	ETTI	ETTA	ETTE	HETTO	HETTI
HETTA	LINO	LINI	INA	INE	INI
NCINO	TINA	TINO	AMENTE	CAMENTE	EMENTE
AMENTO	AMENTI	IMENTO	IMENTI	LMENTE	TAMENTE
CHE(	CHE)	ISSIMA	ISSIME	ISSIMI	ISSIMO
ATIVO	ATIVI	ATIVA	ATIVE	ABILE	ABILI

The Italian verb suffixes that SCRIPT/VS checks for are:

A	AI	AMMO	ANDO	ANO	ANTE
ANTI	ARE	ARONO	ASSE	ASSI	ASSIMO
ASTE	ASTI	ATA	ATE	ATI	ATO
AVA	AVAMO	AVANO	AVATE	AVI	AVO
O	INO	E	EMMO	ENDO	EI
ENTE	ENTI	ERONO	ESIMO	ESSE	ESSERO
ESSI	ESSIMO	ESTE	ESTI	ETE	EVA
EVAMO	EVANO	EVATE	EVI	EVO	UTA
UTE	UTI	UTO	ONO	ERE	ERO
ERAI	ERA	EREMO	ERETE	ERANNO	EREI
ERESTI	EREBBE	EREBBERO	EREMMO	ERESTE	AMERAI
METTERAI	MANGERAI	LASCERAI	ISTE	ISTI	ITA
ITA	II	I	IMMO	IRAI	IRANNO
IRE	IREBBE	IREBBERO	IREI	IREMMO	IREMO
IRESTE	IRESTI	IRETE	ITA	ITE	ITI
ITO	IVA	IVAMO	IVANO	IVATE	IVI
IVO	ISSE	ISSERO	ISSI	ANO	IAMO
IATE	ISCA	ISCANO	ISCE	ISCI	ISCO
ISCONO	CO	GO	CA	GA	GHE
CHE	CIA	RAI	RA	REMO	RETE
RANNO	RESTI	REBBE	REMMO	RESTE	REBBERO
SERO	ETTERO				

The fonts illustrated in this appendix are provided with SCRIPT/VS for use with the 3800 Printing Subsystem. One or two font names can be specified with the CHARS option of the SCRIPT command (for details, refer to "CHARS: Specify Fonts" on page 19). The SCRIPT/VS fonts cannot, in general, be combined in the CHARS option with the IBM 3800 fonts listed in Figure 40 on page 365.

Figure 42 lists the fonts provided by SCRIPT/VS for use with the 3800 Printing Subsystem. Each is a full uppercase and lowercase font.

Text Fonts	Highlight Fonts	Special Fonts
GT10 Gothic (10-pitch)	GB10 Gothic Bold	GR10 Gothic Reverse
GT12 Gothic (12-pitch)	GB12 Gothic Bold	GP12 Proportional
GT15 Gothic (15-pitch)	GI12 Gothic Italic	
ST10 Serif (10-pitch)	SI10 Serif Italic	RT10 Roman Text
ST12 Serif (12-pitch)	SI12 Serif Italic	SO12 Serif Overstruck
ST15 Serif (15-pitch)	SB12 Serif Bold	

Figure 42. Complete 3800 Fonts Provided With SCRIPT/VS: Each font is a complete set of special, upper- and lowercase characters. Any two of these fonts may be specified with the CHARS option of the SCRIPT command.

