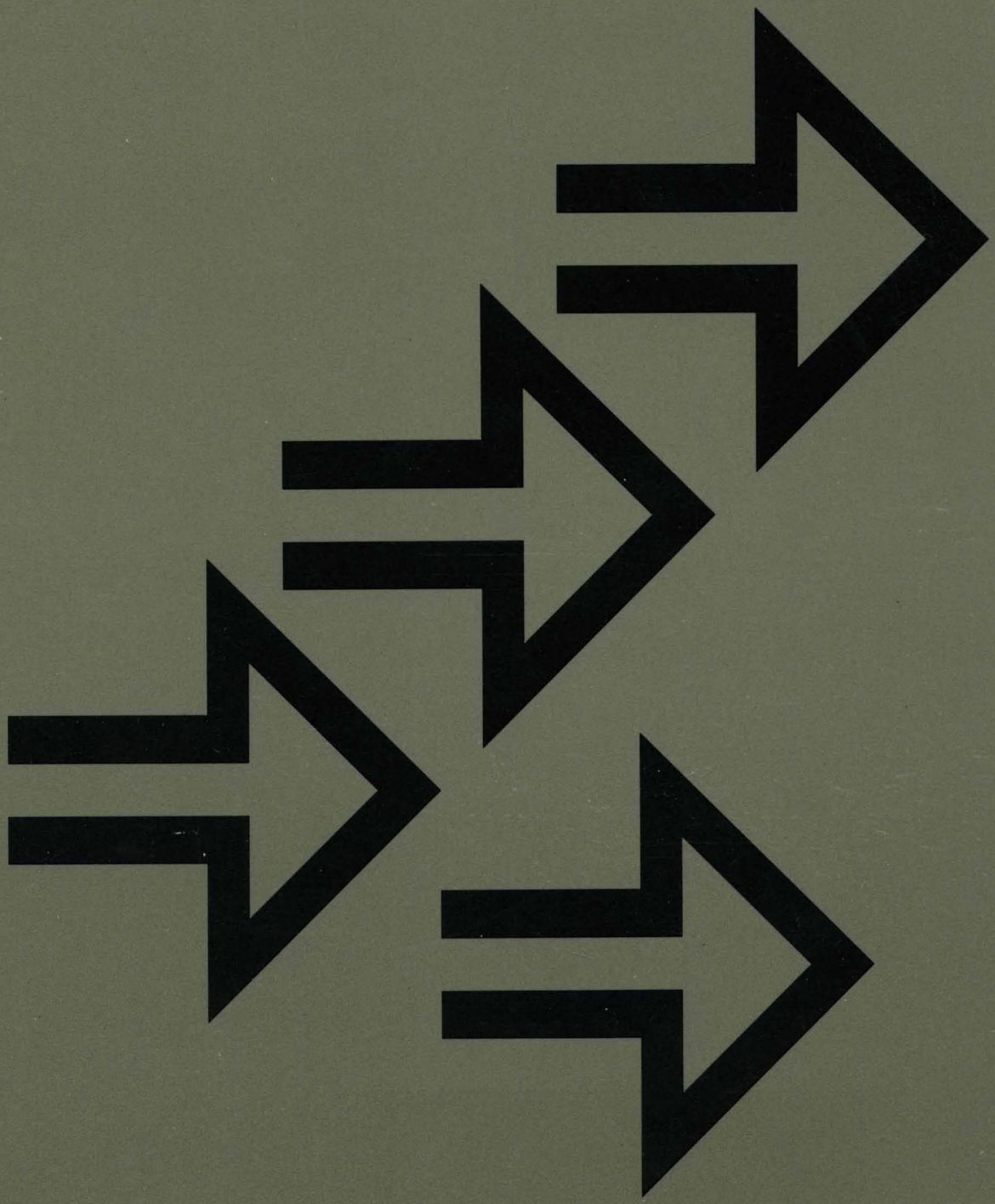
IBM

Document Composition Facility SCRIPT/VS Text Programmer's Guide

Program Product

SH35-0069-2



Program Number 5748-XX9

Release 3

DOCUMENT COMPOSITION FACILITY: SCRIPT/VS TEXT PROGRAMMER'S GUIDE RELEASE 3

Document Number SH35-0069-2

January 22nd, 1985

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

Third Edition (March 1985)

This edition contains information from and makes obsolete the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide*, SH35-0069-1.

Technical changes in this edition are marked by vertical bars in the left margin.

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

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 and 4300 Processors Bibliography* , GC20-0001, for editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not available in your country. Such references or information must not be construed to mean that IBM intends to announce such products in your country.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Products Division, Box 1900, Department 580, Boulder, Colorado, U.S.A. 80301. 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.

Preface

This manual describes the IBM Document Composition Facility (DCF) program product and its component text processing program, SCRIPT/VS, and provides the information necessary to use them. This book should be used in conjunction with the *Document Composition Facility: SCRIPT/VS Language Reference*, which defines the SCRIPT command and the SCRIPT/VS control words.

In order to use this book, the end user must be familiar with:

- The concepts of text processing and formatting
- The operating environment on which SCRIPT/VS resides
- A text editor supported in the above environment.

We recommend that new users satisfy their text formatting requirements by using the Generalized Markup Language (GML).

This publication is specifically designed for users whose tasks may include:

- Formatting documents using SCRIPT/VS control words.
- Modifying the Document Composition Facility Generalized Markup Language starter set. See the *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide* for more details on modifying the GML starter set.[1]
- Creating Generalized Markup Language applications.
- Installing, modifying, and maintaining the Document Composition Facility.

This book is not addressed to users of any installation defined GML applications. These users should refer to their application's documentation. Users of the GML starter set application should refer to the *Document Composition Facility: Generalized Markup Language Starter Set User's Guide*, the *Document Composition Facility: Generalized Markup Language Starter Set Reference*, and the *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide.*

The information in this publication applies equally to OS/VS2 MVS, VSE, VM/SP, 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).

References to the 3800 Printing Subsystem refer to both the 3800 Printing Subsystem Model 1 and to the 3800 Printing Subsystem Model 3 (in compatibility mode) unless otherwise explicitly stated.

---

[1] **Field Engineering support and maintenance is provided *only* on the unmodified GML starter set. If you modify the starter set, it is recommended that you also maintain an *unmodified* starter set for diagnostic purposes.**

References to the 3800 Printing Subsystem Model 1 also apply to the 3800 Printing Subsystem Model 3 (in compatibility mode) unless otherwise explicitly stated.

References to page printers apply to the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer unless otherwise explicitly stated.

Organization

The chapters of this book are:

- "Chapter 1. An Overview of SCRIPT/VS": 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": A description of how to use and specify the SCRIPT command. This chapter discusses file requirements and conventions and how to use the SCRIPT command in several environments.
- "Chapter 3. Marking Up Documents with SCRIPT/VS": A description of SCRIPT/VS language syntax and control words. This chapter includes a discussion of the control word separator and space notation.
- "Chapter 4. Combining SCRIPT/VS Input Files": A description of how to imbed and append SCRIPT/VS files. This chapter includes a brief summary of SCRIPT/VS utility files.
- "Chapter 5. Communicating with SCRIPT/VS": A description of interaction with SCRIPT/VS. This chapter includes a discussion of SCRIPT/VS messages and interactive SCRIPT/VS processing.
- "Chapter 6. Composing Lines": A description of how to define the parameters of a line. This chapter includes a discussion of concatenation, justification, indention, tabs, and marking updated material.
- "Chapter 7. Hyphenating and Horizontally Justifying Text": A description of how to hyphenate words, how to horizontally justify text, and how to use the algorithmic hyphenator.
- "Chapter 8. Creating Vertical Space": A description of how to insert vertical space into your text.
- "Chapter 9. Vertically Justifying Text": A description of how to vertically distribute, format and justify text.
- "Chapter 10. Establishing Page Layout": A description of how to define the parameters of a page, such as page length, page width, column line length, line length, and page numbering. This chapter describes running headings and footings.
- "Chapter 11. Placing Text in Named Areas": A description of how to define and place *named* areas in your text. Also included is a discussion of how segments can be included in your text.
- "Chapter 12. Composing Multiple-Column Pages": A description of how to establish a multicolumn format for the body of a page.
- "Chapter 13. Creating Head Levels and Table of Contents": 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 14. Creating Rules and Boxes": A description of how to create simple and complicated boxes and how to draw horizontal and vertical rules.

- "Chapter 15. Selecting Fonts": A description of how to define and specify fonts for line devices, the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer and how to use fonts for emphasis.
- "Chapter 16. Keeping Blocks of Text Together": A description of widow zones, in-line keeps, and floats.
- "Chapter 17. Creating Footnotes": A discussion of how to create footnotes and footnote leaders.
- "Chapter 18. Translating Characters": A description of character manipulation and input and output character translation. This chapter includes a discussion of uppercase and string translation as well.
- "Chapter 19. Creating an Index": A description of how to create an index by placing index entry information in the text of a document.
- "Chapter 20. Defining the Formatting Environment": A description of the SCRIPT/VS formatting environment.
- "Chapter 21. Processing Symbols": A description of the SCRIPT/VS symbol processing capability and 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 22. Processing Logical Statements": 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 23. Processing Macros": A description of the SCRIPT/VS macro processing capability and how to define a macro, use symbols within a macro, conditionally process parts of the macro, and store macros in a macro library.
- "Chapter 24. Processing GML": A description of how to create 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* and the *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide.*
- "Chapter 25. Verifying Spelling": A description of how to verify spelling using the SPELLCHK option. This chapter also includes a discussion of how to build user and addenda dictionaries.
- "Appendix A. Using SCRIPT/VS with Other Programs": A discussion of SCRIPT/VS compatibility with other programs. Also included in this chapter is a discussion of SCRIPT/VS as a postprocessor and as a preprocessor.
- "Appendix B. Improving System Performance": A discussion of the consumption and use of system resources in formatting documents.

Related DCF and DLF Publications

- *Document Composition Facility: SCRIPT/VS Language Reference*, SH35-0070. This manual describes the SCRIPT command options and the SCRIPT/VS control words and provides a summary of system symbols, special characters, character sets, and 3800 Printing Subsystem fonts.
- *Document Composition Facility and Document Library Facility General Information Manual*, GH20-9158. This manual describes the Document Composition Facility and Document Library Facility program products and summarizes their functions

and capabilities. It also summarizes the operating environment requirements for these products.

- *Document Composition Facility: Generalized Markup Language Starter Set User's Guide*, SH20-9186. This manual provides an introduction to GML and a primer on document markup using 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 Starter Set Implementation Guide*, SH35-0050. This manual describes how to modify, design, and add to the GML starter set.[2]
- *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 Guide*, SY35-0067. This manual is for IBM service personnel and customers who diagnose programming errors.
- *Document Composition Facility Messages*, SH35-0048. This manual documents SCRIPT/VS messages and suggests actions to be taken in response to these messages.
- *Document Composition Facility: SCRIPT/VS Text Programmer'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: Generalized Markup Language Quick Reference*, SX26-3719. This reference card summarizes the GML starter set and how to use SCRIPT/VS in each interactive environment.

Restricted Materials

- *Document Composition Facility Diagnosis Reference*, LY35-0068. This manual is for IBM service personnel and customers who diagnose programming errors. It describes the logic of the DCF program product and lists pertinent control blocks and data areas.

Related Publications

- *IBM Virtual Machine Facility/SP: 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:
  - *IBM Virtual Machine/System Product: CP Command Reference for General Users*, SC20-6211

---

[2] **Field Engineering support and maintenance is provided *only* on the unmodified GML starter set. If you modify the starter set, it is recommended that you also maintain an *unmodified* starter set for diagnostic purposes.**

- *IBM Virtual Machine/System Product: CMS User's Guide*, SC19-6210
- *IBM Virtual Machine/System Product: CMS Primer*, SC24-5236
- *IBM Virtual Machine/System Product: CMS Command and Macro Reference*, SC20-6209
- *IBM Virtual Machine/System Product: Terminal User's Guide*, SC20-6206.

- *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.

Related Printer Publications

- *Introducing the IBM 3800 Printing Subsystem and Its Programming*, GC26-3829. This manual provides general information about the 3800 Printing Subsystem. It describes what the 3800 Printing Subsystem is and provides information about the standard and optional features available for the 3800 Printing Subsystem.
- *IBM 3800 Printing Subsystem Programmer's Guide*, GC26-3846 (for OS/VS2 MVS) and GC26-3900 (for VSE). These manuals include detailed information about programming for the 3800 Printing Subsystem.
- *IBM 3800 Model 3 Printing Subsystem Programmer's Guide: Compatibility*, SH35-0051. This manual provides detailed information about programming for the IBM 3800 Model 3 Printing Subsystem in compatibility mode.
- *Graphical Data Display Manager (GDDM) and Presentation Graphics Feature (PGF) General Information*, GC33-0100. This manual describes the program product and its installation and storage requirements.
- *Composed Document Printing Facility: General Information*, GC33-6133. This manual introduces the CDPF program product.
- *Composed Document Printing Facility: Operation*, GC33-6135. This manual explains how to set up, operate, and service the CDPF program product.
- *Composed Document Printing Facility: Data Stream Interface, Typographic Fonts Interface*, GC33-6134. This manual contains the information those customers will need who want to generate their own input to the CDPF program.

- *IBM 3800 Printing Subsystem Model 3 Introduction*, GA32-0049. This manual provides general information about the 3800 Printing Subsystem Model 3 and the program products associated with it.
- *IBM 3800 Printing Subsystem Models 3 and 8 Programmer's Guide*, SH35-0061. This manual provides information for customer personnel who install the Print Services Facility (PSF) and Print Utility Program Products.
- *IBM 3800 Printing Subsystem Models 3 and 8: Preparing Fonts for Printing*, SH35-0082. This manual gives all procedures necessary to prepare IBM-supplied fonts for use in the printing of documents.
- *IBM 3800 Printing Subsystem Model 3 Font Catalog*, SH35-0053. This manual gives print samples of the fonts available for the 3800 Printing Subsystem Model 3 printer.
- *IBM 3800 Printing Subsystem Model 8 Font Catalog*, SH35-0054. This manual gives print samples of the fonts available for the Printing Subsystem Model 8 printer.
- *IBM 3820 Page Printer and Advanced Function Printing Software: Introduction and Planning Guide*, GBOF-1189 (MVS). Through a series of booklets, this publication introduces the 3820; provides planning information for its data network, physical location, and software; and presents specifics about ordering 3820 supplies and preparing an implementation plan in an MVS environment.
- *IBM 3820 Page Printer and Advanced Function Printing Software: Introduction and Planning Guide*, GBOF-1666 (VSE). Through a series of booklets, this publication introduces the 3820; provides planning information for its data network, physical location, and software; and presents specifics about ordering 3820 supplies and preparing an implementation plan in a VSE environment.
- *IBM 4250 Printer Operator's Guide*, GA33-1551. This manual gives instructions and procedures for operating the 4250 printer.
- *IBM 4250 Printer Font Catalog*, G520-0004. This manual gives print samples of the fonts available for the 4250 printer.
- *Print Management Facility User's Guide and Reference*, SH35-0059. This manual gives guide information for system and application programmers using the Print Management Facility.
- *A Guide to Using IBM Printers for Advanced Function Printing*, S544-3095. This manual describes the use of a program product (PSF, DCF, GML, OGL, GDDM, and PMF) and the use of a subset of a program product in conjunction with the IBM APF printers available, including the 3800 Printing Subsystem Model 3 and the 3820 Page Printer.
- *Print Services Facility User's Programming Guide*, S544-3084. This manual describes, for the application programmer, the abilities of an all-points-addressable printer and the tasks associated with such a printer, including how to use JCL to produce output.