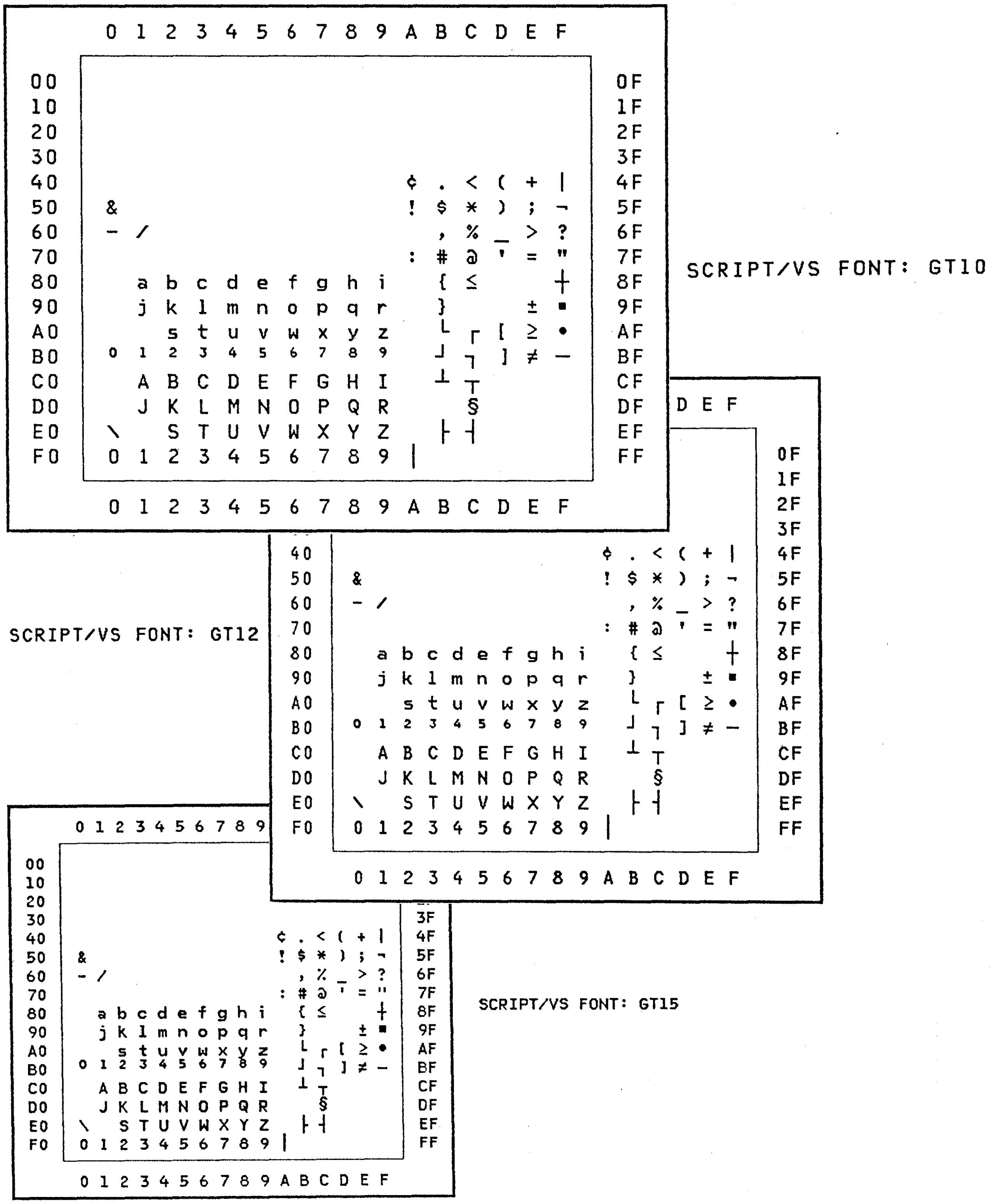


Figure 43. SCRIPT/VS Fonts: Gothic Text

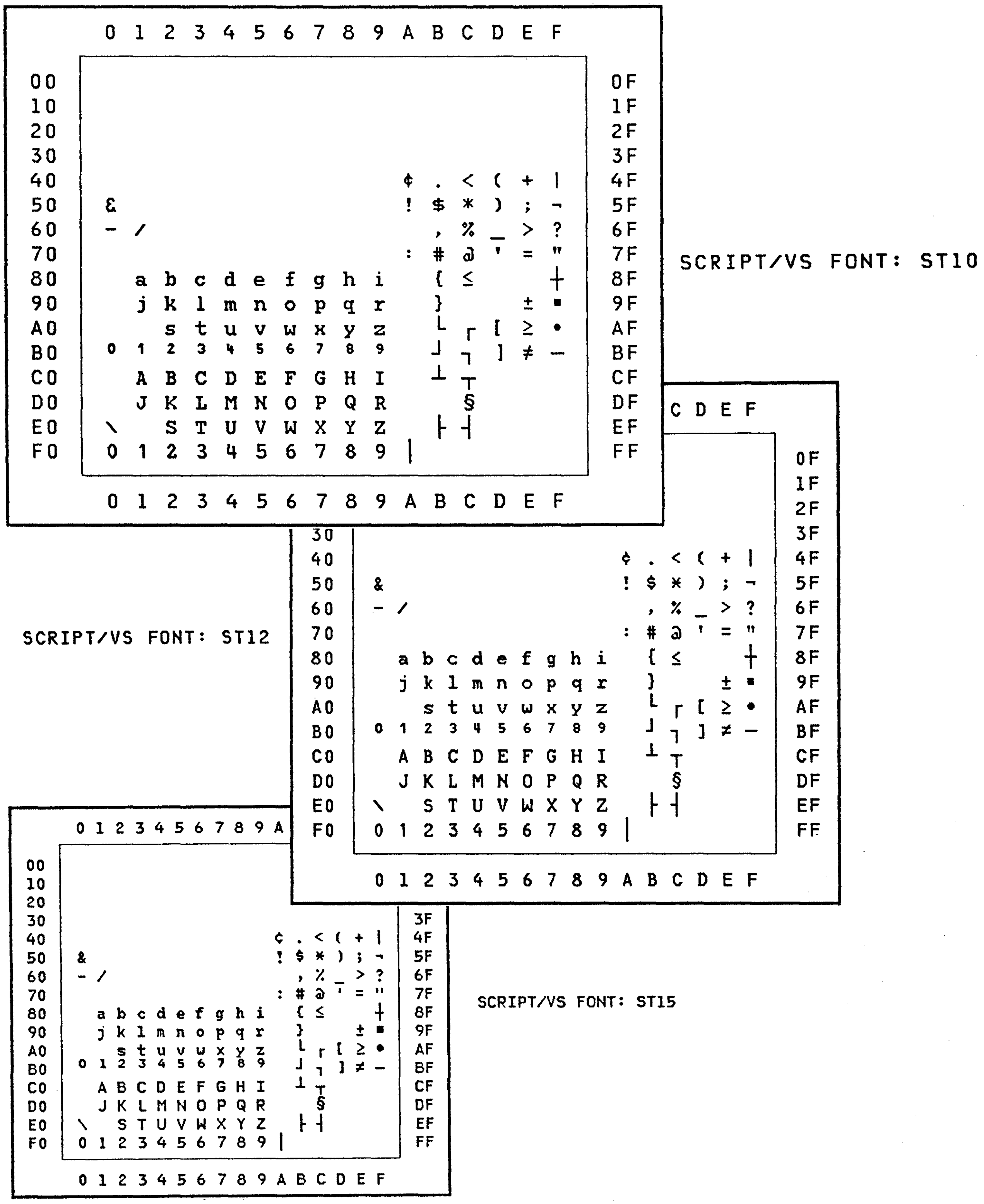

SCRIPT/VS FONT: ST10
SCRIPT/VS FONT: ST12
SCRIPT/VS FONT: ST15

Figure 44. SCRIPT/VS Fonts: Serif Text

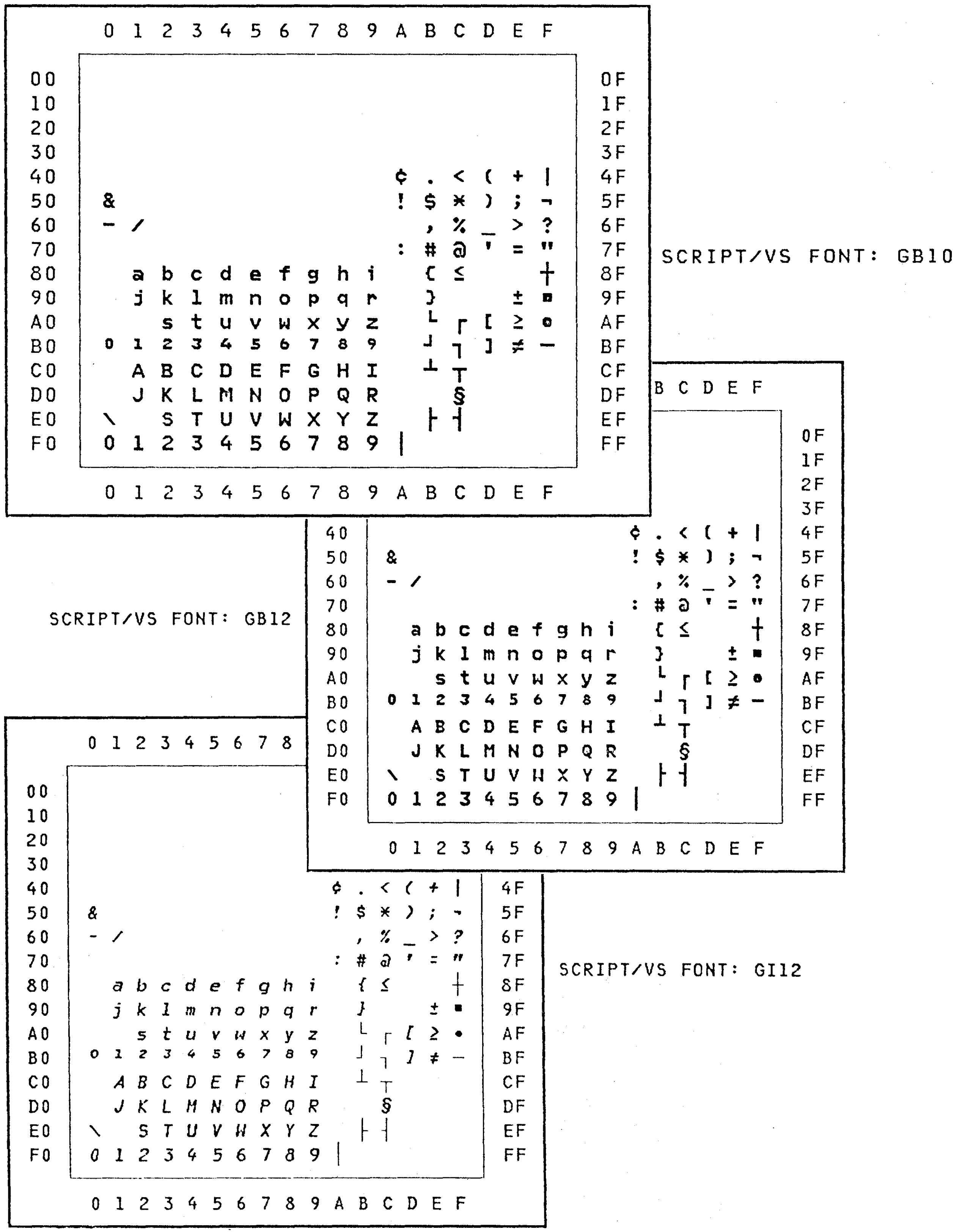