Publication Library Guide for the Document Composition Facility

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

User Tasks	Typical Audience	Recommended Books	Brief Description
Planning and introducing DCF/DLF	Users, system planners	DCF and DLF General Information (GH20-9158)	Provides a general overview of text processing, library facility, and available books
Formatting documents (using the GML starter set)	Novice user to experienced end users	DCF: GML Starter Set User's Guide (SH20-9186) DCF: GML Starter Set Reference (SH20-9187) DCF Messages (SH35-0048)	Provide an introduction to the GML starter set, describe the GML starter set tags and SCRIPT/VS messages
Formatting documents (using SCRIPT/VS control words)	Knowledgeable to experienced end users	DCF: SCRIPT/VS Text Programmer's Guide (SH35-0069) DCF: SCRIPT/VS Language Reference (SH35-0070) DCF Messages (SH35-0048)	Describe the function and use of all SCRIPT/VS control words, SCRIPT/VS macros, SCRIPT diagnostic aids, and the formatting features and messages of SCRIPT/VS
Modifying GML starter set[3]	Document administrator and text programmer[4]	DCF: GML Starter Set Implementation Guide (SH35-0050) DCF: GML Starter Set User's Guide (SH20-9186) DCF: GML Starter Set Reference (SH20-9187) DCF: SCRIPT/VS Text Programmer's Guide (SH35-0069) DCF: SCRIPT/VS Language Reference (SH35-0070)	Contain material on: GML starter set tags, SCRIPT/VS control words, and how to modify the GML starter set

---

[3] **Field Engineering support and maintenance is provided *only* on the unmodified GML starter set. If you modify the starter set, it is recommended that you also maintain an *unmodified* starter set for diagnostic purposes.**

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

User Tasks	Typical Audience	Recommended Books	Brief Description
Creating GML application processing functions	Document administrator and text programmer[4]	DCF: GML Starter Set Implementation Guide (SH35-0050) DCF: GML Starter Set User's Guide (SH20-9186) DCF: GML Starter Set Reference (SH20-9187) DCF: SCRIPT/VS Text Programmer's Guide (SH35-0069) DCF: SCRIPT/VS Language Reference (SH35-0070) DCF: GML Concepts and Design Guide (SH20-9188)	Provide information on: how to design your own GML, GML concepts, GML starter set tags, SCRIPT/VS control words, and usage guidelines
Installing, modifying, and maintaining DCF	Systems programmer	DCF Program Directory DCF: SCRIPT/VS Text Programmer's Guide (SH35-0069) DCF: SCRIPT/VS Language Reference (SH35-0070) DCF Diagnosis Guide (SY35-0067) DCF Diagnosis Reference (LY35-0068) DCF Messages (SH35-0048)	Give information on error isolation, program tailoring, and use of SCRIPT/VS

Note: As an aid to Document Composition Facility users, the following reference cards are also available:

Document Composition Facility: SCRIPT/VS Text Programmer's Quick Reference, SX26-3723
Document Composition Facility: Generalized Markup Language Quick Reference, SX26-3719.

Table of Contents

List of Illustrations

Part 1. Basic Information about SCRIPT/VS

This part of the book contains general information about SCRIPT/VS.

Included in this section are the following chapters:

- Chapter 1 - An Overview of SCRIPT/VS
- Chapter 2 - Using the SCRIPT Command
- Chapter 3 - Marking Up Documents with SCRIPT/VS
- Chapter 4 - Combining SCRIPT/VS Input Files
- Chapter 5 - Communicating with SCRIPT/VS.

Chapter 1. An Overview of SCRIPT/VS

SCRIPT/VS can format documents that include SCRIPT/VS control words and Generalized Markup Language (GML) tags as well as text. SCRIPT is the command that you use to invoke the SCRIPT/VS program. SCRIPT/VS can be used with several operating systems in both the interactive foreground and the batch background environments.

Operating Environments

With the Foreground Environment Feature of the Document Composition Facility installed, the SCRIPT/VS text formatting program can execute in the following interactive environments:

- The Conversational Monitor System (CMS) of the IBM Virtual Machine Facility/System Product (VM/SP)
- 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 (note that CICS/VS here refers to both the CICS/OS/VS and CICS/VSE program products).

With the Document Library Facility (DLF) installed, SCRIPT/VS can execute in the following background environments:

- OS/VS2 MVS
- VSE.

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

- Files created by the ATMS-III, CMS, TSO, or ISPF editors
- A word processing system attached to the host system using a telecommunications network
- A user-written program that calls DLF as a subroutine.

SCRIPT/VS Input Files

SCRIPT/VS reads input data containing text and control information, formats the data into pages, and produces formatted output for a system printer or other suitable output device. The following kinds of information may appear in SCRIPT/VS input files:

- Text. This is the content of the document.
- Symbols. These are character strings that begin with an ampersand (&) and 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 the string 3800 Printing Subsystem.

- Control words. These are two-letter codes 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. These 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 that contains a .PA control word followed by a .CE control word. Then, anytime the .TOP macro is encountered in the input file, 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. This is a formatting language that uses tags to identify the associated text as a particular part of a document, such as paragraph or heading. GML (Generalized Markup Language) provides the syntax and usage rules for marking up a document[5] and allows you to develop a vocabulary of tags for describing your documents. For example, in the GML starter set provided with the Document Composition Facility, the tag :p identifies a paragraph. The tag is identified by the GML delimiter, which is by default the colon (:).

A SCRIPT/VS input file is usually 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.

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 can 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 learn.

Markup Languages

The Document Composition Facility provides two languages for marking up text: SCRIPT/VS and Generalized Markup Language.

SCRIPT/VS Control Word Language

Markup in the SCRIPT/VS language consists of entering SCRIPT/VS control words that direct the SCRIPT/VS formatter.

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

---

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

be regarded as text. A .* at the start of an input line identifies a comment line that does not appear in the output.

Each input line is scanned from left to right for a control word separator, which is initially a semicolon (;). If a control word separator is found and the next character is a period, the character string to the left of the control word separator is processed; the character string to its right is saved. (The character strings can be control words.) This process is repeated until the input line is completely scanned. For example,

```
.sk 7.5mm;.in 10mm for 3
```

causes approximately seven and one-half millimeters of vertical space to be skipped before the next output lines are printed (.sk 7.5mm). It also causes the next three output lines to be indented ten millimeters (.in 10mm 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 can be 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 be centered. For example,

```
.ce 10
```

See the *Document Composition Facility: SCRIPT/VS Language Reference* for descriptions of the SCRIPT/VS control words and associated parameters.

Generalized Markup Language

Generalized Markup Language (GML) provides the syntax and usage rules 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 and causes that figure element to be enclosed in a box.

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

SCRIPT/VS interprets the GML markup and invokes the correct application processing function (APF) for the element the markup describes. In SCRIPT/VS, APFs are implemented as sets of control words in the form of macro definitions. The macro definitions are usually contained in a macro library. The association, or mapping, between the GML markup and the APFs is usually made in a document called a profile. The profile 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 processing results of GML starter set markup. The *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide* discusses the GML starter set profile and macro library and how you can modify the GML starter set.

Logical Output Devices and Output Destinations

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 formats text for printing on terminals, impact printers, or nonimpact printers.

When SCRIPT/VS formats a document, it takes into consideration the characteristics of the specific physical output device, called the *logical output device*. This may be a terminal, a line printer, or a nonimpact page printer. The output devices SCRIPT/VS supports are:

- 2741 typewriter terminal
- 3270 display terminal
- 1403 printer
- 3800 Printing Subsystem Model 1 and Model 3 (in compatibility mode)
- 3800 Printing Subsystem Model 3 (page printing mode)
- 4250 printer
- 3820 Page Printer.

A logical device, which is specified with the DEVICE option of the SCRIPT command, includes a physical device type, a page size, and a number of lines per vertical inch (for line devices). For example, the 3800N8 logical device is a 3800 Printing Subsystem, at 8 lines-per-inch on 8-1/2 by 11 inch paper.

SCRIPT/VS can also be directed to send the output to a specific destination such as a disk file or the terminal, regardless of the logical device for which it was formatted. You can specify almost 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. See the *Document Composition Facility: SCRIPT/VS Language Reference* for a complete list of available logical devices.

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.

Defaults and Initial Settings

SCRIPT/VS can format an input file without any control words or GML tags specified. In such cases, the initial settings for page dimensions and formatting controls are used. The initial settings are associated with the logical device specified with the DEVICE option of the SCRIPT command. The initial settings for each logical device are documented in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Vertical and Horizontal Space Units

In SCRIPT/VS you can specify vertical or horizontal dimensions or distances. As Figure 3 on page 44 illustrates, these dimensions can be expressed in any of several different space units:

- Centimeter
- Character (Horizontal)
- Cicero
- Device Unit (Horizontal)
- Device Unit (Vertical)
- Em-space (Horizontal)
- Em-height (Vertical)
- Inch
- Line (Vertical)
- Millimeter
- Pica.

Fonts

In SCRIPT/VS, a *font* is a set of characters having the same vertical size and type style. Fonts may be *fixed-pitch*, wherein all characters have the same pitch (or width); *mixed-pitch*, wherein the characters are a mixture of 10-, 12-, and 15-pitch characters; or *proportional*, wherein the characters have different widths.[6]

The *pitch* of a font is the number of characters per inch in a line of printed text. Certain output devices, such as the 3270 Display Terminal, are capable only of printing *fixed-pitch* fonts. Other devices, such as the 3800 Printing Subsystem are capable of printing *fixed-pitch* and *mixed-pitch* fonts. The 3800 Printing Subsystem has three pitch values:

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

Output devices such as the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer are capable of printing *fixed-pitch*, *mixed-pitch*, and *proportional* fonts. A wide variety of typographical fonts are available for use with these page printers. Any of these fonts may be used with SCRIPT/VS. See "Chapter 15. Selecting Fonts" on page 175 for more details on specifying fonts for particular output devices.

Capitalization and underscoring can be used to create different logical fonts on all devices, and overstriking allows you to print boldface text on impact printers and typewriter terminals. (See "Defining Fonts" on page 178.)

For details about 4250 printer fonts, see the *IBM 4250 Printer Type Font Catalog* and the appropriate Composed Document Print Facility (CDPF) documents. (See "Related Printer Publications" on page vii.) For details about 3800 Printing Subsystem Model 3 and 3820 Page Printer fonts, see the *3800 Printing Subsystem Model 3 Font Catalog*, the *IBM 3800 Printing Subsystem Models 3 and 8 Programmer's Guide*, and *IBM 3800 Printing Subsystem Models 3 and 8: Preparing Fonts for for Printing.*

---

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

Imbedding Files

You can combine many SCRIPT/VS input files for processing as a single document.

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 global formatting controls for the entire document and the .IM [Imbed] control words that imbed the other files into the master file.

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 47.

SCRIPT/VS Utility Files

SCRIPT/VS creates a number of utility files when it encounters certain control words (such as .WF [Write To File]), or command options, or both. You have the option of defining or redefining these files using the .DD [Define Data File-id] control word.

Communicating with SCRIPT/VS

You can communicate with SCRIPT/VS in order to determine error conditions and to interactively process your documents.

SCRIPT/VS issues program messages accompanied by a severity level code when certain error conditions are encountered.

By using certain SCRIPT/VS control words, you can interact with SCRIPT/VS before your input text is in final form and while your document is being formatted. For details, see "Chapter 5. Communicating with SCRIPT/VS" on page 59.

SCRIPT/VS Functions

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

Formatting Functions

SCRIPT/VS provides you with many text formatting functions including line composition, page composition, head levels, table of contents generation, boxes, keeping text together, footnotes, character translation, indexing, and hyphenation and spelling verification.

Composing Lines

You can control many functions of line composition including the following:

Line Formatting You can specify concatenation, justification, centering, and left or right alignment. For details, see "SCRIPT/VS Text Formatting" on page 71.

Indenting You can specify 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 "Indenting" on page 78.

Tabs You can specify tab positions. Tab characters may be resolved into a number of blanks or into a string of another character. For details, see "Using Tabs" on page 84.

Revision Codes You can select the placement of as many as 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 a document. For details, see "Revision Codes" on page 95.

Fonts You can select fonts for different portions of text, both in the body and in running headings and footings. For details, see "Chapter 15. Selecting Fonts" on page 175.

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 fonts for emphasis. For details, see "Underscoring and Capitalization" on page 187.

Hyphenating and Horizontally Justifying Lines

You can determine if and how hyphenation should be done, how large a word must be before it can be hyphenated, and how much of a word must be left at the beginning or ending of a line. You can use an algorithmic hyphenator[7] to further extend SCRIPT/VS's hyphenation capabilities. Text can be justified horizontally to avoid hyphenation, to achieve fully justified lines within the left and right margins, or to reduce white space in a line. For details, see "Chapter 7. Hyphenating and Horizontally Justifying Text" on page 99.

Creating Vertical Space

You can specify the amount of space left between output lines, including the reservation of space for drop-in art. For details, see "Chapter 8. Creating Vertical Space" on page 105.

Vertically Justifying Text

You can perform column balancing in order to distribute text evenly among columns and you can vertically justify your text in order to achieve fully justified lines within the top and bottom margins. For details, see "Chapter 9. Vertically Justifying Text" on page 111.

---

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

Laying Out Pages

You can specify page dimensions, the number of columns per page and running headings and footings.

Page composition includes:

Margins You can specify the size of the top and bottom margins as well as the left and right margins. For details, see "Default Page Dimensions" on page 119.

Headings and Footings You can create running headings and footings that will be printed on all pages or different ones for odd and even pages. For details, see "Running Headings and Footings" on page 127.

Columns You can define the number of columns, their size and their placement on a page. For details, see "Chapter 12. Composing Multiple-Column Pages" on page 141.

Named Areas

You can define and place text into *named* areas. These areas can be positioned anywhere on the page. For details, see "Chapter 11. Placing Text in Named Areas" on page 133.

Creating Head Levels

You can specify as many as seven head levels for distinctive formatting of headings that represent different levels of topics. Distinctive formatting includes before and after spacing, font selection, overstriking, capitalization, underscoring, and text alignment. For details, see "Chapter 13. Creating Head Levels and Table of Contents" on page 149.

Creating a Table of Contents

You can specify 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 accompanying head levels and automatically supplies the page number. You can also specify phrases other than the text accompanying head levels to appear in the table of contents. For details, see "Chapter 13. Creating Head Levels and Table of Contents" on page 149.

Creating Boxes and Rules

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.

You can define *named* rules of varying thicknesses and place horizontal and vertical rules in the current column. For details, see "Chapter 14. Creating Rules and Boxes" on page 157.

Selecting Fonts

You can select coded fonts for use with line devices and page printers. With page printers, you can also define and select fonts from font families found in a font library. For details, see "Chapter 15. Selecting Fonts" on page 175.

Keeping Text Together

SCRIPT/VS processing includes functions that keep text together to improve the appearance of output. For example, SCRIPT/VS 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 top or bottom of a column or page. For details, see "Chapter 16. Keeping Blocks of Text Together" on page 191.

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 "Chapter 16. Keeping Blocks of Text Together" on page 191.

Footnotes

You can have SCRIPT/VS save text indicated as a footnote and place it at the bottom of the page.[8] Subsequent footnotes are placed below it. For details, see "Chapter 17. Creating Footnotes" on page 197.

Translating Characters

You can define specific character mappings so that SCRIPT/VS will perform character translations on input and output lines as part of its normal processing. See "Chapter 18. Translating Characters" on page 201.

Indexing

You can include index entries in the body of your document at their points of reference. SCRIPT/VS uses these index entries to generate an index for your document that includes appropriate page numbers for all of the entries. For details, see "Chapter 19. Creating an Index" on page 207.

SCRIPT/VS Programming Facilities

SCRIPT/VS provides several programming facilities that enable you to specify the SCRIPT/VS formatting environment, process input conditionally, process symbols and macros, and process Generalized Markup Language tags.

You can set the values and parameters of the formatting environment in order to specify exactly how you want SCRIPT/VS to format each line on an output page.

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. For details, see "Chapter 22. Processing Logical Statements" on page 253.

---

[8] Like this.

Processing Symbols and Macros

You can define symbols and macros for substitution during processing. Symbols are used in many ways: 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 macros, which are sets of SCRIPT/VS control words. For example, you might redefine a particular head level macro to alter the SCRIPT/VS formatting style. For details, see "Chapter 21. Processing Symbols" on page 223 and "Chapter 23. Processing Macros" on page 261.

Symbols and macros are used to support the Generalized Markup Language. The *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide* discusses how symbols and macros are used to create the GML starter set.

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, manipulating attributes, and processing symbols. For details, see "Chapter 24. Processing GML" on page 277.

GML Starter Set Application

The Document Composition Facility 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 either as an example of one way to support GML or you can further enhance the starter set by adding your own tags to suit the needs of your own unique documents. The *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide* documents how the GML starter set is constructed and illustrates how you can modify it.

Verifying Spelling

You can specify whether or not words 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 spelling validity (and hyphenation points) 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. For details, see "Chapter 25. Verifying Spelling" on page 289.

General Document Handling Functions

SCRIPT/VS provides several document handling functions. These are discussed in the sections immediately below.

Saving Input Lines for Subsequent Processing

You can determine whether certain input lines will be written to a file. For details, see "Chapter 4. Combining SCRIPT/VS Input Files" on page 47.

Specifying the Destination of Output

You can specify 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 the discussion of SCRIPT command options in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Printing Part of the Output Document

You can specify whether every page, a single page, or a range or ranges of pages is to be included in the formatted output. For details, see the PAGE option in the summary of SCRIPT command options in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Processing Interactively During Formatting

In an interactive environment (CMS or TSO), you can affect SCRIPT/VS as it processes by entering text or markup at 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 61.

Converting ATMS Documents

If the IBM Document Library Facility program product is installed with SCRIPT/VS, you can convert most ATMS-II or ATMS-III markup to similar or equivalent SCRIPT/VS markup. For details, see the *Document Library Facility Guide* and "The ATMS Conversion Routine" on page 311.

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.

Calling the SCRIPT/VS Processor

You call the SCRIPT/VS processor by issuing the SCRIPT command and specifying the name of the file SCRIPT/VS is to process.