SCRIPT/VS FONT: GB10
SCRIPT/VS FONT: GB12
SCRIPT/VS FONT: GI12

Figure 45. SCRIPT/VS Fonts: Gothic Highlight

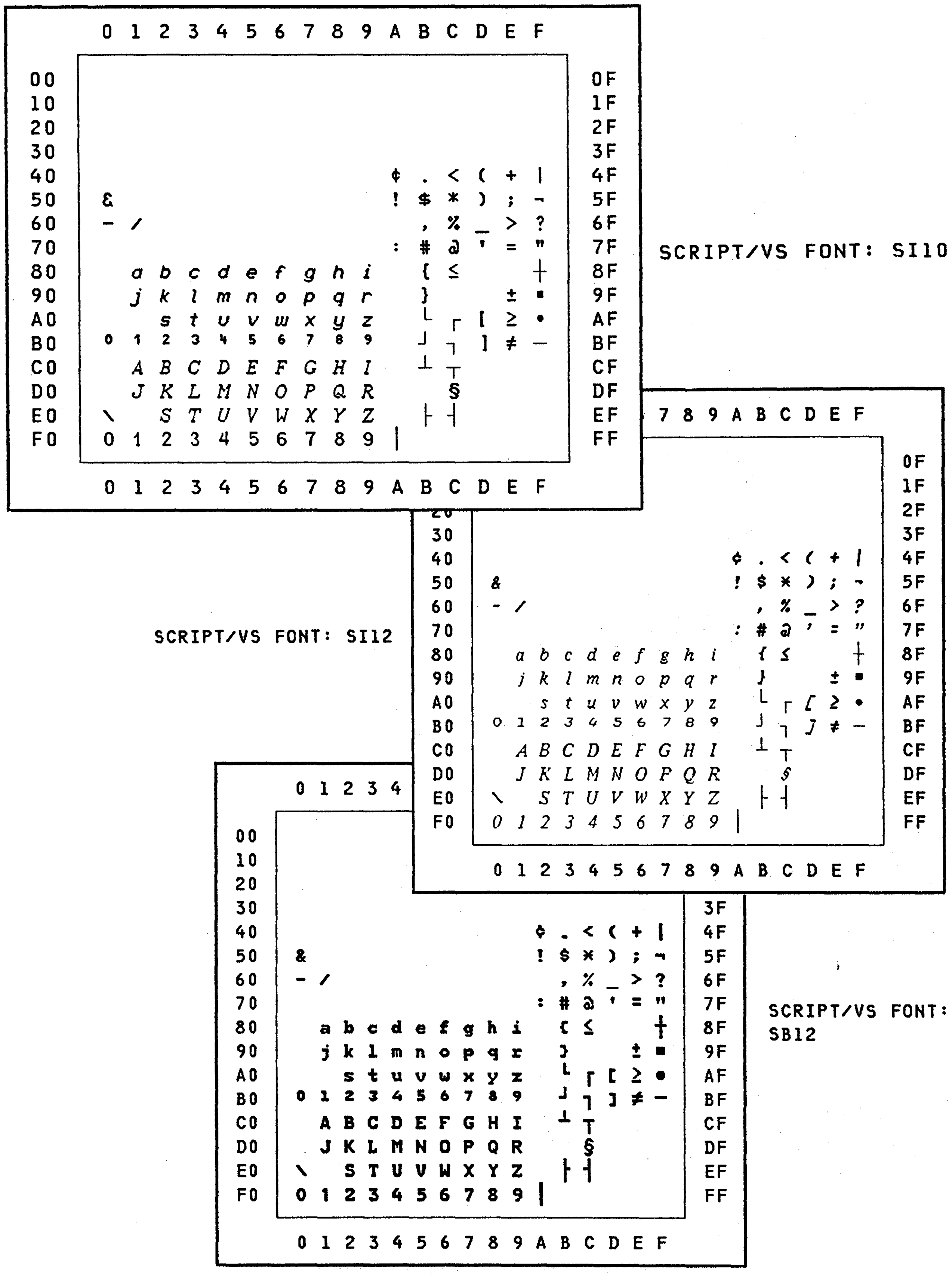
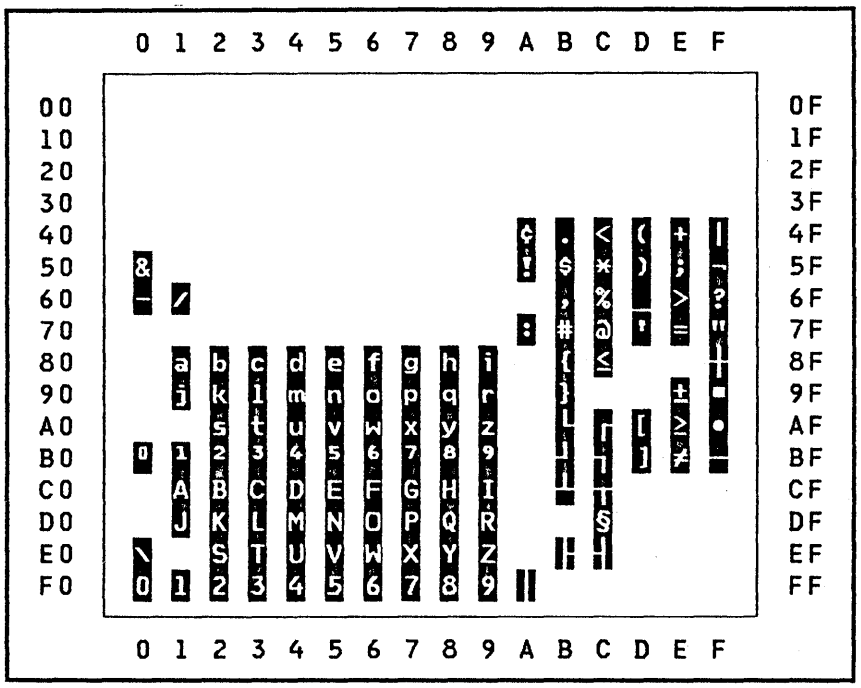
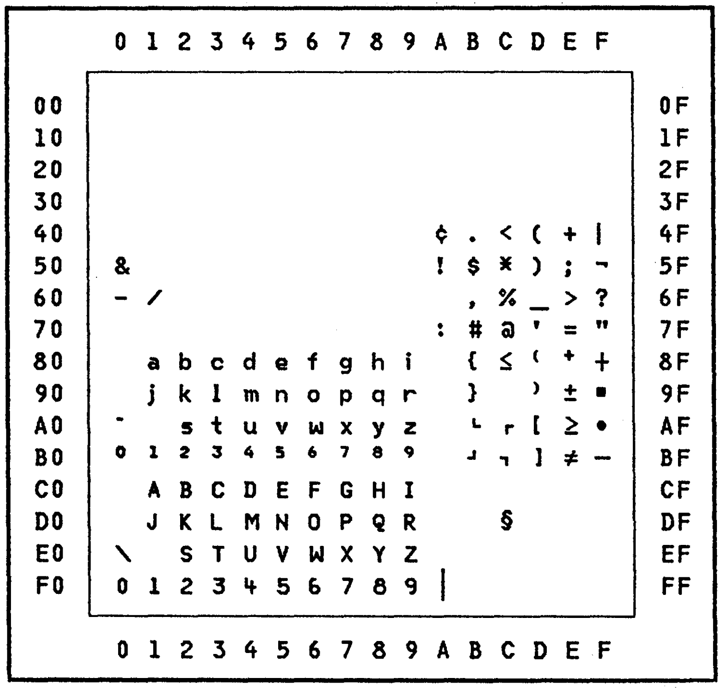

SCRIPT/VS FONT: SI10
SCRIPT/VS FONT: SI12
SCRIPT/VS FONT: SB12

Figure 46. SCRIPT/VS Fonts: Serif Highlight

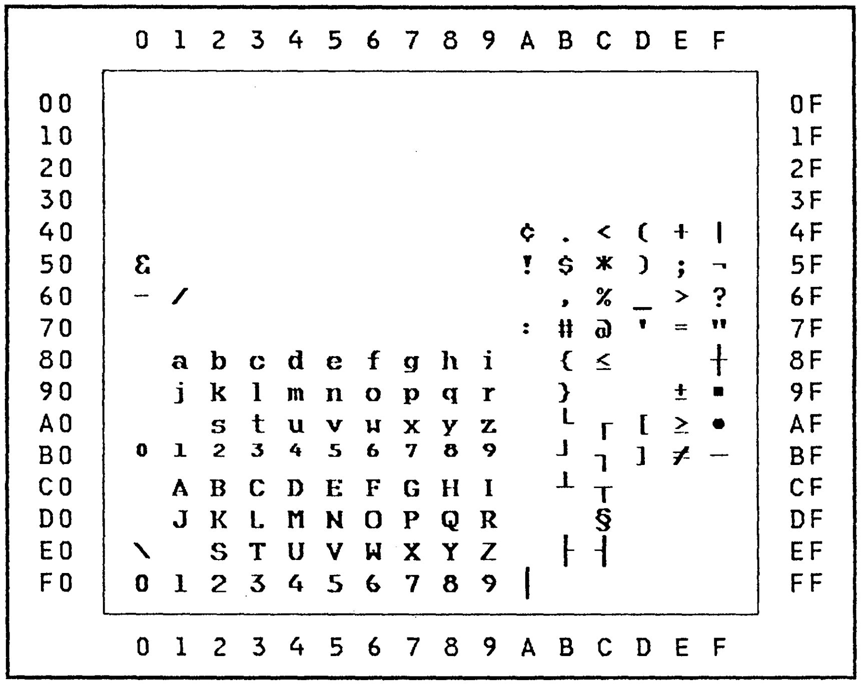


SCRIPT/VS FONT: GR10



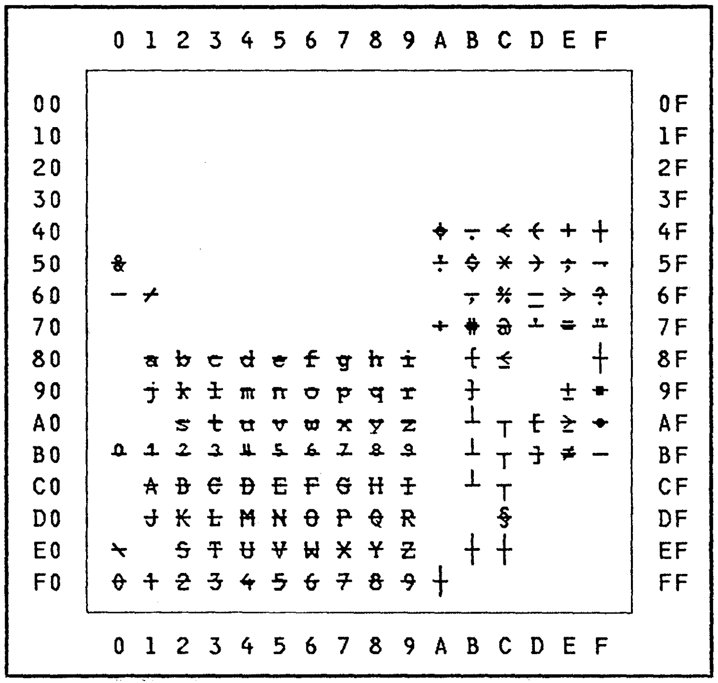
SCRIPT/VS FONT: GP12

Figure 47. SCRIPT/VS Fonts: Gothic Special Purpose



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00																	0F
10																	1F
20																	2F
30																	3F
40											¢	.	<	(	+	|	4F
50	&										!	$	*	)	;	¬	5F
60	-	/										,	%	_	>	?	6F
70											:	#	@	'	=	"	7F
80		a	b	c	d	e	f	g	h	i		{	≤			┼	8F
90		j	k	l	m	n	o	p	q	r		}			±	■	9F
A0			s	t	u	v	w	x	y	z		└	┌	[	≥	●	AF
B0	0	1	2	3	4	5	6	7	8	9		┘	┐	]	≠	—	BF
C0		A	B	C	D	E	F	G	H	I		┴	┬				CF
D0		J	K	L	M	N	O	P	Q	R			§				DF
E0	\		S	T	U	V	W	X	Y	Z		├	┤				EF
F0	0	1	2	3	4	5	6	7	8	9	|						FF
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

SCRIPT/VS FONT:
RT10



SCRIPT/VS FONT: SO12

Figure 48. SCRIPT/VS Fonts: Serif Special Purpose

APPENDIX E. FORMATTING CONSIDERATIONS FOR THE 3800 PRINTING SUBSYSTEM

This section contains some information and simple guidelines to help you when formatting documents for the 3800 Printing Subsystem.

Before reading this section, you should have a working knowledge of the control word syntax and functional capabilities of SCRIPT/VS. Additionally you should be familiar with the 3800 Printing Subsystem hardware and its system control program (SCP) support. Information about the 3800 Printing Subsystem can be found in IBM 3800 Printing Subsystem Programmer's Guide.

Font Management

The current font may potentially change each time a new input line is processed. A new font may be started at any point in the input stream. This may be between words or within a continued word. At the time a new font is started, the following information is available to the formatter:

- Address of the "font width table"
- Table Reference Character (TRC) for this font
- Availability of "special blanks"[50]

Line formatting proceeds based on these parameters until another font change is requested.

Fonts of different pitch that do not contain special blanks cannot be used on the same output line. Failure to observe this restriction may result in severe column misalignment.

A single output line includes all data in all columns that occupy a single print line position on the output device.

Usually, the desired results may be achieved using SCRIPT/VS-supplied fonts which contain the special blanks. However, if a local font is required, it is recommended that you supply "graphmods" for the special blank character codes.

Care should be taken in the definition of new fonts to observe the following conventions:

- The font should contain a multiple of 64 characters less one. This ensures that the Writable Character Generation Matrix (WCGM) storage in the 3800 Printing Subsystem is used efficiently (that is, has no unused WCGM positions).
- The first WCGM position (hexadecimal 00) should be assigned to the normal blank, usually hexadecimal 40, rather than SCRIPT/VS special blanks. Data positions in the output line that are not otherwise assigned (that is, unprintable characters) will be assigned to this character by the 3800 Printing Subsystem.
- The underscore character (hexadecimal 6D) should be assigned to the 45th WCGM position (hexadecimal 2D). The 3800 Printing Subsystem assumes that WCGM position 45 is the underscore character. If this position is unassigned, underscores will not appear in the output. Assigning this position to a character other than the underscore may cause unpredictable results.

---

[50] Hexadecimal 11 is 10 pitch. Hexadecimal 12 is 12 pitch. Hexadecimal 13 is 15 pitch.

- The last WCGM position (hexadecimal 3F) should not be assigned to any character code.

Tab, Backspace, and Underscore Resolution

It is necessary to resolve backspaces and tabs before line formatting can begin. The tab, backspace, or underscore characters,[51] when processed, cause changes in the input line data string.

If no fill character has been specified in the tab definition, tabs are expanded by inserting hexadecimal 40 characters or special blanks in appropriate combinations to fill the space from the character preceding the tab to the next defined tab stop position. Current character position is measured in pels from the beginning of the column, including indention.

A minimum space of 11 pel is required from the current line position to the tab stop; if the space is less than 11 pel, the next defined tab stop is used.

This minimum value guarantees that the tab expansion will end within 1 pel of the desired tab stop position if special blanks are available. See "Inline Space Management" on page 387 for more information.

All data to the left of a tab expansion is considered to be a single word segment. No wordspaces to the left of the tab will be considered for justification purposes. Normal line folding[52] at wordspaces is also inhibited to the left of a tab expansion.

Overstruck characters are ignored when formatting for the 3800 Printing Subsystem, except when the overstrike character is the underscore (_). This is true regardless of whether overstriking occurs because of backspacing or the .DF [Define Font] OS control word.

Interword Space

If justification is on and special blanks are present, all interword spaces in the input line are considered to be 15-pitch blanks.

When an input line does not fill the column width, a wordspace is added to the end. Successive input lines are formatted with intervening spaces until the column width is filled.

- If the input line ends with a full stop character, two wordspaces are added.
- When the input line ends in a continuation character, no wordspace is added.

Revision Code Characters

The revision code character is normally placed immediately preceding each changed line, separated from the column by a blank. Because the RC field has a variable width based on the width of the RC character and the RC adjust, it is necessary to measure and format it in the same way as text data.

It is most desirable for the first character of each text line to start in the same relative position. To ensure this, the RC character and its field must have a combined width that does not vary from line to line. If special blanks are present, this may be

[51] Hexadecimal 05, 16, and 6D are tab, backspace, and underscore, respectively.

[52] .FO ON or .CO ON.

achieved by combining the RC character with a special blank which brings the total width of RC and blank to 30 pel. The following table shows relative widths:

RC Width	Blank Width
12	18
15	15
18	12

The RC field is allocated from the binding margin in the first column, and from the intercolumn gutter space in subsequent columns. If insufficient space is available, the revision code will not appear in the output.

The RC field width should be defined such that sufficient space is allocated on both sides of the revision code character for proper inline space management. This requires that:

- The width of the RC field, less the width of the RC character, should be 0, 11-19, or more than 23 pel.
- The width of the intercolumn gutter, less the width of the RC field, should be 0, 11-19, or more than 23 pel.

If these restrictions are violated, inline space errors of up to 6 pel may result, as illustrated in Figure 49 on page 388.

Inline Space Management

Any time there is a need to fill some space in the output line, a space character string must be generated which will have a measured length equal to that of the desired space. This character string will be hexadecimal 40 characters, or a combination of special blanks. The accuracy of the length of a space string has a direct effect on the column alignment of the output line of which that string is a part.