Interactive Environments

In one of the three interactive environments that support SCRIPT/VS, use one of the following formats for the SCRIPT command:

- 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 detail in the SCRIPT command options section of the *Document Composition Facility: SCRIPT/VS Language Reference.*

Batch Environments

For details about calling SCRIPT/VS in a batch environment, see the *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 invoke DLF as a subroutine which in turn invokes SCRIPT/VS. For details, see the *Document Library Facility Guide.*

Using SCRIPT/VS as a Preprocessor

SCRIPT/VS can be used to prepare an input file for use as input to another text program such as the STAIRS/VS program product. For details, see "Producing Input for STAIRS/VS" on page 309.

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 to be 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 headings 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?
- Do you need a special size or style of type for your documentation? Will you need different types for body text? Headings? Footnotes? And so on?
- Are you going to have illustrations? Are you going to create boxes and rules 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 do not have to reenter all the control words in sequence each time?
- Do you want your output lines fully justified? Do you want them left-aligned or right-aligned? Do you want to balance your columns by distributing text? If you want your columns justified, what are your leadout, skip, space and text linespacing requirements and allowable variations?

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.

Note: The purpose of the examples in this book is to illustrate various formatting techniques using the SCRIPT/VS control words. Because of various factors, such as column line length, hyphenation dictionaries, and algorithmic hyphenators, example results may not always be identical to that shown. However, the effect of the control word will be the same.

Some of the examples in this book are formatted using a predefined column line length of 30; others are formatted to the actual line length of the column, except when a specific column line length is given using the .CL control word.

Chapter 2. Using the SCRIPT Command

You can use the SCRIPT command and its options to process and format an input file, either in an interactive or batch environment.

Using SCRIPT/VS in the Interactive Environment

If you want to process and format an input file in the interactive environment, you simply issue the SCRIPT command along with the necessary options to control processing. SCRIPT/VS formats the input file using GML tags, macros, control words, and text that are included in the file.

The SCRIPT command can be issued as a CMS command, a TSO command, or an ATMS-III command. The format of the SCRIPT command is the same for each system, except that in TSO options must not be placed in parentheses and in ATMS-III the SCRIPT command itself must be entered in lowercase. The forms of the SCRIPT command are as follows:

In CMS,

```
SCRIPT   file-id  [ ( options... ]
         ?
```

In TSO,

```
SCRIPT   file-id [ options... ]
         ?
```

In ATMS-III,

```
script   file-id [ [(] options... ]
         ?
         *
```

where:

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

file-id is the name of the primary input file. When the input file contains imbedded or appended files, *file-id* 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.

***** is the document in ATMS-III working storage.

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. The left parenthesis "(" preceding the option list is required in the CMS environment.

Naming the Primary Input File

The format of the name you specify for *file-id* depends on the environment from which you call SCRIPT/VS. Except when using SCRIPT in the TSO environment, 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, because 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)

ATMS-III Naming Conventions

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

```
script *
```

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

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

This results 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 always results 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 requesting user's permanent storage.

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 can 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 always adds an appropriate destination option, such as PRINT or CTF, to the user's options when the peripheral queue is processed. TERM is always added when the SCRIPT command is issued from a terminal.

Characteristics of an Input File

The following are characteristics of input files that can be processed by SCRIPT/VS:

- In a CMS environment:
  - A filetype of SCRIPT
  - As many as 65,535 fixed- or variable-length records, with a maximum of 132 bytes per record
  - Include uppercase and lowercase letters, numbers, and special characters
  - Do not contain line numbers; however, if the lines are numbered, the number must be in positions 1 to 8 of each record. When the input file is processed, line numbers are ignored.
- In a TSO environment:
  - Data set organization of PO (partitioned organization) or PS (physical sequential)
  - Composed of fixed- or variable-length records, blocked or unblocked, with a maximum of 132 bytes per record
  - Include uppercase and lowercase letters, numbers, and special characters
  - Contain records with or without line numbers; if the input lines are numbered, the numbers are ignored if:
    - A variable-length record has the line number in the first eight positions of each record.
    - A fixed-length record has the line number in the last eight positions of each record.
- In an ATMS-III environment:
  - Contained in ATMS-III working or permanent storage
  - Composed of variable-length records, with a maximum of 230 text characters per record
  - Include uppercase and lowercase letters, numbers, and special characters
  - Include ATMS-III page and unit numbers that are not included in the 230 text characters.

Using SCRIPT/VS in the Batch Environment

With the Document Library Facility (DLF) installed, SCRIPT/VS can be used in a batch environment under OS/VS2 MVS and VSE. Using the DLF SCRIPT command, input files can be formatted with the SCRIPT/VS formatter in a batch environment. Input files are usually stored as documents in the library, or as external data sets to DLF. The output can be directed to either a printer or to an external data set.

SCRIPT/VS files stored as documents in the library are specified by:

- A three-part document identifier, which includes:
  - Library number
  - Document name
  - Password
- Data name
- Version number.

SCRIPT/VS files stored as external data sets to DLF are specified with a ddname or dataset name via the FROM operand of the DLF SCRIPT command.

Environment Restrictions

Depending on the environment in which you are using SCRIPT/VS, certain SCRIPT command options and control word parameters are restricted.

- In the CMS environment, the following should not be used:
  - The DEST command option
  - Suboptions of the PRINT command
  - The DD, DSN, CATALOG, DATA, VERSION, PROC, and PARM parameters of the .DD [Define Data File-id] control word.
- In the TSO environment, the following should not be used:
  - The OPTIONS, TLIB, and @user-option command options
  - The DATA, VERSION, PROC and PARM parameters of the .DD [Define Data File-id] control word.
- In the ATMS-III environment, the following should not be used:
  - The DEST, FILE, NOSPIE, NOWAIT, STOP, TLIB, and @user-option SCRIPT command options
  - The PROMPT option of the PAGE command option
  - The suboptions of the PRINT command option
  - The DD, DSN, CATALOG, DATA, VERSION, PROC and PARM parameters of the .DD [Define Data File-id] control word
  - In VSE, the SEGLIB option should not be used.

- In the DLF Environment, the following should not be used:
  - The LIB, QUIET, NOSPIE, STOP, and TERM options of SCRIPT/VS
  - In VSE, the SEGLIB option should not be used.

  The DSN... parameter of the FROM suboption (which specifies the name of the data set used for input when the document library is not the source) and the DSN... parameter of the FILE option (which specifies the name of the data set used for output) are valid in OS/VS2 MVS only.

The SCRIPT Command Options

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

The name of each option can be shortened to its minimum abbreviation. In TSO, ambiguous truncations are not accepted and you are prompted to reenter the option. In other systems, ambiguous truncations are accepted and interpreted as shown in Figure 1 on page 23.

Default Options

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

For CMS,

```
TERM PROFILE (PROFILE) LIB (DSMGML3) NOCONT NODDUT
```

For TSO,

```
TERM PROFILE (PROFILE) LIB ('SCRIPT.R30.MACLIB') NOCONT NODDUT
```

For ATMS-III,

```
TERM PROFILE (PROFILE) MESSAGE (DELAY) NOCONT
```

For batch,

```
PRINT PROFILE (PROFILE) MESSAGE (DELAY) NOCONT
```

All other options must be explicitly specified when desired.

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 following groups of options, 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.

The mutually exclusive options are:

- PROFILE and NOPROF. The PROFILE option specifies that a file is to be imbedded before the primary input file is processed; the NOPROF option specifies that no profile is needed, respectively.
- CTF, FILE, PRINT, and TERM. These options specify the destination of the formatted output. If a logical output device is not also specified, SCRIPT/VS selects one, based on the destination. If CTF is specified and the device type is not STAIRS, CTF is ignored. Figure 2 on page 25 lists the default logical device for each destination.
- CONTINUE and NOCONT. These options determine whether processing is to continue after SCRIPT/VS detects an error condition and issues an error message. Even if CONTINUE has been specified, SCRIPT/VS will stop processing if a *severe* or *terminal* error is encountered.
- DDUT and NODDUT. These options determine if SCRIPT/VS utility file redefinition to a non-utility file is allowed. The redefinition is disallowed by NODDUT. DDUT allows redefinition.
- SEGLIB and NOSEGLIB. These options determine whether the segment library is to be searched for a specified segment. If the CONTINUE option has been specified, SCRIPT/VS will continue processing even if the specified segment does not exist.
- SYON and SYOFF. These options determine if the .SY [System Command] control word is enabled or disabled. SYOFF disables .SY [System Command] and SYON enables it.

Descriptions of all of the SCRIPT command options can be found in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Logical Output Devices and Destinations

SCRIPT/VS can format a document for a number of different output devices, including the 1403 printer, the 3800 Printing Subsystem, and the 4250 printer. During formatting, SCRIPT/VS takes into consideration the characteristics of the specific output device.

SCRIPT/VS always formats documents for some specific logical output device. A logical output device is a combination of a physical device type, such as the 3800 Printing Subsystem, form size, such as 8 1/2 by 11 inches, and a specific lines-per-inch specification, such as 6 or 8. The logical device type is specified with the DEVICE option of the SCRIPT command. For example, you can direct SCRIPT/VS to format a document for a 3800 Printing Subsystem standard page size (8-1/2 by 11 inches) at 8 lines-per-inch with the following command.

```
SCRIPT TEST ( DEVICE(3800N8)
```

See Figure 2 on page 25 for a list of the logical devices that SCRIPT/VS can format for.

If the DEVICE option is not specified with the SCRIPT command, SCRIPT/VS uses a default logical device, usually 1403W6.[9]

The logical device that SCRIPT/VS uses in formatting the document is independent of the the actual destination of the formatted output. For example, you can not only direct SCRIPT/VS to format for a 3800 Printing Subsystem but also tell it to put the format-

[9] This default device can be changed by the installation.

Option	non-TSO Environments	TSO Environment
BIND	B	B
CHARS	C	CH
CONTINUE	CO	CO
CTF	CT	CT
DDUT	DD	DD
DEST	DE	DES
DEVICE	D	DEV
FILE	F	FI
FONTLIB	FO	FO
INDEX	I	I
LIB	L	L
MESSAGE	M	M
NOCONT	NOC	NOC
NODDUT	NOD	NOD
NOPROF	N	NOP
NOSEGLIB	NOSE	NOSE
NOSPIE	NOS	NOSP
NOWAIT	NOW	NOW
NUMBER	NU	NU
OPTIONS	O	
PAGE	P	PA
PRINT	PR	PRI
PROFILE	PRO	PRO
QUIET	Q	Q
SEARCH	S	SEA
SEGLIB	SEG	SEG
SPELLCHK	SP	SP
STOP	ST	ST
SYOFF	SYOF	SYOF
SYON	SYON	SYON
SYSVAR	SYS	SYS
TERM	T	TE
TLIB	TL	TL
TWOPASS	TW	TW
UNFORMAT	U	UN
UPCASE	UP	UP

Figure 1. Minimum Abbreviations of SCRIPT Options.

ted output in a disk file, rather than send it to the printer. There are several options of the SCRIPT command that specify the destination of the output. These are:

- FILE (put it in a disk file)
- TERM (send it to your terminal)
- PRINT (send it to the printer)
- CTF (put it in a disk file in Condensed Text Format for the STAIRS program product).

You can specify almost any combination of output destination and logical device. For example, if you specify

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

then SCRIPT/VS formats a document for the 3800 Printing Subsystem at 8 lines-per-inch but saves the output in a disk file for later printing, if you so request, on a physical printer.

Note: There are two exceptions to this rule:

1. The CTF destination is valid only for the STAIRS logical device and is ignored if any other logical device is specified.
2. The PRINT option is not valid for 4250 printer logical devices.[10]

Additionally, certain destinations are invalid in certain environments. See the descriptions of the FILE, PRINT, TERM and CTF command options in the *Document Composition Facility: SCRIPT/VS Language Reference* for more details.

If you specify only a logical device with the DEVICE option, SCRIPT/VS assumes an appropriate output destination. For example, if you specify a 1403 logical device, SCRIPT/VS may send the output to the printer. If you specify a 4250 printer logical device, SCRIPT/VS may file the output for you in a disk file.[10]

Similarly, if you specify an explicit output destination, SCRIPT/VS assumes an appropriate logical device. If you specify neither a destination nor a logical device, SCRIPT/VS formats the document for and sends it to your terminal. The logical output device and output destination for a document when various combinations of options are specified are shown in Figure 2 on page 25.

Printer Classes

SCRIPT/VS supports two basic classes of printer devices: line printers and page printers. A line printer or line device is any printer that accepts one line of text from the host system at a time. SCRIPT/VS supports such line devices as the 1403 printer, the 2741 typewriter terminal, and the 3800 Printing Subsystem Model 1. A page printer is any printer that accepts composed pages, which are constructed of composed text and images, among other things. SCRIPT/VS supports the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer.

Note: The Document Composition Facility (DCF) and the Generalized Markup Language (GML) starter set require the following font program products be installed for the 4250 printer:

- 5771-AAR Monotype Times New Roman[11]
- 5771-AAW Typewriter and Pi

and DCF requires tne following font program products be installed for the 3800 Printing Subsystem Model 3 and the 3820 Page Printer:

---

[10] In the ATMS-III environment, ATMS-III will cause the output for a 4250 printer to be sent to a CICS/VS extra partitioned dataset.

[11] Trademarks of The Monotype Corporation, Limited.

- 5771-ABA Sonoran Serif[12]
- 5771-ABC Pi and Specials.

You may tailor DCF and/or use the CHARS option of the SCRIPT command to point to typeface families other than the required ones listed here.

Options Specified	Logical Device	Physical Device	Output Destination
none [Foreground]	TERM	2741 or 3270	Terminal
none [Background]	1403W6	1403	Printer
CTF	STAIRS	1403	(1)
FILE	1403W6	1403	File
PRINT	1403W6	1403	Printer
TERM	TERM	2741 or 3270	Terminal
DEVICE(1403xx)	1403xx	1403	Printer
DEVICE(2741)	2741	2741	Terminal
DEVICE(3270)	3270	3270	Terminal
DEVICE(38PPxxxx)	38PPxxxx	3800-3	Printer
DEVICE(3800xx)	3800xx	3800	Printer
DEVICE(3820xx)	3820xx	3820	Printer
DEVICE(4250xx)	4250xx	4250	File (2)
DEVICE(STAIRS)	STAIRS	1403	(1)
CTF DEVICE(devtype)	devtype	device	(3)
FILE DEVICE(devtype)	devtype	device	File
PRINT DEVICE(devtype)	devtype	device	Printer
TERM DEVICE(devtype)	devtype	device	Terminal
CTF DEVICE(STAIRS)	STAIRS	1403	(1)
FILE DEVICE(STAIRS)	STAIRS	1403	File (4)
PRINT DEVICE(STAIRS)	STAIRS	1403	Printer (4)
TERM DEVICE(STAIRS)	STAIRS	1403	Terminal (4)

1 The destination of Condensed Text Format output depends upon the environment:

- In CMS, TSO, and VS2: a file named DSMUTCTF
- In VSE: a file named DSMUCTF
- In ATMS-III: a CICS/VS partitioned data set.

2 In ATMS-III, the destination is a CICS/VS partitioned data set.

3 If the CTF and DEVICE options are both specified, and *devtype* is not STAIRS, CTF is ignored.

4 This output is in STAIRS Proof format.

Figure 2. 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 classes and forms must be included in these considerations. Refer to Figure 12 on page 120 for a description of all logical devices.

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

Printing on the 4250 Printer

DCF enables you to get output printed on the 4250 printer. The method you use to get this output depends on the operating environment in which you are working.

In CMS, for example, you must use the FILE option of the SCRIPT command to get 4250 printer printed output. It is assumed that you have a font library, a font library index, and the necessary 4250 printer fonts on your CMS system. If you have these necessary requisites and if you have formatted your document with SCRIPT/VS, then you can send your output to a file by entering, for example:

```
script docname (dev(4250a) file
```

Output created by this command is then placed into a file called **docname LIST4250 A**.

To get the file **docname LIST4250 A** printed, you must use the Composed Document Printing Facility (CDPF) running as an application program in a virtual machine to which the 4250 printer is attached. An example of the command to do this is:

```
bfucdpf docname list4250 a (processing options)
```

where "processing options" are CDPF command options that you want to specify, such as MESSAGE, PRINT, and so on. For more information on CDPF see the *Composed Document Printing Facility: Installation and Operation.*

In MVS, you can get 4250 printer printed output either in the foreground or in the background. In the foreground (TSO), you must first enter the command

```
script docname dev(4250a) file
```

in order to create a file called **userid.docname.LIST4250**. This assumes you used the default for your document name. If you did not use the default naming convention, then this file will be called by the name of what other preallocated data set you specified with the SCRIPT command.

Next you will have to invoke a print CLIST (command list) input file that you have created or had created for you that contains such information as the font library name, the segment library name, and the CDPF command. By entering the name of the CLIST, you cause the print to be sent to the 4250 printer.