The best accuracy that may be hoped for with the hexadecimal 40 string is ± one-half hexadecimal 40 character width. With care, one may set parameters in such a way as to minimize the probability that half-character alignment errors will occur.

One-half-character rounding errors may occur when space units are not specified in multiples of the character width. This occurs because the space unit value is resolved to native device units, pels in the horizontal direction, and may not always be satisfied accurately with a string of hexadecimal 40 characters. The situation is not affected by the magnitude of the request but by the relationship of the space value to the width of the hexadecimal 40 character at the point in time when the space is generated.

The best accuracy that may be hoped for with the special blank string is ± one pel. This level of accuracy is the best attainable on the 3800 Printing Subsystem and is quite satisfactory for most applications. This level of accuracy can normally be expected when these guidelines are followed:

- Define tabs so that the text before the tab always has room to end 11-19 pel, or more than 23 pel, before the defined tab stop position.
- Define columns and revision code fields in accordance with the discussion of revision codes earlier in this appendix.
- Specify indent values of 11-19 pel, or values greater than 23 pel.
- Limit split text filler strings to 1 or 2 characters.

If special blanks are present and the above guidelines are not followed, inline space errors of ± six pel may be encountered.

This is because it is not always possible to satisfy space requests accurately if the value requested is less than 23 pel.

With a 10-pitch font, errors of ± 9 pel may be encountered.

Alignment errors may potentially occur any time a space string is generated. The cumulative effect across a multicolumn line may be much greater than six pel.

Figure 49 illustrates the alignment errors that will be encountered when using special blanks and space values of less than 23 pel.

As can be seen from the table, the degree of error varies considerably in the 0 to 6 pel range for any request of less than 23 pel. Any request equal to or greater than 23 pel can be satisfied within 1 pel by a combination of the special blanks.

When special blanks are present, and a space request is processed which can not be accurately satisfied, a warning message is issued. The line in error will be flagged in the right margin with "<---- SPACE ERROR."

The formatter takes steps to ensure that inline space is specified outside the error windows shown in the table. The area in which errors are most likely to occur is in the gutter space. This is because the column definition may be disregarded when processing a line longer than column length with the EXTEND option of the .FO [Format Mode] control word. For this reason, the use of EXTEND is not recommended.

Requested	Actual	Error
1	0	-1
2	0	-2
3	0	-3
4	0	-4
5	0	-5
6	0	-6
7	12	+5
8	12	+4
9	12	+3
10	12	+2
11	12	+1
12	12	0
13	12	-1
14	15	+1
15	15	0
16	15	-1
17	18	+1
18	18	0
19	18	-1
20	18	-2
21	18	-3
22	18	-4
23	24	+1
24	24	0
25	24	-1

Figure 49. Justification Alignment Error for 3800 Printing Subsystem Output: Horizontal space cannot be reliably generated for distances less than 23 pel.

Box Processing

Boxes are supported in logical overlay mode relative to the output line; the box characters are overlaid on the output line after it is completely processed. Characters in the box line that occupy the same print position as a text character are printed; the text character does not appear in the output.

The use of special blanks is inhibited while box processing is in effect. That is, from the .BX definition input line to the .BX OFF input line. Slight variations in column alignment may occur in transition from the normal formatting environment, when special blanks are available for inline space management, to the box formatting environment, when hexadecimal 40 characters are used for inline space management. See "Inline Space Management" on page 387 for more information.

All characters in the box line and all characters in the text within the box must be of the same width:

- Truly proportional fonts may not be used for the box characters or the text within a box. The special blank characters may be present in the font, but all other characters must be of the same width.
- All fonts used within a box must be of the same pitch as the box font. The box font is the current font at the time the box is defined.

Formatter Escape Character

Formatted lines contain imbedded controls which are prefixed with the "escape" character hexadecimal 27. This use of hexadecimal 27 by SCRIPT/VS precludes its use as a data character code.

APPENDIX F. MAINTAINING USER DICTIONARIES

You can create user dictionaries to provide spelling verification and hyphenation support for words that are frequently used in your documents but are not included in the SCRIPT/VS main dictionary. These words may reflect the nature of your business and include such things as technical terms and company acronyms.

The user dictionaries are created and maintained using the dictionary maintenance process. This process reduces the words that are being placed in the user dictionary to a numerical representation that is expressed in Assembler language source statements. These source statements are assembled and linked to produce a module that is loaded by SCRIPT/VS when a .DL [Dictionary List] control word referencing the user dictionary is encountered.

Three types of reports can be generated when the dictionary maintenance process is run. They are:

- The alpha dictionary report, which is an alphabetical list of all words in the alpha dictionary file
- The hyphenation report, which is an alphabetic list of all words in the alpha dictionary file with their hyphenation points shown
- The oldest usage report, which lists words that have not been frequently updated by the dictionary maintenance process

The hyphenation report can be used to check for invalid hyphenation. It can also be used in conjunction with the oldest usage report to identify high-usage words for which hyphenation information should be added.

The oldest usage report can be used to determine which nonpermanent words should be deleted because of lack of use. It can also be used to determine which words are frequently used and should, therefore, be made permanent.

Two types of files are used in creating and maintaining a user dictionary:

- The alpha dictionary file, which contains the records that make up the user dictionary
- The alpha dictionary control file, which contains the user dictionary control information, such as the 4-character name used to access it and the date when it was last updated by the dictionary maintenance process

To create a user dictionary, the dictionary maintenance process must be run as a batch job in a background environment with the appropriate JCL. A sample of the JCL needed to create a user dictionary in an OS/VS1, OS/VS2 MVS, or DOS/VSE environment is provided with SCRIPT/VS.

The dictionary maintenance process requires an input file that consists of a control card and dictionary update transaction records:

- The control card indicates which of the three reports, if any, are to be produced, and the type of updates that are to be made to the alpha dictionary file. Figure 50 illustrates the format of this control card.
- The update transaction records contain the words that are to be added to or deleted from the user dictionary. Figure 51 on page 392 illustrates the format of these records. The control card and the update transaction records are all 80 characters in length.

Byte	Field	Description
1	ACTION	Is blank or "+" (The individual dictionary record for each word determines what action is to be performed.)
2	REPORT	Indicates which reports are to be generated: blank - no reports 1 - all three reports 2 - the alpha dictionary report 3 - the oldest usage report 4 - the hyphenation report 5 - the alpha dictionary and oldest usage reports 6 - the alpha dictionary and hyphenation reports 7 - the oldest usage and hyphenation reports
3-5	BLANK	This field is reserved.
6-7	OLDUSG	Contains a 2-digit number that is decremented by the number specified in the OUDEC field each time the dictionary maintenance process is run. When this number becomes less than or equal to the value specified in the OUDEC field, the temporary word it applies to is eligible for deletion if the alpha dictionary file becomes too large. This field only applies for nonpermanent words.
8-9	OUDEC	Contains a two-digit number (nn) that is used to determine when a nonpermanent word is eligible for deletion from the Alpha Dictionary file.
10	BLANK	This field is reserved.
11	DELALP	Indicates that any temporary word whose oldest usage counter (field OLDUSG) is below the number specified in the DELSET field is to be deleted. ("D" is specified in this case.) This field is left blank if you do not want to delete infrequently used words.
12-14	DELSET	Specifies a 3-digit number that is used in conjunction with the DELALP field to determine when a nonpermanent word should be deleted.
15-20	MAXDICT	Contains a six-digit number that indicates the maximum size of the user dictionary (usually 100000 bytes).
21-26	RUNDATE	Indicates (in the form mmddyy) the date that the dictionary maintenance process is being run. This date is placed in the DATE field of the record for each word being added if the field was left blank.
27-70	BLANK	This field is reserved.
71-72	CONTROL	Contains the characters "TR," which indicate that this is a control card.

Figure 50. Dictionary Maintenance Process Control Card Format

The update input file is checked for syntax errors and duplication. Any syntactically incorrect and duplicate transaction records are eliminated. The remaining records are then used to update the alpha dictionary file.

Bytes	Field	Description
1	ACTION	Contains the code that indicates what action is to be performed: "A" indicates that the word is to be added to the dictionary; "D" indicates that it is to be deleted; and "H" indicates that the word includes its hypenation points and is to be added to the dictionary if it does not already exist there.
2	PERM	Indicates whether the word is to be permanent or temporary. "P" is specified if the word is permanent and can only be deleted by specifying a "D" transaction code; it is left blank if the word is temporary and can be deleted if not used for a set period of time (it can also be deleted by specifying the "D" action code); and "@" is specified if the status of the word is to be changed from permanent to temporary.
3-65	ALPHA	Contains the word that is to be placed in the Alpha Dictionary file. This word must not start or end with a hyphen, and must consist of alphabetic characters or an apostrophe. (Hyphens can be included if "H" was specified in the ACTION field.) Imbedded blanks are not allowed.
66-71	DATE	Specifies the date that the word was added to the Alpha Dictionary file. The date must be specified in the form "mmddyy." It defaults to the date on the control card if the record is being added to the Alpha Dictionary file via the Input Output file.
72-80	BLANK	These columns should be left blank.

Figure 51. Format of an Update Transaction Record

The dictionary maintenance process consists of fourteen job steps:

- Step 1: The records in the update input file are checked for syntax errors. If any errors are found, warning messages will be issued; however, processing of the dictionary maintenance process will continue.
- Step 2: The file is sorted and the words are placed in alphabetical order.
- Step 3: The file is scanned for multiple entries for the same word. If any are found, the extra entries are deleted according to the transaction that they are requesting; an input card for a delete transaction takes precedence over a card for a hyphenation or addition transaction; and an input card for a hyphenation transaction takes precedence over a card for an addition transaction.
- Step 4: The alpha dictionary file is updated with the contents of the update input file, and the alpha dictionary report is produced if one was requested on the control card.
- Step 5: The hyphenation report is produced if one was requested on the control card. This job step is optional.
- Steps 6-12: Two assembler files are built, assembled, and linked into the SCRIPT/VS load library.
- Steps 13-14: The oldest usage report is produced if one has been requested on the control card. This job step is optional.

Once a user dictionary is created, you can issue a .DL control word to indicate to SCRIPT/VS that this dictionary is to be used, along with the SCRIPT/VS main dictionary, for hyphenation and spelling verification processing.

APPENDIX G. PERFORMANCE CONSIDERATIONS

Several facilities provided by SCRIPT/VS can significantly increase the system resources consumed in formatting documents. The facilities discussed in this appendix should be used with discretion, only when really needed, with an understanding of their impact on performance.

SCRIPT COMMAND OPTIONS

These options of the SCRIPT command can significantly increase the CPU resources needed to format a document:

- TWOPASS - Process the document twice
- SPELLCHK - Perform spelling verfication
- INDEX - Create an index

Each function must be explicitly specified; none are defaults. The effect on performance of each option is independent of the other options, and is discussed separately.

The TWOPASS Option

The TWOPASS option causes the document being formatted, including all imbedded files and macros, to be processed twice. Only the SCRIPT/VS symbol table and table of contents file are saved from the first pass; formatted output is produced only on the second pass.

The TWOPASS option must be used when an automatically generated table of contents is placed in the front of a document, or when reference is made to symbols which are set later in the document.[53]

Not unexpectedly, the TWOPASS option roughly doubles the system resources consumed in formatting a document. However, unresolved forward references are often acceptable in early proofing versions of a document. Similarly, the table of contents can often be moved temporarily to the back of the document. In these cases, the TWOPASS option may be omitted for all but the final formatting runs, when the table of contents is replaced properly.

The SPELLCHK Option

When spelling verification is enabled, each occurrence of every word in the document being formatted is reduced to its root form and checked against the active dictionaries.

Because spelling verification significantly increases the processor time required to format a document, it should be used only occasionally. Often it is sufficent to perform spelling verification only twice: once, when the document is first created, to find entry errors, acronyms, and valid words which are not in the dictionaries, and again, just before the final formatting runs, to catch any errors made while updating or revising the document.

The INDEX Option

When the INDEX option is included, index terms specified in the body of the document are saved and sorted to produce an index in the back of the document. Producing a large index can consume sig-

[53] These are called "forward references," because the value of the symbol is used before the symbol is defined.

nificant amounts of both virtual storage and processor time, since the index entries are kept in storage and sorted dynamically.

Since an INDEX is often not needed for draft copies of a document, the INDEX option may simply be omitted; the index terms specified in the document will be ignored.

SCRIPT/VS IN THE CICS ENVIRONMENT

In the CICS environment, ATMS-III provides facilities for creating and editing documents, and SCRIPT/VS may be used to format these documents. If you are an ATMS user, you may invoke SCRIPT/VS to format your documents in any one of three ways:

Online formatting: SCRIPT/VS may be invoked at the terminal to format a document currently residing in ATMS Working Storage. The output is placed in CICS/VS auxiliary temporary storage; it may then be transferred to a printer, or reviewed at your terminal.

Via a peripheral queue: Requests may be placed on a special queue for deferred document processing by SCRIPT/VS. A single CICS task is used to process all such queues.

Batch formatting: A batch job may be submitted to format a document using SCRIPT/VS if the document and all imbedded documents and macros reside within the Document Library Facility.

Tuning ATMS-III for SCRIPT/VS

Five parameters provided with the ATMS-III system generation macro DOKVA may be used to regulate SCRIPT/VS in the CICS environment. They are:

SPA limits the amount of virtual storage used by SCRIPT/VS when formatting documents submitted to a peripheral queue. SPA gives the number of 8K byte blocks which may be obtained per document, beyond an initial 64K byte block.

SPP limits the number of output pages which may be produced when formatting documents submitted to a peripheral queue. SPP gives the number of pages permitted.

STA limits the amount of virtual storage used by SCRIPT/VS when formatting documents online from a terminal. STA gives the number of 8K byte blocks which may be obtained per document, beyond an initial 64K byte block.

STO limits the number of concurrent online users of SCRIPT/VS.

TSP limits the number of output pages which may be produced when formatting documents online from a terminal. TSP gives the number of pages permitted.

Further information on tuning ATMS-III in the CICS environment may be found in the ATMS-III Program Reference Manual.

The glossary illustrates some basic SCRIPT/VS formatting concepts, and defines words and phrases that have special meanings in SCRIPT/VS or special meanings in a typographical sense. The terms are defined as they are used in this book. If you do not find the term you are looking for, refer to the index or to the _IBM Data Processing Glossary,_ GC20-1699.

This glossary includes definitions developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). This material is reproduced from the American National Dictionary for Information Processing, copyright 1977 by the Computer and Business Equipment Manufacturers Association, copies of which may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

ampersand: The "&" character.

When an ampersand begins a character string, SCRIPT/VS assumes the character string is a symbol name. If the symbol name is defined, SCRIPT/VS replaces the symbol with its value (unless symbol substitution is off).

In running footings, running headings, and running titles, the ampersand is usually the page number symbol.

When encountered by itself on the right side of a .SE [Set Symbol] control word, it is interpreted as the page number symbol.

APF: See application processing function.

application processing function (APF): In GML processing, the processing that is performed when a document element or attribute is recognized. In SCRIPT/VS, an APF is implemented as a sequence of control words, possibly intermixed with text and symbols, in one of three forms: macro definition, value of a symbol, or imbedded file.

attribute: A characteristic of a document (or document element) other than its type or content. For example, the security level of a document or the depth of a figure.

attribute label: In GML markup, a name of an attribute that is entered in the source document when specifying the attribute's value.

back matter: In a book, those sections (such as glossary and index) that are placed after the main chapters or sections.

balancing: In multicolumn formatting, the process of making column depths on a page approximately equal.

batch environment: The environment in which noninteractive programs are executed.

binding edge: The edge of a page to be bound, stapled, or drilled. Defined with the BIND option of the SCRIPT command. (See also Figure 21 on page 342.)

body: (1) Of a printed page, that portion between the top and bottom margins that contains the text. (2) Of a book, that portion between the front matter and the back matter.

boldface: A heavy-faced type. Also, printing in this type.

bottom margin: On a page, the space between the body or the running footing, if any, and the bottom edge of the page. The bottom margin area includes the bottom title lines, if any. (See also Figure 21 on page 342.)

bottom title: Up to six lines of data repeated at the bottom of consecutive pages (or of consecutive odd- or even-numbered pages) in the footing space. (See also Figure 21 on page 342.)

break: An interruption in the formatting of input lines, so that the next input line is printed on a new output line.

caps: Capital letters. (See also initial caps.)

caption: Text accompanying and describing an illustration.

character: A symbol used in printing. For example, a letter of the alphabet, a numeral, a punctuation mark, or any other symbol that represents information.

character set*: A finite set of different characters that is agreed to be complete for some purpose. For example, in printing, the characters that constitute a font.

character spacing: The space between characters in a word.