In the background environment (DLF), you will have to specify in your batch procedure the name of the font and segment libraries to be used and that the printing is to be in page mode. The JCL statements for DLF SCRIPT command processing are as follows:

```
//SCRIPT JOB
//STEP1 EXEC PGM=DSMSPEXC,PARM='LIST'
//SYSPRINT DD SYSOUT=A
//DEV4250 DD DSN=DOCNAME.LIST4250,DISP=MOD
//FONTLIB   DD DISP=OLD,DSN=SYS1.FONT4250
//DSMLIST   DD SYSOUT=A
//DSMINDIR  DD DSN=DSMLIB.DIRECTORY,DISP=SHR
//DSMINLIB  DD DSN=DSMLIB.SOURCE,DISP=SHR
//DSMUTMSG  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTDIM  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTTOC  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTWTF  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//SYSIN DD *
  AUTH 88/CITADEL
  SCRIPT 1314151 DSMIVC30 (PROF(DSMPROF3) CONT -
                              FILE(DEV4250) DEV(4250A))
/*
//
```

These JCL statements cause the specified input dataset to be formatted to a file, **docname.LIST4250**. To then get this file to a 4250 printer for printing, you must again use CDPF as shown in the following example:

```
//jobname     JOB
//*MAIN SYSTEM=systemname,CLASS=classname
//JOBLIB      DD   DSN=load.library.name,DISP=SHR
//CDPF        EXEC PGM=BFUDCPF,REGION=nnnnK,
//  PARM='PRINT(BR,AFP001)'
//INPUT       DD   DSN=DOCNAME.LIST4250,DISP=OLD
//FONTLIB     DD   DSN=SYS1.FONT4250,DISP=SHR
//PSEGLIB     DD   DSN=SYS1.PSEG4250,DISP=SHR
//SYSPRINT    DD   SYSOUT=A
/*
```

For more information on CDPF see the *Composed Document Printing Facility: Installation and Operation.*

Printing on the 3800 Printing Subsystem Model 3

DCF enables you to print output on the 3800 Printing Subsystem Model 3. The method you use to get this output depends on the operating environment in which you are working.

It is assumed that you have a font library containing a font library index, the necessary 3800 Printing Subsystem Model 3 fonts on your CMS system, and a page segment library. If you have these necessary requisites, then you can format your output to a file by entering, for example:

```
script docname (dev(38ppn)
```

Output created by this command is then placed into a file called **docname LIST38pp A**.

Because your job can not be printed directly on CMS, you will have to send this file to an MVS system that has the Print Services Facility (PSF) support to print it. You may

need to use a different program product (such as the SENDFILE command, if you are a CMS/SP user) or a user written program.

In TSO, you can get 3800 Printing Subsystem Model 3 printed output in either of two ways. One way of getting 3800 Printing Subsystem Model 3 output in TSO is to use the SCRIPT command. If, for example, you issue the command

```
script docname dev(38ppn)
```

a file called **userid.docname.LIST38PP** is created. This file is the default. Your next step will be to use a system utility, such as IEBGENER, to put the file into the output queue for JES and PSF to print it.

The most direct way is to use the PRINT option of the SCRIPT command. If, for example, you issue the command

```
script docname dev(38ppn) print (1,c)
```

your output goes directly to SYSOUT as a JES allocated spool dataset. In this example, the suboptions of the PRINT option (1,c) represent the number of copies requested and the SYSOUT spool class for page mode printing, respectively.

There are also two ways of getting 3800 Printing Subsystem Model 3 printed output in DLF/MVS; in either method you will have to specify in your JCL statements the name of the font library and the page segment library to be used and that the printing is to be in page mode. If you want your output to go directly to a JES spool output queue for immediate printing, you could enter:

```
//SCRIPT JOB
//STEP1 EXEC PGM=DSMSPEXC,PARM='LIST'
//SYSPRINT DD SYSOUT=A
//DEV3800P DD SYSOUT=P
//FONTLIB   DD DISP=OLD,DSN=SYS1.FONT38PP
//PSEGLIB   DD DISP=OLD,DSN=SYS1.PSEG38PP
//DSMLIST   DD SYSOUT=A
//DSMINDIR  DD DSN=DSMLIB.DIRECTORY,DISP=SHR
//DSMINLIB  DD DSN=DSMLIB.SOURCE,DISP=SHR
//DSMUTMSG  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTDIM  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTTOC  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTWTF  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//SYSIN DD *
  AUTH 88/CITADEL
  SCRIPT 1314151 DSMIVC30 (PROF(DSMPROF3) CONT -
                           FILE(DEV3800P) DEV(38PPN))
/*
//
```

If, however, you want to first have a file created and then use a utility like IEBGENER to put the file into an output queue for JES and PSF to then print, simply change the line printed in boldface in our example above to read as follows:

```
//DEV3800P DD DSN=DEV3800.LIST38PP,DISP=MOD
```

In DLF/VSE, 3800 Printing Subsystem Model 3 formatted output is not supported.

Printing on the 3820 Page Printer

DCF enables you to print output on the 3820 Page Printer. The method you use to get this output depends on the operating environment in which you are working.

It is assumed that you have a font library containing a font library index, a page segment library, and the necessary 3820 Page Printer fonts on your CMS system. If you have these necessary requisites, you can format your output to a file by entering, for example:

```
script docname (dev(3820a)
```

Output created by this command is then placed into a file called **docname LIST3820 A**.

Because your job can not be printed directly on CMS, you will have to send this file to an MVS system that has the Print Services Facility (PSF) support to print it. You may need to use a different program product (such as the SENDFILE command, if you are a CMS/SP user) a or user written program.

In TSO, you can get 3820 Page Printer printed output in either of two ways. The most direct way is to use the PRINT option of the SCRIPT command. If, for example, you issue the command

```
script docname dev(3820a) print (1,c)
```

your output goes directly to SYSOUT as a JES allocated spool dataset. In this example, the suboptions of the PRINT option (1,c) represent the number of copies requested and the SYSOUT spool class for page mode printing, respectively.

Another method of getting 3820 Page Printer output in TSO is to use the SCRIPT command. If, for example, you issue the command

```
script docname dev(3820a)
```

a file called **userid.docname.LIST3820** is created. Your next step will be to use a system utility, such as IEBGENER, to put the file into a JES output queue for PSF to print.

There are also two ways of getting 3820 Page Printer printed output in DLF/MVS; in either method you will have to specify in your JCL statements the name of the font library and the page segment library to be used and that the printing is to be in page mode. If you want your output to go directly to a JES spool output queue for immediate printing, you could enter:

```
//SCRIPT JOB
//STEP1 EXEC PGM=DSMSPEXC,PARM='LIST'
//SYSPRINT DD SYSOUT=A
//DEV3820P DD SYSOUT=P
//FONTLIB   DD DISP=OLD,DSN=SYS1.FONT3820
//PSEGLIB   DD DISP=OLD,DSN=SYS1.PSEG3820
//DSMLIST   DD SYSOUT=A
//DSMINDIR  DD DSN=DSMLIB.DIRECTORY,DISP=SHR
//DSMINLIB  DD DSN=DSMLIB.SOURCE,DISP=SHR
//DSMUTMSG  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTDIM  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTTOC  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//DSMUTWTF  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//SYSIN DD *
  AUTH 88/CITADEL
  SCRIPT 1314151 DSMIVC30 (PROF(DSMPROF3) CONT -
                              FILE(DEV3820P) DEV(3820A))
/*
//
```

If, however, you want to first have a file created and then use a utility like IEBGENER to put the file into an output queue for JES and PSF to then print, simply change the line printed in boldface in our example above to read as follows:

```
//DEV3820P DD DSN=DEV3820.LIST3820,DISP=MOD
```

Printing on Page Printers in ATMS-III

In ATMS-III, you can get page printer output by following these steps:

1. Add a queue. The command syntax to do this for the 4250 printer, for example, is:

   ```
   $qadd; devpq; function; destid
   ```

   where: devpq is the queue name for the 4250 printer

   function is a required 4250 function keyword, such as 4250a. See the *ATMS-III Operation's Guide* for a list of supported function keywords.

   destid is the destination name for your output (look at the CICS DCT table - DSCNAME for the 4250 entry). Once the queue is added, it does not have to be added again.

2. Transmit a DCF input file to a queue. The command syntax to do this is:

   ```
   xfo; devpq; docname; ; [options]
   ```

   where devpq is the queue name for the 4250 printer

   docname is the name of the document to be transmitted.

   options are the SCRIPT command options.

   The following example shows some of the SCRIPT command options you might add to the ATMS command given earlier:

   ```
   xfo; devpq; docname; ; bind(1) co
   ```

If, after you specify this command, you receive a message indicating you have requested an invalid device type, then issue the command:

```
set; script; r3
```

and then reissue the **xfo** command. If the device type is correct, you should simply get a message indicating that the document was transmitted.

3. Process the queue. The command syntax to do this is:

```
$qstart; devpq
```

This command invokes the asynchronous task, which in turn calls DCF to format the document.

Note: Once the **Sqstart** command is issued, all documents that are already in the CPDS output dataset will be lost. You can transmit more than one document to the queue before starting and the output from all the documents will then be in the CPDS output dataset. Information about **Sqadd** and **Sqstart** is in the *ATMS-III Operation's Guide*. Information about **xfo** is in the *ATMS-III Terminal Operator's Guide*, SH20-2425.

To determine that processing has completed in the asynchronous task and that the data is in the dataset, use the **ql** command

```
ql; devpq; all
```

This gives a list of all entries in the queue. An entry of ACT means the entry has not been processed. An entry of PRC/xx means the entry has been processed. **xx** is the return code and it should be 0; if it is not, there was an error. See the *ATMS-III Terminal Operator's Guide* for a list of these codes.

Once the three previous steps have been done, you can get 4250 printer printed output by setting up a CDPF job with the DLBL/EXTENT or DD statement pointing to the dataset described in the DCT. Printing of documents in this dataset will end when BFUCDPF gets a return code greater than 8.

Migration and Conversion Considerations for Release 3

Because of differences among the printers you may be using, you should be aware of certain migration considerations when you format and print the same document on two different types of page printers or when you go from a line printer to a page printer. Many of these migration considerations are listed in the sections that follow. Refer to *Document Composition Facility: SCRIPT/VS Language Reference*, "Compatibility with Earlier Releases of SCRIPT".

3800 Printing Subsystem Model 3 to 3820 Page Printer

If you have formatted a document with DCF and printed it on the 3800 Printing Subsystem Model 3 and then want to format and print this same document on the 3820 Page Printer, the logical page size should be considered.

The primary migration consideration between the 3800 Printing Subsystem Model 3 and the 3820 Page Printer is the difference in logical page sizes. There are certain 3800 Printing Subsystem Model 3 logical pages that can not be printed on the 3820 Page Printer. These are:

13.5i x 11i (0-degree rotation)
11i x 8.5i (0-degree rotation)
13.5i x 8.5i (0-degree rotation)
11i x 8.5i (90-degree rotation)
13.5i x 11i (270-degree rotation).

For example, if you format a document for the 38PPW logical device which uses 13.5i x 11i paper; there is no equivalent paper size for a 3820 logical device. Therefore, certain page length and page width settings that work on large paper sizes for the 3800 Printing Subsystem Model 3 may "fall off" the logical page on the 3820 Page Printer.

3820 Page Printer to 3800 Printing Subsystem Model 3

If you have formatted a document with DCF and printed it on the 3820 Page Printer and then want to format and print this same document on the 3800 Printing Subsystem Model 3, you need to consider the following:

- Hardware requirements for the 3800 Printing Subsystem Model 3 prohibit the use of the top and bottom half inch of a page. If the bottom margin plus the reserved bottom half inch for the 3800 Printing Subsystem Model 3 is bigger than the bottom margin for the 3820 Page Printer, then the body length of your printed page will be smaller for the 3800 Printing Subsystem Model 3 than it is for the 3820 Page Printer.
- Because the 3820 Page Printer can print in the top and bottom half inch of the page, you can put data in those areas, for example, using page name areas. However, such formatting will not print on the 3800 Printing Subsystem Model 3 without errors because the top and bottom half inch of the page is reserved.
- The page origin is different on the 3820 Page Printer than it is on the 3800 Printing Subsystem Model 3. The physical origin and the logical origin of a page are the same for the 3820 Page Printer. The physical origin and the logical origin of a page are different for the 3800 Printing Subsystem Model 3 because of the half inch restricted area at the top of the page. As a result of this difference, text that fits on a 3820 Page Printer page may not fit on the page on the 3800 Printing Subsystem Model 3.
- A 180-degree rotation is not allowed on the 3800 Printing Subsystem Model 3 but is allowed on the 3820 Page Printer.
- The 3820 Page Printer has 16 possible combinations of character rotation and baseline direction by having only one copy of a base font. This is not true for the 3800 Printing Subsystem Model 3. Therefore, documents formatted for the 3820 Page Printer, that use different character rotations and/or baseline directions, may not print on the 3800 Printing Subsystem Model 3 because the fonts are not available on the 3800 Printing Subsystem Model 3.

4250 Printer to 3800 Printing Subsystem Model 3

If you have formatted a document with DCF and printed it on the 4250 printer and then want to format and print this same document on the 3800 Printing Subsystem Model 3, you need to consider the following:

- Hardware requirements for the 3800 Printing Subsystem Model 3 prohibit the use of the top and bottom half inch of a page. If the bottom margin plus the reserved bottom half inch for the 3800 Printing Subsystem Model 3 is bigger than the bottom margin for the 4250 printer, then the body length of your printed page will be smaller for the 3800 Printing Subsystem Model 3 than it is for the 4250 printer.

- Page length is variable for the 4250 printer. For the 3800 Printing Subsystem Model 3, the page length is limited by the physical size of the paper.
- The 4250 printer and the 3800 Printing Subsystem Model 3 have different typeface families. A .BF control word that is valid for the 4250 printer can be invalid for the 3800 Printing Subsystem Model 3.
- Line and page endings may be different because the widths of the equivalent characters in the fonts for the 4250 printer and the 3800 Printing Subsystem Model 3 are different.
- There is a limit to how many fonts you can use on a page with the 3800 Printing Subsystem Model 3, but there is no limit on the number of fonts that can be used on a page with the 4250 printer. Because of this difference, pages that print properly on the 4250 printer can cause errors and be unprintable on the 3800 Printing Subsystem Model 3.
- Images are incompatible.
- Negative intercharacter spacing is not available on the 3800 Printing Subsystem Model 3.
- Text that goes beyond the physical page for the 4250 printer truncates. When text goes beyond the physical page for the 3800 Printing Subsystem Model 3 the rest of the page may not be printed.
- The page origin is different on the 4250 printer than it is on the 3800 Printing Subsystem Model 3. The physical origin and the logical origin of a page are the same for the 4250 printer. The physical origin and the logical origin of a page are different for the 3800 Printing Subsystem Model 3 because of the half inch restricted area at the top of the page. As a result of this difference, text that fits on a 4250 printer page may not fit on the page on the 3800 Printing Subsystem Model 3.
- A character printed very close to the edge of a page on the 4250 printer may "fall off" the logical page when printed on the 3800 Printing Subsystem Model 3 because of the extra padding around each 3800 Printing Subsystem Model 3 character.

3800 Printing Subsystem Model 3 to 4250 Printer

If you have formatted a document with DCF and printed it on the 3800 Printing Subsystem Model 3 and then want to format and print this same document on the 4250 printer, you need to consider the following:

- Rotation is not supported by the 4250 printer. Therefore, if you try to print a rotated area on the 4250 printer, no rotation is done, and the output will be different.
- Some characters are included in the 3800 Printing Subsystem Model 3 fonts that are not in the 4250 printer fonts, such as: superscripts, long em dash, and logical not sign.
- The 4250 printer and the 3800 Printing Subsystem Model 3 have different typeface families. A .BF control word that is valid for the 4250 printer can be invalid for the 3800 Printing Subsystem Model 3.
- Line and page endings may be different because the widths of the equivalent characters in the fonts for the 4250 printer and the 3800 Printing Subsystem Model 3 are different.
- Images are incompatible.

4250 Printer to 3820 Page Printer

If you have formatted a document with DCF and printed it on the 4250 printer and then want to format and print this same document on the 3820 Page Printer, you need to consider the following:

- Page length is variable for the 4250 printer. For the 3820 Page Printer, the page length is limited by the physical size of the paper.
- The 4250 printer and the 3820 Page Printer have different typeface families. A .BF control word that is valid for the 4250 printer can be invalid for the 3820 Page Printer.
- Line and page endings may be different because the widths of the equivalent characters in the fonts for the 4250 printer and the 3820 Page Printer are different.
- There is a limit to how many fonts you can use on a page with the 3820 Page Printer, but there is no limit on the number of fonts that can be used on a page with the 4250 printer. Because of this difference, pages that print properly on the 4250 printer can cause errors and be unprintable on the 3820 Page Printer.
- Images are incompatible.
- Negative intercharacter spacing is not available on the 3820 Page Printer.