* American National Standard Definition

cicero: In the Didot point system, a unit of 0.1776 inch (4.512 millimeters) used in measuring typographical material.

CMS: An interactive processor that operates within VM/370.

column width: The width of each text column on a page. Specified with the .CL [Column Line Length] control word. (In multicolumn formatting, all columns on a page usually have the same width.) (See also Figure 21 on page 342.)

command: A request from a terminal or specified in a batch processing job for the performance of an operation or the execution of a particular program. For example, a request given at a terminal for SCRIPT/VS to format a document, or for an editor to edit a line of text.

comment: A control word line which is ignored by SCRIPT/VS. Such lines begin with either ".*" or ".cm". (See "Chapter 3. Basic Text Processing" on page 33.)

composition: The act or result of formatting a document.

concatenation: The forming of an output line that contains as many words as the column width allows, by placing the first words from an input line after the last words from the preceding input line. When words from an input line would reach beyond the right margin and hyphenation cannot be performed, they are placed at the beginning of the next output line, and so on.

control word: An instruction within a document that identifies its parts or tells SCRIPT/VS how to format the document. (See also macro.)

control word line: An input line that contains at least one control word.

current left margin: The left limit of a column that is in effect for formatting. Each column's left margin is specified with the .CD [Column Definition] control word. However, the current left margin (that is, the left boundary for an output line) might vary to the right of the column's left margin when indention is changed with the .IN [Indent], .UN [Undent], .IL [Indent Line], and .OF [Offset] control words. (See also Figure 6 on page 40.)

current line: The line in a source document at which a computer program (such as an editor or a formatter) is positioned for processing.

debug: To detect, trace, and eliminate errors in computer programs and SCRIPT/VS documents.

default value: A value assumed by a computer program when a control word, command, or control statement with no parameters is processed.

dictionary: A collection of "word stems" that is used with the spelling verification and automatic hyphenation functions.

Didot point system: A standard printer's measurement system on which type sizes are based. A Didot point is 0.0148 inch (0.376 millimeter). There are 12 Didot points to a cicero. (See also cicero and point.)

document: (1) A publication or other written material. (2) A machine-readable collection of lines of text or images, usually called a source document. (See also output document and source document.)

document conversion processor: A computer program that processes a machine-readable document which includes formatting controls written in one formatter language, to produce a machine-readable document which includes formatting controls appropriate for another formatter language.

document library: A set of VSAM data sets, accessible in a batch environment, that contain documents and related files.

duplex: A mode of formatting appropriate for printing on both sides of a sheet.

EBCDIC*: Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

edit: To create or modify the contents of a document or file. For example, to insert, delete, change, rearrange, or copy lines.

editor: A computer program that processes commands to enter lines into a document or to modify it.

eject: In formatting, a skip to the next column or page.

em: A unit of measure for a particular font that is equal to the point size of that font.

extended symbol processing: The processing of a symbol whose value causes the remainder of the line to be stacked

* American National Standard Definition

and later processed as a new input line.

fill character: The character that is used to fill up a space; for example, blanks used to fill up the space left by tabbing.

float: (1) (noun) A keep (group of input lines kept together) whose location in the source file may vary from its location in the printed document. (2) (verb) Of a keep, to be formatted in a location different from its location in the source file.

flush: Having no indention.

fold: (1) To translate the lowercase characters of a character string into uppercase. (2) To place that portion of a line which does not fit within a column on the next output line.

font: An assortment of type, all of one size and style.

font set: The set of fonts to be used in formatting a source document.

footing: Words located at the bottom of the text area. (See also running footing, bottom title, and Figure 21 on page 342.)

footing margin: That part of the bottom margin area between the body of the page or running footing, if any, and the bottom title(s), which is located in the footing space. (See also Figure 21 on page 342.)

footing space: That part of the bottom margin that is available for bottom title(s). (See also Figure 21 on page 342.)

footnote: A note of reference, explanation, or comment, placed below the text of a column or page, but within the body of the page (above the running footing).

foreground: The environment in which interactive programs are executed. Interactive processors reside in the foreground.

format: (1) (noun) The shape, size, and general makeup of a printed document. (2) (verb) To prepare a document for printing in a specified format.

formatting mode: In document formatting, the state in which input lines are concatenated and the resulting output lines are justified.

formatter: (1) A computer program that prepares a source document to be printed. (2) That part of SCRIPT/VS that formats input lines for a particular logical device type.

front matter: In a book, those sections (such as preface, abstract, table of contents, list of illustrations) that are placed before the main chapters or sections.

Generalized Markup Language (GML): A language that may be used to identify the parts of a source document without respect to particular processing.

GML: Generalized Markup Language

gutter: In multicolumn formatting, the space between columns. (See also Figure 21 on page 342.)

hanging indention: The indention of all lines of a block of text, following the first line (which is not indented the same number of spaces). Specified with the .OF [Offset] or .UN [Undent] control word. (See also Figure 6 on page 40.)

head-level: The typeface and character size associated with the words standing at the beginning of a chapter or chapter topic.

heading: Words located at the beginning of a chapter or section or at the top of a page. (See also head-level, running heading, and top title.

heading margin: That part of the top margin area between the body of the page or running heading, if any, and the top title, which is located in the heading space. Specified with the .HM [Heading Margin] control word. (See also Figure 21 on page 342.)

heading space: That part of the top margin area that is available for top title(s). Specified with the .HS [Heading Space] control word. (See also Figure 21 on page 342.)

hexadecimal: Pertaining to a number system based on 16, using the sixteen digits 0, 1, . . . 9, A, B, C, D, E, and F. For example, hexadecimal 1B equals decimal 27. (See also EBCDIC.)

indent: To set typographical material to the right of the left margin.

indention: The action of indenting. The condition of being indented. The blank space produced by indenting. Specified with the .IN [Indent], .IR [Indent Right], .UN [Undent], .OF [Offset], and .IL [Indent Line] control words. (See also hanging indention and Figure 6 on page 40.)

initial caps: Capital letters occurring as the first letter of each word in a phrase. To set a phrase in initial caps is to capitalize the first letter of each word in the phrase.

initial value: A value assumed by SCRIPT/VS for a formatting function until the value is explicitly changed with a control word. The initial value is assumed even before the control word is encountered, whereas the default value is assumed when the control word is issued without parameters. (See also default value.)

input device: A machine used to enter information into a computer system (for example, a terminal used to create a document).

input line: A line, as entered into a source file, to be processed by a formatter.

interactive: Pertaining to an application in which entries call forth a response from a system or program, as in an inquiry system. An interactive system might also be conversational, implying a continuous dialog between the user and the system. Interactive systems are usually communicated with via terminals, and respond immediately to commands. (See also foreground.)

interactive environment: The environment in which an interactive processor operates.

italic: A typestyle with characters that slant upward to the right.

JCL: Job control language.

job control language (JCL)*: A language of control statements used to identify a computer job or describe its requirements to the operating system.

job control statement: A statement that provides an operating system with information about the job being run.

justify: To insert extra blank space between the words in an output line to cause the last word in the line to reach the right margin. As a result, the right-hand edge of each output line is aligned with preceding and following output lines.

keep: (noun) In a source document, a collection of lines of text to be printed in the same column. When the vertical space remaining in the current column is insufficient for the block of text, the text is printed in the next column. (In the case of single-column format, the next column is on the next page.)

layout: The arrangement of matter to be printed. (See also format.)

leader: (1) Dots or hyphens (as in a table of contents) used to lead the eye horizontally. (2) The divider between text and footnotes on a page (usually a short line of dashes, although you can redefine it).

left-hand page: The page on the left when a book is opened; usually even-numbered.

line spacing: The space between the baseline of one output line and the baseline of the adjacent output line.

lowercase: Pertaining to small letters as distinguished from capitals; for example, "a, b, g" rather than "A, B, G."

machine-readable: Data in a form such that a machine can acquire or interpret (read) it from a storage device, from a data medium, or from another source.

macro*: An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language. In SCRIPT/VS, a macro is a sequence of one or more control words, symbols, and input lines. A macro's definition can be recursive.

macro substitution: During formatting, the substitution of control words, symbols, and text for a macro.

margin: (1) The space above, below, and on either side of the body of a page. (2) The left or right limit of a column. (See also Figure 21 on page 342.)

mark up: (verb) (1) To determine the markup for a document. (2) To insert markup into a source document.

markup: (noun) Information added to a document that enables a person or system to process it. Markup may describe the document's characteristics, or it may specify the actual processing to be performed. In SCRIPT/VS, markup consists of GML tags, attribute labels and values, and control words.

offset: (verb) To indent all lines of a block of text, except the first line. (noun) The indention of all lines of a block of text following the first line. (See also Figure 6 on page 40.)

option: Information entered with a SCRIPT command to control the execution of SCRIPT/VS.

output device: A machine used to print, display, or store the result of processing.

output document: A machine-readable collection of lines of text or images

* American National Standard Definition

that have been formatted, or otherwise processed, by a document processor. The output document can be printed or it can be filed for future processing.

output line: A line of text produced by a formatter.

paginate: To number pages.

parameter: Any one of a set of properties whose values determine the characteristics or behavior of something. The syntax of some SCRIPT/VS control words includes parameters, which establish the properties of a formatting function or a printed page.

PDS: partitioned data set.

pel: The unit of horizontal measurement for the IBM 3800 Printing Subsystem. One pel equals approximately 1/180th inch.

pica: A unit of about 1/6 inch used in measuring typographical material. Similar to a cicero in the Didot point system.

pitch: A number that represents the amount of horizontal space a font's character occupies on a line. For example, 10-pitch means 10 characters per inch, or each character is 0.1 (1/10) inch wide. 12-pitch means 12 characters per inch, and 15-pitch means 15 characters per inch.

point: (1) A unit of about 1/72 of an inch used in measuring typographical material. There are twelve points to the pica. (2) In the Didot point system, a unit of 0.0148 inch There are twelve Didot points to the cicero.

profile: (1) In SCRIPT/VS processing, a file that is imbedded before the primary file is processed. It can be used to control the formatting of a class of source documents. When processing GML markup, the profile usually contains the mapping from GML to APFs, and the symbol settings that define the formatting style. (2) In the Document Library Facility library, a collection of information that identifies a batch SCRIPT/VS user (user profile) or a document processor (attribute profile) or that defines certain library parameters (system profile).

proportional spacing: The spacing of characters in a printed line so that each character is allotted a space proportional to the character's width.

ragged right: The unjustified right edge of text lines. (See also justify.)

residual text: The line of text following the markup/content separator of a GML tag.

right-hand page: The page on the right when a book is opened; usually odd-numbered.

rule: (noun) A straight horizontal or vertical line used, for example, to separate or border the parts of a figure or box.

running footing: A footing that is repeated above the bottom margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the page's body (text area). (See also Figure 21 on page 342.)

running heading: A heading that is repeated below the top margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the page's body (text area). (See also Figure 21 on page 342.)

running title: In SCRIPT/VS, up to six lines of data that may be repeated in the top or bottom margin area of consecutive pages (or of odd- or even-numbered pages.)

section: When an output page has two or more single-column parts with the same or different column-widths, or a single-column part and a multicolumn part, or two or more different multicolumn parts, each part of the output page is called a section.

small caps: Capital letters in the same style as the normal capital letters in a font, but approximately the size of the lowercase letters.

source document: A machine-readable collection of lines of text or images that is used for input to a computer program.

space: A blank area separating words or lines.

symbol: A name in a source document that can be replaced with something else. In SCRIPT/VS, a symbol is replaced with a character string. SCRIPT/VS may interpret the character string as a number, a character string, a control word, or another symbol.

symbol substitution: During formatting, the replacement of a symbol with a character string which SCRIPT/VS may interpret as a value (numeric, character string, or control word) or as another symbol.

tab: (1) (noun) A preset point in the typing line of a typewriter-like terminal. A preset point in an output line. (2) (verb) To advance to a tab for printing or typing. (3) (noun) a tab character, hexadecimal 05.

tag: In GML markup, a name for a type of document (or document element) which is entered in the source document to identify it. For example, ":p." might be the tag used to identify each paragraph.

terminal: A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

text line: An input line that contains only text.

title: See running title.

token: A string of characters which is treated as a single entity. In SCRIPT/VS, a parameter passed to a macro in one of the local variables &*1, ... &*n. (See "Chapter 13. Writing SCRIPT/VS Macro Instructions" on page 147.)

top margin: On a page, the space between the body or running heading and the top edge of the page. The top margin includes the top titles, if any. (See also Figure 21 on page 342.)

top title: Up to six lines of data repeated at the top of consecutive pages (or of consecutive odd- or even-numbered pages) in the heading space. (See also running title and Figure 21 on page 342.)

TRC: table reference character. In printer SYSOUT data sets, a second control byte, following the carriage control byte, which indicates which font the record is to be printed in. The presence of TRCs is indicated by the JCL parameter DCB=OPTCD=J.

TSO: An interactive processor within OS/VS2.

typeface: All type of a single style. There might be several fonts (different sizes) with the same typeface or style.

typeset: (1) (verb) To arrange the type on a page for printing. (2) (adjective) Pertaining to material that has been set in type.

underscore: (1) (noun) A line printed under a character. (2) (verb) To place a line under a character. To underline.

unformatted mode: (1) In document formatting, the state in which each input line is processed and printed without formatting. Other SCRIPT/VS control words remain in effect and are recognized. (2) In document printing using the UNFORMAT option, the state in which each input line (control words as well as text) is printed as it exists in the input, in the order in which it is processed. No formatting is done.

uppercase: Pertaining to capital letters, as distinguished from small letters; for example, "A, B, G" rather than "a, b, g."

widow: One or two lines or words at the end of a paragraph that are printed separately from the rest of the paragraph.

word spacing: The space between words in a line. Also called wordspace.

102721970

Document Composition Facility:
User's Guide
Order No. SH20-9161-3

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or the IBM branch office serving your locality.*

	Yes	**No**
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐
• How do you use this publication:		
As an introduction to the subject?	☐	
For advanced knowledge of the subject?	☐	
To learn about operating procedures?	☐	
As an instructor in class?	☐	
As a student in class?	☐	
As a reference manual?	☐	

• What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

SH20-9161-3

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

Attention: Information Development
Department 61C

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
General Products Division
P. O. Box 27155
Tucson, Arizona 85726

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

10

Document Composition Facility: User's Guide Printed in U.S.A. SH20-9161-3

Document Composition Facility:
User's Guide
Order No. SH20-9161-3

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or the IBM branch office serving your locality.*

	Yes	No
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐
• How do you use this publication:		
As an introduction to the subject?	☐	
For advanced knowledge of the subject?	☐	
To learn about operating procedures?	☐	
As an instructor in class?	☐	
As a student in class?	☐	
As a reference manual?	☐	

• What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

SH20-9161-3

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

Attention: Information Development
Department 61C

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
General Products Division
P. O. Box 27155
Tucson, Arizona 85726

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

Document Composition Facility: User's Guide Printed in U.S.A. SH20-9161-3

Document Composition Facility:
User's Guide
Order No. SH20-9161-3

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or the IBM branch office serving your locality.*

	Yes	**No**
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐
• How do you use this publication:		
As an introduction to the subject?	☐	
For advanced knowledge of the subject?	☐	
To learn about operating procedures?	☐	
As an instructor in class?	☐	
As a student in class?	☐	
As a reference manual?	☐	

• What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

SH20-9161-3

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

Attention: Information Development
Department 61C

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
General Products Division
P. O. Box 27155
Tucson, Arizona 85726

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

Document Composition Facility: User's Guide Printed in U.S.A. SH20-9161-3

SH20-9161-3

Document Composition Facility: User's Guide Printed in U.S.A. SH20-9161-3