3820 Page Printer to 4250 Printer

If you have formatted a document with DCF and printed it on the 3820 Page Printer and then want to format and print this same document on the 4250 printer, you need to consider the following:

- Rotation is not supported by the 4250 printer. Therefore, if you try to print a rotated area on the 4250 printer, no rotation is done, and the output will be different.
- Some characters are included in the 3820 Page Printer fonts that are not in the 4250 printer fonts, such as: superscripts, long em dash, and logical not sign.
- The 4250 printer and the 3820 Page Printer have different typeface families. A .BF control word that is valid for the 4250 printer can be invalid for the 3820 Page Printer.
- Line and page endings may be different because the widths of the equivalent characters in the fonts for the 4250 printer and the 3820 Page Printer are different.
- Images are incompatible.

Other Page Printing Considerations

It is possible to create documents that can be formatted and printed with acceptable results on a line printer but not a page printer. The following should be considered:

- **Data off the page.** When data runs off a page, line printers will truncate the data without issuing a message. Page printers will issue an error message, and either mark the printed output at the point where truncation occurred or leave the rest of the page blank. This condition may occur if:
  - You are using format off mode (.FO OFF control word)
  - You are creating examples (:XMP tag)

- You are using an input line that is too long
- You are using a large font size.

- **Undefined characters.** When a line printer encounters an undefined character, one of the following occurs:
  - A blank is printed
  - The previous character in a print buffer is printed
  - You get a data check.

  For a page printer, one or more of the following occurs:
  - A special character symbol is printed
  - A DCF error message is issued
  - A PSF error message is issued
  - You get a data check.

  The above will occur whenever you use a codepoint (via character translation with the .TI, .TR, or the .TU control words or via keyboard input) which is not defined in the coded font that you are using.

  If the font you are using has a specified default character to use when undefined characters are encountered, you will get a bold cross (for the 4250) or a zero with a diagonal through it (for the 3800-3) printed. If the font does not have a specified default character, an error occurs and printing stops.

  Unlike line printing, page printers do not print a blank when an undefined character is encountered. For page printers, use the .IS control word to leave blank a space in which to draw or paste in characters.

  To print the character you desire, one of the following may accomplish your results.
  - Change the character codepoint
  - Change to a different font
  - Change to a different code page
  - Use a substitute character.

Check your font catalog to determine the codepoint of the character you desire.

- **Descenders.** A bottom print line that is printable on a line printer may not be printable on a page printer. In line printing, the bottom of the print line is at the bottom of all the characters — the descenders sit at the print line. But in page printing, the baseline of the font characters is at the bottom of the print line and, therefore, the descenders of the characters extend below the print line. When the bottom print line is too close to the logical bottom of the page, the descenders may extend off of the page and one of the following can happen:
  - An error message is printed
  - The entire line or remainder of the page will not print
  - A special character appears where the error occurred.

  This condition may occur if you changed the default bottom margin to zero or if you are using a font with very large descenders and the print line is positioned too close to the logical bottom of the page.

Chapter 3. Marking Up Documents with SCRIPT/VS

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
- 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.[13] 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.

Language Syntax

When you use a text formatting language like SCRIPT/VS, certain conventions of that language, called its syntax, must be observed. The correct syntax for SCRIPT/VS control words and for SCRIPT/VS macro and symbol processing is given below.

Control Word Syntax

All control words have two-character names. A control word is identified by a period (.) in the first position of an input line, followed by the two-character name.[14] 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
```

---

[13] See the *Document Composition Facility: Generalized Markup Language Starter Set Reference* for a description of how to mark up a document with GML tags.

[14] SCRIPT/VS control words are presented throughout this book. For a complete description of SCRIPT/VS control words see the *Document Composition Facility: SCRIPT/VS Language Reference.*

The blank separating the control word name from the first parameter is usually optional;[15] 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. Conversely, if you incorrectly enter a macro name, SCRIPT/VS may interpret this invalid macro as a control word. If macro substitution is off, any macro name may be interpreted as a control word.

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 *control word separator*.[16] 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, you can quickly see the sequence and context of one control word within the group.

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 a separator character 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 the .FO to be recognized as a control word.

---

[15] The blank separating the control word name from the first parameter is *not* optional with .LI OFF, .DM OFF, .CS n OFF, and .WF OFF.

[16] The character to be used as the control word separator may be changed with the .DC CW [Define Character] control word.

The control word separator character may also be used to place a control word within a line of text. For example,

```
an ;.us on;underscored;.us off; word.
```

results in:

an underscored word.

SCRIPT/VS scans every text line for the control word separator character. If a separator character is found which 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 Control Word Modifier

The SCRIPT/VS control word processor recognizes a single quotation mark (') after the period as a control word modifier. Most control words[17] can be entered with the modifier (') as shown in the following example:

```
.'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 and macro substitution is on, 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. The input line

```
.'ce centered line; one line.
```

results in:

```
                    centered line; one line.
```

[17] With the exception of .LI OFF, .DM OFF, .PX OFF, and .WF OFF.

Macro Syntax

A SCRIPT/VS macro name can look much like a control word but its syntax is slightly different. A macro name can be up to ten characters long and these characters must be chosen from the valid character set: A-Z, 0-9, $, # or @.

Symbol Syntax

A SCRIPT/VS symbol is preceded with the ampersand sign (&). Symbols can also be up to ten characters long and these characters must be chosen from the valid character set: A-Z, 0-9, $, #, or @.

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 Position 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 position one. Occasionally, you may want to enter lines that begin with blank characters or tabs. 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 Position One

When SCRIPT/VS processes an input line, data that follows a period in position one 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 position one (even though you intended it to be text), SCRIPT/VS processes it as a control word. In such a case, the results might be undesirable.

You can use the .LI [Literal] control word to have a 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. For example, the underscoring and capitalization control words are good examples of control words that do not cause breaks:

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

results in:

```
This SENTENCE has several
control WORDS IN IT, and its
text is concatenated.
```

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 others, you can include 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.
- Documenting options: If you use a special set of SCRIPT command options to format a document, you can include notes within the file to list the options.

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, ".*" to enter 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, the entry

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

causes SCRIPT/VS to ignore this entire input line. Therefore, file DOC3 will not be imbedded.

Valid Space Unit Notation

Many SCRIPT/VS control words accept parameters that specify vertical or horizontal dimensions or distances. As Figure 3 on page 44 illustrates, these dimensions may be expressed in any of several different space units:

Centimeter One-hundredth of a meter. There are 0.39 inches in one centimeter.

Cicero A standard measurement in the Didot Point System, used in most countries except Great Britain and the United States. The Cicero is 4.511 millimeters (0.1776 inches), and there are twelve Didot points in one Cicero. Ciceros can be specified in tenths of units (for example, 1.5c = 1.5 Ciceros).

Device Unit

Horizontal An integral number of horizontal device units. The size of a horizontal device unit depends upon the device and varies from 2.540 millimeters (1/10 inch) for the 1403 to 0.0423 millimeters (1/600 inch) for the 4250 printer and 0.1058 millimeters (1/240 inch) for the 3800 Printing Subsystem Model 3 and the 3820 Page Printer.[18]

Vertical An integral number of vertical device units. The size of a vertical device unit depends upon the device and varies from 4.233 millimeters (1/6 inch) for a 2741 to 0.0423 millimeters (1/600 inch) for the 4250 printer and 0.1058 millimeters (1/240 inch) for the 3800 Printing Subsystem Model 3 and the 3820 Page Printer.[18]

Em-space A decimal number of horizontal em-widths. The size of an em-width depends upon the current font.

Em-height A decimal number of vertical em-heights. The size of an em-height depends upon the current font.

Inch One-twelth of a foot (25.4 millimeters).

Millimeter One-thousandth of a meter. There are 10 millimeters in one centimeter. (25.4 millimeters = 1 inch).

Pica A standard printer's measurement in Great Britain and the United States. A pica is 4.224 millimeters (0.1663 inches). There are twelve points in a pica and 72 points in an inch.[19]. Picas can be specified in tenths of units (for example, 1.5p = 1.5 picas).

Unqualified space units are defined in the following ways:

- Horizontal space units (such as .IN 5) are defined in one of the following ways in the order given:
  - The size of a figure space in the initial font
  - The size of an en in the initial font
  - One-half the size of an em in the initial font
- Vertical space units (such as .SP 5) are defined as:
  - The linespacing value of the current font

Also note that fractional unqualified space units and ems, such as 1.5, are now supported. In order to avoid a problem with the symbol delimiter, which is a period (.), fractional units may be specified with a comma instead of a period. For example, you can specify 1,5 instead of 1.5.

---

[18] Because of the wide variation in magnitude of device units between devices, the use of these space unit designations can bind individual documents to particular devices. To maintain device independence, formatting using device units should always be done in conjunction with calculations using device units symbol attributes. It is not always possible to satisfy space requests exactly on all devices. In this case, the nearest available amount is used.

[19] In SCRIPT/VS, 72 points equals exactly one inch rather than .996 or 1.008 inches.

Space Unit	Specified As	Examples
Centimeter	aCM	4.25cm 2,54cm 15cm
Character (Horizontal)	a	5 12.5 1,33
Cicero	nCp	c12 (12 didot points) 2c3 (2 Ciceros and 3 points) c1.5 (1.5 didot points)
Device Unit (Horizontal)	nDH	10dh 600dh
Device Unit (Vertical)	nDV	10dv 600dv
Em-space (Horizontal)	aMH -or- aM	6mh 6m .33mh .33m
Em-space (Vertical)	aMV	1mv .5mv
Inch	aI	3.5i 6,5i .75i
Line (Vertical)	a	2 3.5 1,75
Millimeter	aMM	12.7mm 25,4mm 100mm
Pica	nPp	p6 (6 points) 3p2 (3 picas and 2 points) p1.5 (1.5 points)

Where:

a is a number of centimeters, characters, ems, inches, lines, or millimeters. The number may be fractional, with up to two decimal positions, and either a period (.) or comma (,) can be used to separate the integral and fractional portions of the number.

n is a number of whole ciceros, picas, or device units.

p is a number of points. (There are twelve points in a cicero or pica, and 72 points in an inch.)

Figure 3. 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: Character spaces are equal in size to the figure space of the default (or initial) font. Line spaces are equal in size to the linespacing of the current font.

Text

Because SCRIPT/VS formats your document based on default settings appropriate for the logical device you have specified, you need to be aware of certain implicit markup. In this case, implicit markup refers to such horizontal spacing mechanisms as spaces, tabs, and backspaces.

Implicit Markup

In a SCRIPT/VS context, spaces, tabs (see "Using Tabs" on page 84 for a full discussion of tabs), and backspaces function as word delimiters. Their hexadecimal representations are as follows:

- Spaces - hexadecimal 40
- Tabs - hexadecimal 05
- Backspaces - hexadecimal 16
- Nulls - hexadecimal 00

Continuation and the Continuation Character

Ordinarily SCRIPT/VS appends a word space to the last word on a text input line. However, if the continuation character is the last character on a text input line, it is removed and the word 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
light+
.pf
ed characters.
```

will produce this output:

```
A few highlighted characters.
```

If a formatter control that causes a break follows the continued word, continuation is cancelled for that line. The control words that cause breaks are listed in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Initially, there is no continuation character; it must be explicitly set before it can be used.

Even if a previous line does not end with a continuation character, you can use the .CT [Continued Text] control word to cause a line to be treated as a continuation of a previous text line. For example, if you specified

```
This input line a
.ct nd this input line should be one line.
```

then the two input lines will be joined as one:

```
This input line and this input line should be one line.
```

If the .CT control word is given without a line of text, then nothing will be continued and any continuation that may be in effect from a continuation character on the previous text line is cancelled.

Chapter 4. Combining SCRIPT/VS Input 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 without returning to the appending file
- .SI [Segment Include], which identifies a segment to be included in a column
- .WF [Write To File], which causes lines of text or control words to be written to the output file DSMUTWTF.

Imbedding and Appending Files

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 file OUTER 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 unless respecified by control words in TESTER. When SCRIPT/VS reaches the end of the file TESTER, 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. 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. 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 4 on page 49.

You can pass values to the imbedded or appended file, so the file can be customized each time it is called.

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 Primary Input File" on page 18.
- If you enclose the file-id in parentheses, SCRIPT/VS uses the file-id, which in this case can be more than eight characters long, as the real file or data set name.
- If no .DD control word has been processed for file-id, SCRIPT/VS uses the 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 or the filetype specified with the SEARCH option of the SCRIPT command.
  - In TSO, SCRIPT/VS assumes that the file-id is a member of the partitioned data set (PDS) 'userid.text' and imbeds this file if it exists.
  - In ATMS-III, SCRIPT/VS assumes that the document is in the invoking operator's permanent storage.

In CMS, you should use the .DD [Define Data File-id] control word when:

- The imbedded filename on the .IM control word is different from the actual CMS filename.
- The filetype is other than SCRIPT and was not specified with the SEARCH option of the SCRIPT command.
- A specific filemode that is not the first in the CMS search sequence is to be used.

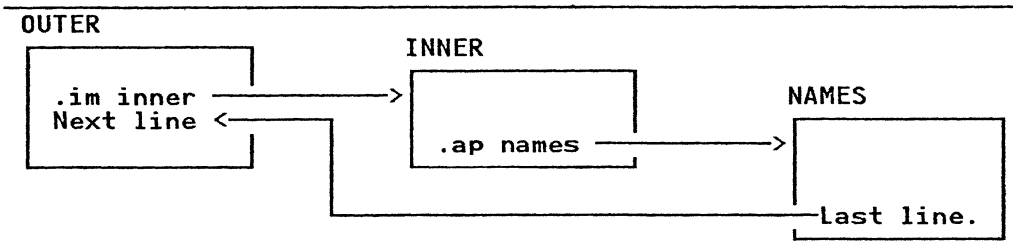


Figure 4. Imbedding and Appending SCRIPT/VS Files

In TSO, you must 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.
- 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 the invoking operator's permanent storage or the permanent storage of another operator whose number has not been specified in the SEARCH command option.
- The document has been protected by a password by the other operator.

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 in a library other than the ones listed with SCRIPT command options.

The format and use of the .DD control word in the definition of files are explained in full in the description of the .DD [Define Data File-id] control word found in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Indicating the End of a File

The .EF [End of File] control word causes a file to end and this can be useful when you are imbedding 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.

Alternatively, if you specify:

```
.ef close
```

the next time the file is imbedded, SCRIPT/VS begins reading at the beginning of the file rather than where you left off.

Note: If the .EF [End of File] control word is included in the profile specified with the SCRIPT command, the contents of the file preceding the .EF [End of File] control word will be processed before the main document. The remainder of the file, after the .EF

control word, is referred to as the epifile and is automatically processed after the main document. This indicates the end of processing.

Two control words, .QU [Quit] and .QQ [Quick Quit], cause SCRIPT/VS to stop processing entirely, regardless of whether the current file is an imbed file or not. When you use the .QU [Quit] control word, processing stops after SCRIPT/VS prints the remainder of the current page (and any running footings in effect) and after SCRIPT/VS closes all open files. In contrast, the .QQ [Quick Quit] control word causes processing to stop immediately 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 stop 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 can be displayed at your terminal for you to note and correct later. However, processing stops before the second pass occurs, and no formatted output will be displayed.

Master Files

Using imbeds in SCRIPT/VS has several advantages.

- 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 special formatting controls 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 files where appropriate. Therefore, you do not need to keep duplicate copies of the same information.
- Although there may be a limit to the number of records that can be contained in a single disk file, within the limits of your virtual storage there is no restriction on the number of files that SCRIPT/VS can process.
- Many different people can work on pieces of the same document simultaneously.

Figure 5 on page 51 illustrates a typical master file structure.

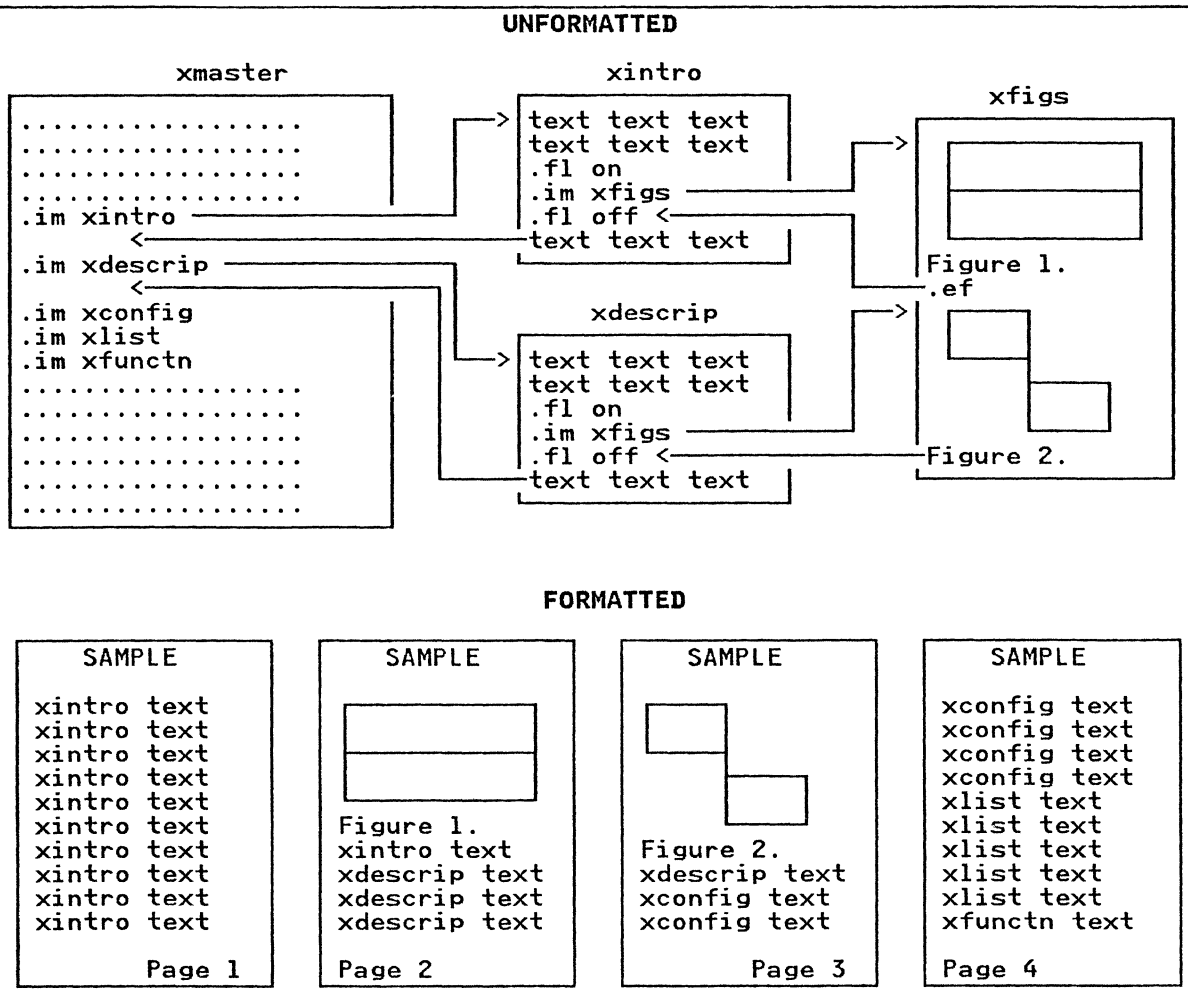


Figure 5. Master File Structure

SCRIPT/VS System Generated Files

SCRIPT/VS has a number of utility files that are generated by the system when it encounters certain control words, or command options, or both. The user has the option of defining or redefining these files using the .DD [Define Data File-id] control word. These files and a brief description of them are listed below.

DSMTERMI The file from which terminal input is read when SCRIPT/VS encounters the .TE [Terminal Input] or .RV [Read Variable] control words. By default this is the terminal.

DSMTERMO The file into which terminal output is written when the .TY [Type on Terminal] control word is specified, the TERM option of the SCRIPT command is specified, or messages are given when the MESSAGE (DELAY) option of the SCRIPT command is not specified. By default this is the terminal.

DSMUTCTF The file into which STAIRS/VS CTF output is written when SCRIPT/VS encounters the CTF option of the SCRIPT command.

DSMUTMSG The file into which messages are written when the MESSAGE(DELAY) option of the SCRIPT command is specified.

DSMUTTOC The file into which the table of contents entries are written when SCRIPT/VS encounters the .PT [Put Table of Contents] or .H0 - .H6 [Head Level 0 - 6] control words.

DSMUTWTF The file into which input lines may be placed dynamically when SCRIPT/VS encounters the .WF [Write To File] control word.

Writing to an Output File

The .WF [Write To File] control word allows you to put input lines into a file dynamically[20]. For example, you can collect figure captions for a figure list in one file and index entries in another.

While you can have several .WF files, 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 input line into the file with:

```
.wf contents of the input line
```

- You can insert a specific number of input lines into the file with:

```
.wf 5
.in 3m
.ce 3
These are the
lines to go
into DSMUTWTF.
```

  Input lines that are written to the file will be processed for symbol substitution and GML tag processing unless these functions have been specifically inhibited.

- You can also insert a number of input lines into the file with:

```
.wf on
  .
  .
  .
Many input lines
  .
  .
  .
.wf off
```

[20] In ATMS-III, the .WF control word can only be used to write to a document in CICS/VS auxiliary storage. It cannot be used to write to a document in either working or permanent storage.

Note: The .WF OFF control word must appear on an input line by itself exactly as it is shown here.

If you want to use .WF ON in a GML start tag APF, you can use the .WF TAG form as shown in the following example:

```
.aa tag tag etag
.gs tag on
.ms on
.dm tag on
.wf tag
.dm off
.dm etag /.*
```

Then when you specify :TAG. all lines will be written to DSMUTWTF until the end tag (:ETAG.) is encountered.

The write to file request is automatically ended before the end APF is invoked. GML scanning is off during this form of write to file processing until the end tag is found.

You can later imbed 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.

To add lines to the end of the CMS file PART6 ZORCH Z1:

```
.dd dsmutwtf part6 zorch z1
.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 dsmutwtf
```

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.

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 sections of a document because each group of lines from the imbedded file can contain the specific information for a particular section 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

.tp 3 21
.cs 2 on
.cs 1 ignore
.sp 2
.fo off
.bx 1 19 36
.se bxoff =
.cs 2 ignore
.cs 2 off
.im tablsym
&$TAB.&state.&$TAB.&capital
.bx &bxoff
.cs 1 on
.fo on
.cs 2 include
.sp 2
.ef
.cs 1 off
.ap table
```

```
File: TABLSYM

.se state   'STATE
.se capital 'CAPITAL
.ef
.se state   'Alabama
.se capital 'Montgomery
.ef
.se state   'Alaska
.se capital 'Juneau
.ef
.se state   'Arizona
.se capital 'Phoenix
.ef
.se state   'Arkansas
.se capital 'Little Rock
.ef
.se state   'California
.se capital 'Sacramento
.se bxoff = off
.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

Imbedding Segments in Your Documents

You can use the .SI [Segment Include] control word to identify a segment to be included in a column when a document is printed on a page printer. A segment can also be used to reserve space for artwork when a document is printed on a line device. A segment can be composed of text and images and it can be imbedded either directly in your document or within a *named* area. See "Chapter 11. Placing Text in Named Areas" on page 133 for more details on *named* areas.

You must specify the external name of the file that contains the segment. This name depends on the system in which you are operating:

- CMS - the name is that of a CMS file, whose filetype and filemode were identified with the SEGLIB option of the SCRIPT command.
- TSO - the name is that of a member in the segment library.
- ATMS-III - the name is that of a member in the segment library, which was identified with the SEGLIB option of the SCRIPT command.
- MVS - the name is that of a member in the segment library, which was identified with the SEGLIB option of the SCRIPT command.
- VSE - segments are not supported in VSE.

The existence of the segment identified with the .SI control word is verified when the document is formatted unless the NOSEGLIB option of the SCRIPT command is specified.

When your segment is printed on a page printer, it will be aligned according to the current text formatting settings.

Specifying Segment Width and Depth

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

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

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

Specifying Inline Page Segments

If you want the actual contents of the page segment and not just its name included in the output data stream, use the INLINE parameter of the .SI [Segment Include] control word.

If the requested page segment is not found and the INLINE parameter was not specified, SCRIPT/VS will still put the page segment name in the output data stream. If the requested page segment is found and the INLINE parameter was specified, SCRIPT/VS will not put the page segment name in the output data stream. In either case, a message is issued.

Using the &SW' and &SD' Symbol Attributes

The &SW' and &SD' symbol attributes can be used to determine the width and depth of a segment. These symbol attributes can be useful when you are trying to dynamically place a segment on the page.

If a segment named **seg1** exists, for example, the value of &SW'seg1 will be the width of **seg1** specified in unqualified horizontal space units. Similarly, &SD'seg1 will return the depth of **seg1** in unqualified vertical space units.

When formatting for a line printer or when NOSEGLIB has been specified on the SCRIPT command, the value returned by both symbol attributes will be 0.

To obtain the width or depth of a segment in pels, thereby avoiding rounding, use &DH'&SW'seg1 and &DV'&SD'seg1.

The Segment Library

Keep in mind that when you process your document, SCRIPT/VS searches either the default segment library or a segment library you have specified with the SEGLIB option of the SCRIPT command for any segments you request with the .SI control word. SCRIPT/VS automatically searches the default library for requested segments but you must specify the SEGLIB option of the SCRIPT command if the segment you request is in a segment library you have created. Note also that SCRIPT/VS searches only one segment library, either the default library or the one you created, but not both.

The defaults for the 4250 printer are:

- In CMS, SEGLIB(PSEG4250)
- In TSO, SEGLIB(SYS1.PSEG4250)
- In ATMS-III, NOSEGLIB
- In batch MVS, SEGLIB(PSEG4250)
- In batch VSE, segments are not supported.

The defaults for the 3800 Printing Subsystem Model 3 are:

- In CMS, SEGLIB(PSEG38PP)
- In TSO, SEGLIB(SYS1.PSEG38PP)
- In ATMS-III, NOSEGLIB
- In batch MVS, SEGLIB(PSEG38PP)
- In batch VSE, segments are not supported.

The defaults for the 3820 Page Printer are:

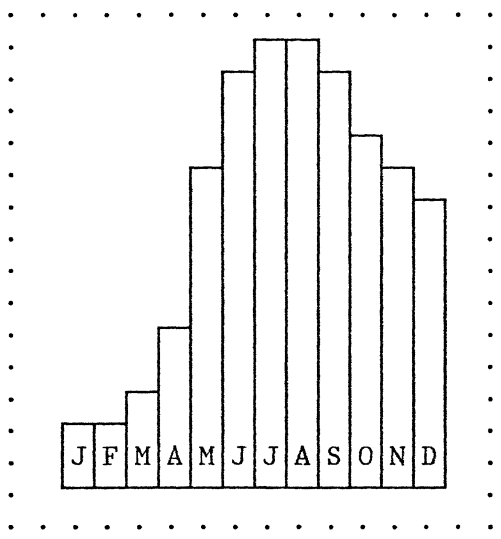
- In CMS, SEGLIB(PSEG3820)
- In TSO, SEGLIB(SYS1.PSEG3820)
- In ATMS-III, NOSEGLIB
- In batch MVS, SEGLIB(PSEG3820)
- In batch VSE, segments are not supported.

If the segment you request is not in the library that SCRIPT/VS searches or there is no segment library, then processing stops (unless you have specified the CONTINUE option of the SCRIPT command) and an error message is issued.

If you know a segment has not yet been created or there is no segment library, you can specify the NOSEGLIB option of the SCRIPT command. SCRIPT/VS will not search for a segment library and no error message will be issued. In this case, if no depth or width was specified, no space is reserved for the segment in the formatted output.

The Composed Document Printing Facility (CDPF), which is used to print SCRIPT/VS output on the 4250 printer, does not allow the use of a segment of the same name more than once on the same page. SCRIPT/VS, however, has no such restriction for any printer.

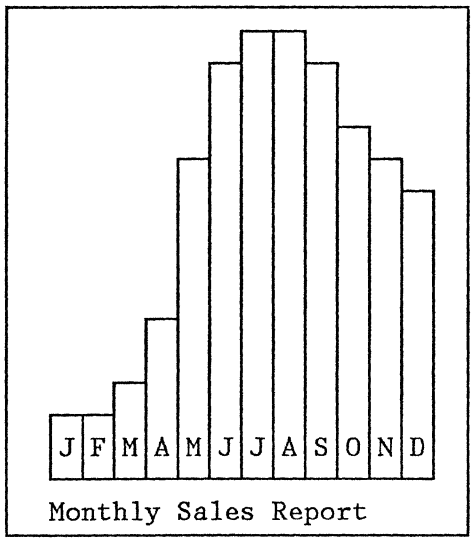
Segments can be included as part of a figure. For example, assume the following segment named BARCHART has been created and that it exists in the segment library.



This segment can then be included as the body of the following figure:

```
:fig frame=box place=inline width=column.
.si barchart
:figdesc: Monthly Sales Report
:efig.
```

When the document is printed on a page printer, the segment will be combined with the figure frame and figure caption to produce the following figure:



Chapter 5. Communicating with SCRIPT/VS

SCRIPT/VS Messages and Severity Levels

When certain error conditions are encountered, SCRIPT/VS issues messages in the form of a 10-character message identifier that includes a one-character severity level code.

The message identifier is in the form DSMmmmnnnx

where

DSM identifies the Document Composition Facility

mmm is a three-character identifier of the program module that caused the message to be sent

nnn is a three-digit message number

x is a severity level

The severity levels associated with these SCRIPT/VS messages are as follows:

R reply required

I informational

W warning

E error

S severe error

T terminal error

Using a SCRIPT/VS Command Option to Control Message Printing

You can use the MESSAGE option of the SCRIPT command to control message printing. You can specify:

- When messages are printed
- Whether the message number is to be included
- How the line causing the error was imbedded.

The MESSAGE option also 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 line 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 can specify two or all three parameters, separated by blanks. Each of the options can be abbreviated as a single letter.

DELAY 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.

ID causes SCRIPT/VS to include the error message identifier along with the error message.

TRACE 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.

If CONTINUE is specified, SCRIPT/VS continues processing unless a severe (S) or terminal (T) error is encountered. Severe and terminal errors cause SCRIPT/VS to stop processing even if CONTINUE was specified.

The NOCONT option stops processing after SCRIPT/VS encounters an error condition of severity level E (error), S, or T and issues an error message.

Note: CMS truncates messages that are more than 130 characters long. Truncation can also occur in the batch or TSO environments when the messages exceed the record length of the message data set (DSMUTMSG).

For a description of SCRIPT/VS error messages, see the publication *Document Composition Facility Messages.*

The .MG [Message] Control Word

You can use the .MG [Message] control word to write out a message and to provide diagnostic messages from macros.

Messages generated by the .MG control word can affect the return code from SCRIPT/VS and can cause SCRIPT/VS processing to stop. Type S (severe) or type T (terminating) messages always stop processing, and type E (error) messages stop processing if the CONTINUE option of the SCRIPT command is not in effect. If, for example, you specified

```
.mg /T/London's Burning/
```

then processing would stop and the message - London's Burning - would be printed.

When a message is displayed, a prefix of "+ + +" appears before the identifier or text to indicate the message was generated by the .MG control word. If no data is given with the .MG control word, it is ignored.

The following is an example of the .MG [Message] control word as you might use it:

```
The control word:

 .mg /001e/This is a message./

is displayed as:

 +++001E This is a message.

if MESSAGE(ID) is in effect, or:

 +++ This is a message.

if MESSAGE(ID) is not in effect.
```

Interactive SCRIPT

You can interact with SCRIPT/VS while it is processing your document in order to communicate with VM/SP and TSO or to trace SCRIPT/VS processing.

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. Several control words allow you to interact with SCRIPT/VS as your document is being formatted.

If you are in CMS (only), you can use the .RD [Read Terminal] control word when you want to stop a typewriter terminal during SCRIPT/VS output to type in some text. SCRIPT/VS does not process this text in any way, but resumes its output when you signal ATTENTION.

The .RD [Read Terminal] control word is meaningful only when the formatted output is actually being typed at your terminal in CMS. The text typed is not processed by SCRIPT/VS, but appears in the output exactly as it was typed. When using the .RD [Read Terminal] control word under CMS, specify

```
cp term attn off
```

before invoking SCRIPT/VS to suppress CP's normal attention acknowledgment. Remember, the .RD [Read Terminal] control word is recognized only in the CMS environment.

You can use the .RV [Read Variable] control word to set symbols to read values from the terminal during SCRIPT/VS processing. When the .RV [Read Variable] 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 [Read Variable] control word.

Any expression that would be allowable as the value in a .SE [Set Symbol] control word is allowable here. (See the *Document Composition Facility: SCRIPT/VS Language Reference* for a complete description of the .SE [Set Symbol] control word and its syntax rules.) If no name is given on the .RV [Read Variable] control word, it is ignored and no line is read from the terminal.

The .RV [Read Variable] control word will be ignored in batch environments unless the file DSMTERMI can be read.

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 .TE [Terminal Input] control word accepts several operands. If, in the input file, you specify

```
.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 [Terminal Input] control word. For example,

```
.te 4
```

causes SCRIPT/VS to read four lines from the terminal.

You can also stop 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 .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 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 (/)[21] 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.

---

[21] 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 21. Processing Symbols" on page 223.

Communicating with VM/SP

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, use the .SY [System Command] control word. For example,

```
.sy cp spool printer class s
```

Note: Because the SYOFF SCRIPT command option (which is the default) disables the .SY control word, you must specify the SYON command option (which enables the .SY control word) when you process your document. SYOFF and SYON are applicable only in the CMS and TSO environments.

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 process a command during SCRIPT/VS processing,[22] 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 stops processing. If the return code is zero, execution continues with the next input line following the .IF control word 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.

Note: Because the SYOFF SCRIPT command option (which is the default) disables the .SY control word, you must specify the SYON command option (which enables the .SY

---

[22] Caution must be exercised when processing commands in this way because they might cause SCRIPT/VS to prematurely end processing as a result of the way in which these commands use and/or manage storage.

control word) when you process your document. SYOFF and SYON are applicable only in the CMS and TSO environments.

The .SY [System Command] control word, for example, 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, if you have followed correct TSO naming conventions and if you have properly allocated another file, specify

```
script infile file('outfile') ...
```

Then if you enter

```
.sy edit 'outfile' old
```

this causes the output file to be displayed.

For more details, see "TSO Naming Conventions" on page 18 and the discussion of the FILE option of the SCRIPT command in the *Document Composition Facility: SCRIPT/VS Language Reference.*

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:

```
*¢* [file-id] [nn]  x  <current 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

file-id identifies the origin 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.

(FNLEAD) 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 that will be used in scanning the current GML tag.

(SCAN) The current source line displays the text that 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, control word tracing, symbol substitution tracing, and macro substitution tracing (as specified with .IT ALL), are being traced:

- 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 *before* it is executed. SCRIPT/VS then executes the *modified* 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 that 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 symbol has been replaced with its value. 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 (*S*).
  - 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 them with the .IT control word:

```
.it ctl .if .el .th
```

  When .IT ON is specified, all occurrences of these control words will be traced.

  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, specify

```
.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 later tracing, or if you want to turn off all input tracing, specify:

```
.it off
```

  When you want to resume tracing the control words currently in the table, specify:

```
.it on
```

  To add more control words to the control word table, issue another .IT CTL command:

```
.it ctl .if .el
```

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 the symbol or macro is then displayed. The SNAP parameter does not affect other parameters of the .IT control word and can be specified even when input tracing is turned off.

Part 2. Document Composition Facilities of SCRIPT/VS

In this section of the book the many document composition facilities provided by SCRIPT/VS are discussed.

Included in this section are the following chapters:

- Chapter 6 - Composing Lines
- Chapter 7 - Hyphenating and Horizontally Justifying Text
- Chapter 8 - Creating Vertical Space
- Chapter 9 - Vertically Justifying Text
- Chapter 10 - Establishing Page Layout
- Chapter 11 - Placing Text in Named Areas
- Chapter 12 - Composing Multiple-Column Pages
- Chapter 13 - Creating Head Levels and Table of Contents
- Chapter 14 - Creating Rules and Boxes
- Chapter 15 - Selecting Fonts
- Chapter 16 - Keeping Blocks of Text Together
- Chapter 17 - Creating Footnotes
- Chapter 18 - Translating Characters
- Chapter 19 - Creating an Index.

Chapter 6. Composing Lines

SCRIPT/VS Text Formatting

SCRIPT/VS can format input text to build output lines. This formatting consists of two processes that 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
- *Justification*: distributing space between words to align the right edges of output lines (right-justified).

Format Mode

Most documents that you compose require some kind of formatting. With format mode on, lines that are entered in a SCRIPT/VS file as:[23]

```
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:

```
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 output 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 output 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 concatenated and justified as in the above example.

---

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

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,[24] or you can use the .FO [Format Mode] or the .NF [No Formatting] control words to suspend SCRIPT/VS justification and concatenation:

```
.fo off                          .nf on
Boston                           Boston
Chicago          -or-            Chicago
New York                         New York
Providence                       Providence
```

To restore normal formatting, use the control word:

```
.fo on          -or-          .nf off
```

Because ON is the default for .FO [Format Mode], you can also specify:

```
.fo
```

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

Centered Text

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
```

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,

---

[24] The .BR [Break] control word is discussed later in this chapter under "Breaks" on page 77.

```
.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 center text using the CENTER parameter of both the .FO and .NF control words.

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 text is formatted using the .NF CENTER control word.

```
      Up and spoke an elderly knight,
    Who sat at the king's right knee:
"Sir Patrick Spence is the best sailor
        That sails upon the sea."
```

The difference between .FO CENTER and .NF CENTER is that .FO CENTER will cause input lines to be concatenated while .NF CENTER will not.

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 it, 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
```

If you want your text to be left-aligned in the column but not concatenated, you can use the .NF LEFT control word to produce the following output.

The king sits in Dumferling town,
Drinking blood-red wine:
"O where will I get a good sailor
To sail this ship of mine?"

Ragged Left

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
```

You can also use the ON and OFF parameters with the .RI [Right Adjust] control word. 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.

You can use the .FO RIGHT control word if you want to format input lines that are concatenated and that produce *ragged left* output (unjustified output lines aligned with the right margin).

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

> 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).

If you want your text to be right-aligned in the column but not concatenated, you can use the .NF RIGHT control word to produce the following output.

```
                                    The king has written a braid letter,
                                           And signed it with his hand,
                                     And sent it to Sir Patrick Spence,
                                           Who was walking on the sand.
```

Alternate Formats

You can use the .FO INSIDE control word if you want input lines to be concatenated and aligned so that resulting output lines are against the *inside* margin of the column - towards the presumed binding edge of the duplexed page. This is equivalent to .FO LEFT for odd pages and .FO RIGHT for even pages. The following text was formatted using the .FO INSIDE control word.

```
The first line that Sir Patrick read, A loud laugh laughed
he; The next line that Sir Patrick read, Caused the tears to
flow full free.
```

You can use the .FO OUTSIDE control word if you want input lines to be concatenated and aligned so that resulting output lines are against the *outside* margin of the column - away from the presumed binding edge of the duplexed page. This is equivalent to .FO RIGHT for odd pages and .FO LEFT for even pages. The following text was formatted using the .FO OUTSIDE control word.

"Make haste, make haste, my merry men so fine Our guide ship sails in the morn." "O say 'tis not so, my captain great, For I fear a deadly storm."

If you want your text to be aligned against the inside margin of the column (towards the presumed binding edge of the duplexed page) and not concatenated, you can use the .NF INSIDE control word to produce the following output.

"Late last night I saw the new moon,
With the old moon in his arm,
And I fear, so fear, my captain dear,
That we will fall to harm."

This is equivalent to .NF LEFT for odd pages and .NF RIGHT for even pages.

If you want your text to be aligned against the outside margin of the column (away from the presumed binding edge of the duplexed page) and not concatenated, you can use the .NF OUTSIDE control word to produce the following output.

```
"O who is this has done this deed,
                This ill deed done to me,
send me out this time of the year,
                To sail upon the sea!"
```

This is equivalent to .NF RIGHT for odd pages and .NF LEFT for even pages.

Overdraw Options

With concatenation suspended, if the input line is longer than the output column line length or if concatenation is on and a single word is longer than the column line length, the placement of excess characters depends on the other parameters of the .FO [Format Mode] or .NF [No Formatting] control words:

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

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

Splitting Text

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 left and right tabs as described in "Using Tabs" on page 84. You can also 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
```

In this example, the slash (/) is used as a delimiter to separate the control word 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 Text Programmer's Guide¢¢Control Words¢
```

is formatted as:

```
SCRIPT/VS Text Programmer's Guide                       Control Words
```

The space between the parts of split text can be left blank or 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.[25] The default action is to repeat the fill string. For example,

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

results in:

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

You can also cause the fill string to be centered by specifying the C parameter on the .SX control word. For example,

```
.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 allow the left side text to extend past the column line length or truncate it at the column line length depending upon the overdraw option in effect (if EXTEND is in effect it will extend, if FOLD or TRUNCATE is in effect it will truncate). To prevent this, specify the F parameter. This parameter causes SCRIPT/VS to fold the

---

[25] A fill string that is to be centered and not repeated may be as long as the space remaining between the left-side text and the right-side text. If a fill is too long it will be ignored.

portion of the left-side text that does not fit on the current line onto the next line. A foldable split text, as used in tables of contents, could be specified as:

```
.of 1
.sx f /An example of a folded split text line/ ./58/
```

The result is:

```
An example of a folded split
 text line    . . . . . . . . 58
```

The fill character and the right-side text are never folded. The F parameter can be particularly useful when producing such things as a list of illustrations that has figures with long captions.

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[26], 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.
```

---

[26] This is not always true during GML processing. See "Residual Text Processing" on page 283 for more details.

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

```
The quick brown
.br
fox came over to greet ... but
.br
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 the *Document Composition Facility: SCRIPT/VS Language Reference.*

Indenting

To improve readability or emphasize a block of text, you may want to alter the left or right column margins. 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.

These control words normally cause a break. When the NOBREAK parameter of .IN and .IR is specified, a break is not performed.

Simple Indention

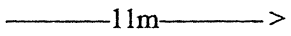
The most basic form of indention is simple modification of the left or right margin. When the indention is zero, all text output lines originate in the leftmost print position of the column as specified with the .PM [Page Margins] control word or the BIND option of the SCRIPT command. By increasing the 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 incremental value to be applied to the

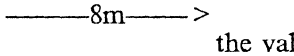
current left margin. This is called *relative* indenting. For example, by specifying

```
. in +5m
```



the value 5M is added to the current left margin. In this example, 6M + 5M is 11M, so the current left margin is now 11M to the right of the column origin. You can move the current left margin to the left by specifying a negative value. For example, by specifying

```
. in -3m
```



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

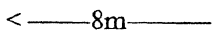
You can return the left margin to the column origin by specifying

```
. 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 can be moved to the left.

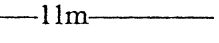
For example, by specifying

```
. ir 8m
```



the right margin is moved 8M to the left. As with .IN [Indent] you can modify the current right margin using relative values. For example, by specifying

```
. ir +3m
```



the value 3M is added to the current right indent. In this example 8M + 3M is 11M, so the current right margin 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 can 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 speci-

fied 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 5 for 1
```

The indention of five spaces 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 can be applied to a permanent indention. For example,

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

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

The .IL [Indent Line] and .UN [Undent] control words provide functions similar to the FOR parameter of .IN [Indent]. Figure 7 on page 82 illustrates a more general use of temporary indention with both .IN and .IR.

```
Current margins:
|<-----                                                        ----->|
The current left margin is the 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 Line Length] and .IR [Indent Right] control words.

 .IL 5 [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 9 [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 5 [Offset]:
         |---->|
         The first line following the offset control word is not indented
              from the current left margin;  all subsequent lines are in-
              dented. The offset remains in effect until changed by another
              offset or indent control word. (Changes affect the current left
              margin after one output line.)

 .UN 5 [Undent]:
    |<----|
    The first line following the undent control word is shifted to the left
         of the current left margin; all subsequent lines start at the cur-
         rent left margin. (Changes affect the current left margin for one
         line.)

 .IR 5 [Indent Right]:
                                                             |<-----
         All lines following an indent right control word are justified
         to the column line length minus the right indention. (Changes
         affect the current right margin for all subsequent lines until
         respecified.)
```

Figure 6. How the Current Margins Are Established

By default, the .IN [Indent] and .IR [Indent Right] control words cause a break, and take effect on the next output line. For example, if you enter

```
Some lines of text that have
little or no meaning to anyone
.in .5i for 3
and use the .IN control word
to request an indention of
one-half inch for the next 3
lines or until indention is
reset.
```

the result is:

```
Some lines of text that have
little or no meaning to anyone
      and use the .IN control
      word to request an
      indention of one-half
inch for the next 3 lines or
until indention is reset.
```

If you do not want a break to occur, you can use the NOBREAK parameter of either the .IN [Indent] or .IR [Indent Right] control words. For example, if you enter

```
Some lines of text that have
little or no meaning to anyone
.in .5i for 3 nobreak
and use the .IN control word
to request an indention of
one-half inch for the next 3
lines or until indention is
reset.
```

the result is:

```
Some lines of text that have
little or no meaning to anyone
      and use the .IN control
      word to request an
      indention of one-half
inch for the next 3 lines or
until indention is reset.
```

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 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.
```

The FOR and AFTER parameters of the .IN [Indent] and .IR [Indent Right] control 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 is decreased; if you specify it as a positive value (+), the current margin is increased. After the duration of a temporary indention has been reached, the current margin reverts to the permanent indention that was in effect before the temporary indention. If another temporary indention is encountered prior to the completion of an existing one, the existing one is immediately stopped and the new margin is the sum of the permanent indention margin and the new temporary indention. A temporary margin change can either start immediately (if the AFTER parameter is not specified) or after the vertical 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

Using Indention with Tabs

A definition list contains definition terms of varying length followed by the text that defines these terms. To ensure that all the text lines originate at the same point on the output line, follow each term with a tab to the current indention.[27] For example, if you specify

[27] You can also use the TO parameter of the .IS control word to perform a single, immediate tab to the value of the current indention.

```
.in 12m
.tp 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.
```

The result will be

TERM	DEFINITION
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 next output line. If you did not use the tab or the .IS [Inline Space] control word, 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 Printing Subsystem.
- With proportional fonts, such as those you can use with page printers, the exact amount of space required may vary depending on the width of the characters on the left hand side.

Indentions and tab-like results similar to those described above can be created with several other control words as well. The .IL [Indent Line] control word, for example, can be useful for beginning paragraphs. To create a paragraph with just the first line indented you could enter

```
.il 3m
```

which results in output like this:

```
   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.
```

Another way to make tab-like indentions is to use the .OF [Offset] control word. Since the .OF control word does not take effect until after the next line is formatted, you could enter

```
.of 3
```

to achieve the following results:

```
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
  three character spaces from
  the current margin setting.
```

To end offset, enter

```
.of
```

and the effect of any previous .OF request is cancelled and all output after the next line continues at the current left margin setting.

You can create output much like that shown in the .OF example above by using the .UN [Undent] control word. If, for example, you have a normal indention of 3 picas from the left margin you could enter:

```
.un 3p
```

to achieve the following results:

```
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.
```

Using Tabs

Use the .TP [Tab Position] control word to define how tab characters (hexadecimal 05) are to be resolved.

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 the hexadecimal 05 using the .TI [Translate Input] control word. The .TI [Translate Input] control word can be used to translate any keyable character to a tab character on input. For example, to set the not sign (¬) character to a tab character, specify

  ```
  .ti ¬ 05
  ```

  This causes every ¬ 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., which has the value of a tab character (hexadecimal 05), anywhere in a text line where a tab is needed. 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. Some text editors, such as the CMS editor, allow you to assign the tab function to any keyable character. When the specified character is entered, the editor changes it into a tab character.

- Build the input file using an input device that can generate a hexadecimal 05 in response to pressing a key (some terminals, the IBM 2741 Communications Terminal, for example, have a special key that, when pressed, generates a tab character.)

  The last two techniques have the disadvantage of putting nondisplayable data into the input file. When such a file is examined with a different terminal or editor, the tab characters may be invisible.

Processing Tabs

When SCRIPT/VS processes an input line and encounters a tab character, it formats the line using the current tab settings that were established using the .TP [Tab Position] control word.

The default tab settings (the ones SCRIPT/VS uses if you do not specify any with the .TP control word) are at every fifth horizontal space[20] position to position 80.

If a tab character is found in an input line, the text following the tab character is positioned at the next tab position. For example, these input lines

```
.tp .5i 1.5i 2.5i
Position&$TAB.this with a tab.
```

will be formatted as

```
Position          this with a tab.
|---->  ---------->  ---------->
```

Tab positions can be added to those currently in effect. These input lines

```
.tp add 1i
Position&$TAB.this with a tab.
```

will be formatted as

---

[20] Horizontal space is based on the width of the figure space in the initial font.

```
Position    this with a tab.
|---->  ---->  ---->  ---------->
```

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

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

results in the following format:

```
         This line starts with a tab.
|---------->---------->---------->
```

Once a .TP control word has been processed, the tab settings remain in effect until explicitly reset by another .TP control word.

You can add tab settings to the ones that already exist by including the ADD parameter when specifying the .TP [Tab Position] 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

```
.tp 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 .TP [Tab Position] control word 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

```
.tp 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 .TP SET followed by the new tab settings that you want to have in effect. For example, specifying

```
.tp set 10m 20m 40m 60m
```

leaves you with tab settings at positions 10m, 20m, 40m, and 60m regardless of the previous tab settings.

If the .TP control word is entered with no parameters, the initial tab positions at 5, 10, 15, ..., 80 are restored. For example, the input lines

```
.ti ¬ 05
.tp
¬*¬*¬*¬*¬*¬...
```

will be formatted as

```
    *    *    *    *    *    ...
|---> ---> ---> ---> ---> --->
```

Tab Fill Characters

Ordinarily, tab characters are replaced with an amount of horizontal white space sufficient to position the text following the tab character at the next tab position. You can specify a "tab fill character" to be used instead of horizontal white space. For example,

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

is formatted as:

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

You can specify different fill characters for each tab setting position you specify with the .TP control word. The fill character and its tab position designation are separated by a slash (/). For example, the input lines

```
.ti ¬ 05
.tp ./10 ,/20 -/30
hup¬one¬two¬ten
```

will be formatted as

```
hup.......one,,,,,,,two-------ten
├─────────> ─────────> ─────────>
```

The fill character is formatted in the current font when the fill string is being formatted.

If the space to the next tab stop is less than the width of one fill character (or less than 24 pels, for 3800 line printers), the tab stop after the next is used.

On the 3800 Printing Subsystem, fill characters are only supported with monospaced fonts. If you use fill characters with proportionally spaced fonts, vertical misalignment may result.

Tab Positioning and Alignment

You can use the .TP [Tab Position] control word to specify that text following a tab character is to be left- or right-aligned at a tab position, centered about a tab position, or aligned with the first occurrence of a particular character positioned at a tab position.

Tab characters at the beginning of an input line cause a break in concatenation. Therefore, you can use tab characters to create simple lists. For example, the input lines:

```
.ti ¬ 05
.tp 5m
Go look for:
¬Jake
¬Frederick
¬Santiago
```

are formatted as:

```
Go look for:
     Jake
     Frederick
     Santiago
├──> ──────────>
```

The text following a tab character is normally left-aligned at the next tab position. You can also define tab positions at which text is centered and right-aligned. For example,

```
.ti ¬ 05
.tp .5i right 1.5i center
¬1)¬Dog
¬2)¬Crump
¬3)¬Cramp
¬4)¬Tackled
¬5)¬Bedazzled
```

The text following the first tab character will be right-aligned at the first tab position, one-half inch from the margin. The text following the second tab character will be centered about the second tab position, one and one-half inches from the margin:

```
    1)          Dog
    2)         Crump
    3)         Cramp
    4)        Tackled
    5)       Bedazzled
├──> ──────────>
```

You can also indicate that the text following a tab character be aligned with the first occurrence of a specific character at the tab position. For example,

```
.ti ¬ 05
.tp .5i left -/2i char .
¬Expensive¬$1234.
¬Cheap¬$.005
¬Reasonable¬$1.00
```

The text following the first tab character will be left-aligned at the first tab position. The text following the second tab character will be positioned so that the first period is left-aligned at the second tab position and the space between the two pieces of text will be filled with hyphens (-):

```
     Expensive----$1234.
     Cheap------------$.005
     Reasonable------$1.00
├──> ──────────────────>
```

You can also use the .TP control word to interchange columns of data without actually changing the input data. In the following example, the .TP control word is used to define tab positions at 10, 20, and 45. The text following the third tab position at 45 will be left-aligned with the period.

```
.ti ¬ 05
.tp   10  20   45 char .
¬name 1¬description of 1¬$11.50
¬name 2¬description of 2¬$1.50
¬name 3¬description of 3¬$33
¬name 4¬description of 4¬$0.50
¬name 5¬description of 5¬$44.50
¬name 6¬description of 6¬$101.50
```

This results in the following columns of data. Name is placed at position 10, description at position 20 and cost at position 45.

```
name 1   description of 1   $11.50
name 2   description of 2    $1.50
name 3   description of 3   $33
name 4   description of 4    $0.50
name 5   description of 5   $44.50
name 6   description of 6  $101.50
```

When using the .TP control word, tab stops need not be specified in ascending order. This allows you to rearrange columns of data without changing the data. By changing the tab stops as shown below, you can rearrange the data so that the description appears first at position 10, followed by name at position 32, and cost at position 45.

```
.ti ¬ 05
.tp   32  10   45 char .
¬name 1¬description of 1¬$11.50
¬name 2¬description of 2¬$1.50
¬name 3¬description of 3¬$33
¬name 4¬description of 4¬$0.50
¬name 5¬description of 5¬$44.50
¬name 6¬description of 6¬$101.50
```

Resulting in the following columns of data:

```
description of 1   name 1   $11.50
description of 2   name 2    $1.50
description of 3   name 3   $33
description of 4   name 4    $0.50
description of 5   name 5   $44.50
description of 6   name 6  $101.50
```

Using Inline Spacing for Tabs

Another way to create immediate tabs is to use the .IS [Inline Space] control word. That is, using the .IS control word, you can create an immediate tab that will position subsequent text at a specific point on a line without disturbing the current tab settings established with the .TP [Tab Position] control word. For example,

```
Sign in ink please:
.is to 8p
```

The amount of inserted space is the difference between the amount specified and the width of the text preceding it in the output line:

```
Sign in ink please:  _____
```

An immediate tab will be considered missed if the width of the text already placed on the output line plus the value specified on the **MINIMUM** parameter exceeds the value specified with the TO parameter of the .IS control word. If it is not specified, the **MINIMUM** value defaults to one horizontal device unit. If the **BLANK** parameter has been specified, then a missed immediate tab will be treated as an ordinary wordspace. For example,

```
Place your score here:
.is to 1i blank
_____
```

will result in

```
Place your score here: _____
```

If the **BREAK** parameter has been specified, then a missed immediate tab will cause a break and the immediate tab is processed on a new output line. For example,

```
Place your score here:
.is to 1i break
_____
```

will result in

```
Place your score here:
      _____
```

If the **ABSOLUTE** parameter is specified, the immediate tab will always be processed on the current output line; if the tab is missed, negative horizontal space will be inserted. For example,

```
Place your score here:
.is to 1i absolute
_____
```

will result in

```
Place your score here:
```

Leading Blanks and Tabs

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, 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.[29]

---

[29] Note that input lines processed in literal mode, under the .LI [Literal] control word, do not invoke the .LB or .LT functions. Also, GML scan processing may cause .LB or .LT not to be processed.

If you have defined a .LB macro or a .LT macro and macro substitution is on, the .LB macro will be executed whenever a leading blank is processed or the .LT macro will be executed whenever a leading tab is processed.

Note, however, that after the .LB or .LT control word or macro is processed, the leading blank or tab is still on the line and it is processed as part of that text input line. In other words, you cannot use the .LB or .LT macro to remove leading blanks or tabs from a line.

Blank and Null Lines

Whenever SCRIPT/VS encounters a blank input line, it generates and processes a .BL control word which has the same effect as a .SP control word.

Blank lines can originate from:

- A source input file (not all systems in which SCRIPT/VS operates allow this)
- A macro line that is blank
- Terminal input (.TE)
- A line containing control word separators with only blanks between them
- A non-blank line that becomes blank as a result of symbol substitution.

To redefine the SCRIPT/VS implicit formatting convention for blank lines, define a .BL [Blank Line] macro that will be processed whenever a blank line is encountered and macro substitution is on. For example,

```
.dm bl /.sk 2
```

Now, when SCRIPT/VS encounters a blank line, the result is two line spaces on your output page.

Note that a blank line is not the same as a null line. Null lines contain no characters and are processed by the .NL [Null Line] control word.

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 reset line continuation in case the previous line ended with a continuation character.

Like blank lines, null lines can also originate from a number of sources:

- A source input file
- Terminal input (.TE)
- A non-null line that becomes null as a result of substitution
- A macro line that is null.

To redefine the SCRIPT/VS implicit formatting convention for null lines, define a .NL [Null Line] macro that will be processed whenever a null line is encountered and macro substitution is on. 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 /.*
```

Full Stop Characters

Normally, when concatenation of input lines is in effect, SCRIPT/VS inserts a word space between the last word of each input line and the first word of the next input line. If the input line ends in a full stop, SCRIPT/VS will add a second word space, unless, for example, continuation is performed.

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 (?), an exclamation point (!), or a colon (:). 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. For example, if you enter

```
.dc stop : .
```

only the colon and period will result in full stops.

Determining Word Space Values

Each font is designed with a default wordspace value appropriate to the size of the characters and SCRIPT/VS normally uses this as the width of blanks. But when more than one blank is found in text, the first blank is considered a word space and any other blanks are considered extra spaces. The width of the first blank is determined by the .WS [Word Space] control word. The width of each subsequent, successive blank is determined by the .ES [Extra Space] control word.

You can use the .WS [Word Space] control word to control the width of word spaces in your text. If, for example, you specified

```
.ws normal p8
```

then the width of all word spaces will be eight pica points, as shown in the following example:

```
The width of all word spaces will
be eight pica points.
```

Until changed, this new value will remain in effect for all subsequent font changes. If you want to revert to default wordspacing, then specify

```
.ws
```

to restore the default wordspace values of your current font.

The BY parameter of the .WS [Word Space] control word also can be used to increase or decrease wordspace values. If, for example, you wanted to decrease word spaces, you could specify

```
.ws by .8
```

which means that the current wordspace value (either the default wordspace value of the current font, or the value you specified with the NORMAL parameter of the .WS [Word Space] control word) will be multiplied by .8 to give a fixed wordspace value equal to 80% of the currently set value, as shown in the following example:

```
The current wordspace value will be
multiplied by .8 to give a fixed
wordspace equal to 80% of the
currently set value.
```

If you wanted to increase word spaces, you could specify

```
.ws by 1.2
```

which means that the current wordspace value (either the default word space value of the current font, or the value you specified with the NORMAL parameter of the .WS [Word Space] control word) will be multiplied by 1.2 to give a fixed wordspace value equal to 120% of the currently set value, as shown in the following example:

```
The current wordspace value will be
multiplied by 1.2 to give a fixed
wordspace equal to 120% of the
currently set value.
```

The width of word spaces may be increased or decreased according to the expansion and compression ranges given on the .HY [Hyphenate] control word. See "Chapter 7. Hyphenating and Horizontally Justifying Text" on page 99 for more details on using the .HY [Hyphenate] control word. Wordspace values set with the .WS [Word Space] control word are also subject to horizontal justification.

Values given with the .WS [Word Space] control word are subject to rounding in accord ance with the limitations of the device.

Determining Extra Space Values

The .ES [Extra Space] control word is used to specify the width of extra spaces when more than one blank follows some part of your text.

If you want to establish the width of extra spaces in your text, regardless of the default value (which is equal to the wordspace value of the current font), you can do so using the .ES NORMAL control word. Specifying

```
.es normal p6
```

will fix the width of extra spaces in your text to six pica points regardless of any currently set defaults. Until changed, this new value will remain in effect for all subsequent font changes.

The .ES control word is also particularly useful if you use the typing convention of following sentences with two blanks. The .ES control word can be used to decrease the width of the second blank without changing the width of other word spaces.

If, for example, you were using a proportional typeface and you wanted to set the extra spaces after a full stop to be half as wide as word spaces you would enter

```
.es by .5
A full stop.  And an additional line.
```

which results in:

```
A full stop.  And an additional line.
```

If you enter

```
.es
```

then SCRIPT/VS resumes using the values determined by the current font.

Extra space values set with the .ES [Extra Space] control word are also subject to horizontal justification.

Values given with the .ES [Extra Space] control word are subject to rounding in accordance with the limitations of the device.

Inserting Horizontal White Space

In addition to using the .IS [Inline Space] control word to perform an immediate tab to a specific position on a line, you can use it to insert a specified amount of horizontal white space between two words.

The .IS control word can be thought of as a required blank or backspace character of arbitrary width, depending upon whether the inserted space is positive or negative, respectively.

To include a fixed amount of horizontal white space in text, you could specify:

```
Shadwell
.is .5i
was sacked.
```

The inserted space is treated as a single character, and is not subject to justification:

```
Shadwell      was sacked.
```

The inserted horizontal white space may be negative:

```
An underscored
.is -.5i
______ word.
```

The inserted space is treated as a single backspace character:

```
An underscored word.
```

Revision Codes

If you process documents that are frequently revised, you can identify revised text with a revision code in the left margin. Use the .RC [Revision Code] control word to identify changed material. You can define as many as 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 the revision code used has not been
! defined, no revision code will appear because the default revision code is a blank. The
! control word

```
.rc 1 off
```

indicates the end of the revised piece of text.

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.

To turn revision code 1 off, specify

```
.rc 1 off
```

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
```

! before that line rather than bracketing the line with ".rc 1 on" and ".rc 1 off".

You may also flag a single line by specifying

```
.rc * $
```

$ without changing any other revision codes.

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 or the .PM [Page Margins] control word. 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.

Revision Code Considerations

The 3800 Printing Subsystem

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 available in the current font, this can be achieved by combining the RC character with a special blank that brings the total width of RC and blank to 30 pels. The following table shows relative widths:

RC Width	Blank Width
12	18
15	15
18	12

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 pels.
- The width of the intercolumn gutter, less the width of the RC field, should be 0, 11-19, or more than 23 pels.

If these restrictions are violated, inline space errors of up to 6 pels can result, as illustrated in Figure 8.

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 8. **Justification Alignment Error for 3800 Printing Subsystem Output:** When requesting horizontal space values, you must remember that the values will be rounded to be a multiple of 3 pels with the exception that 3, 6, 9, and 21 are not obtainable. As a result, you may not get the exact space that you requested.

Chapter 7. Hyphenating and Horizontally Justifying Text

This chapter describes the SCRIPT/VS control words you can use to hyphenate and horizontally justify your text.

Hyphenation and Horizontal Justification

Hyphenation

SCRIPT/VS hyphenates words in your text based on the values you specify with the .HY [Hyphenate] control word. You can turn hyphenation on by specifying

```
.hy on
```

or turn hyphenation off by specifying

```
.hy off
```

When .HY ON is specified, hyphenation is controlled by the values that you assign to the MINPT, MAXPT, MINWORD and LADDER parameters of that control word.

- MINPT controls the minimum hyphenation point: the smallest number of characters before the hyphenation point that is acceptable.

  The initial value for MINPT is 4. However, in large column line lengths, it may be preferable to hyphenate a word after the third character. In short column line lengths, it may be preferable to hyphenate a word after the second character. For example, you might want to specify

  ```
  .hy minpt 2
  ```

  After this control word is encountered, there must be at least two characters left on the line before SCRIPT/VS will hyphenate the word.

- MAXPT indicates the minimum number of characters that is acceptable after the hyphenation point.

  The initial value for MAXPT is 2. However, in large column measures, it may be preferable to have at least three characters left after a word is hyphenated. In small column measures, it may be preferable to have at least two characters left after a word is hyphenated. For example, you might want to specify

  ```
  .hy maxpt 3
  ```

  After this control word is encountered, there must be at least three characters left in the word after hyphenation has taken place.

- MINWORD specifies the minimum size that a word can be before it is eligible for hyphenation.

  The initial value for MINWORD is 6. However, in large column measures, it may be preferable to hyphenate words that are at least seven characters long. In small column measures, it may be preferable to hyphenate words that are at least five characters long. For example, you might want to specify

  ```
  .hy on minword 7
  ```

  After this control word is encountered, a word must be at least seven characters long before hyphenation will be attempted.

  Note that the sum of the MINPT and MAXPT values can not exceed the MINWORD word value.

- LADDER specifies the maximum number of lines that may be hyphenated consecutively.

  The initial value for LADDER is 2. To change it, you could specify

  ```
  .hy on ladder 3
  ```

  After this control word is encountered, up to three consecutive lines are eligible for hyphenation.

The .HY RANGE Control Word and Horizontal Justification

You can use the RANGE parameter of the .HY control word to specify the factors by which word spaces may be compressed or expanded to avoid hyphenation.

Compression and expansion values are specified with the .HY RANGE control word[30] as in the following example:

```
.hy range .75 1.2
```

The following three steps are performed whether hyphenation is on or off. If .FO OFF is in effect, then .HY RANGE is ignored.

1. Given the compression and expansion values, SCRIPT/VS first attempts to keep the last word on the line by compressing word space values in the line by up to 75 percent.
2. If compression fails, SCRIPT/VS attempts to force the word off the line by expanding the wordspace values by up to 20 percent.
3. If compression and expansion with the .HY RANGE control word both fail, SCRIPT/VS checks to see if hyphenation is on.

If hyphenation is on, SCRIPT/VS then checks to see if the value set with the LADDER parameter of the .HY control word has been exceeded. If it has, then the word is moved onto the next line. If the specified LADDER value has not been exceeded, then hyphenation is attempted based on the MINPT, MAXPT and MINWORD values you have specified. If hyphenation is off, the word is moved to the next line.

---

[30] Because horizontal adjustments must be made in even multiples of the horizontal escapement of the device, the exact amount of adjustment may be more or less than the range you specified.

After the hyphenation step, if justification is on (.FO ON), SCRIPT/VS will then justify the line by expanding the word spaces. The expansion is done proportionally according to the wordspace sizes and the amount of space needed to achieve a fully justified line. The expansion is done without regard to the values given on the RANGE parameter of the .HY control word.

These steps are illustrated in the following figure which assumes these values:

```
.hy on
.hy range .5 2.0
.ws normal 8
```

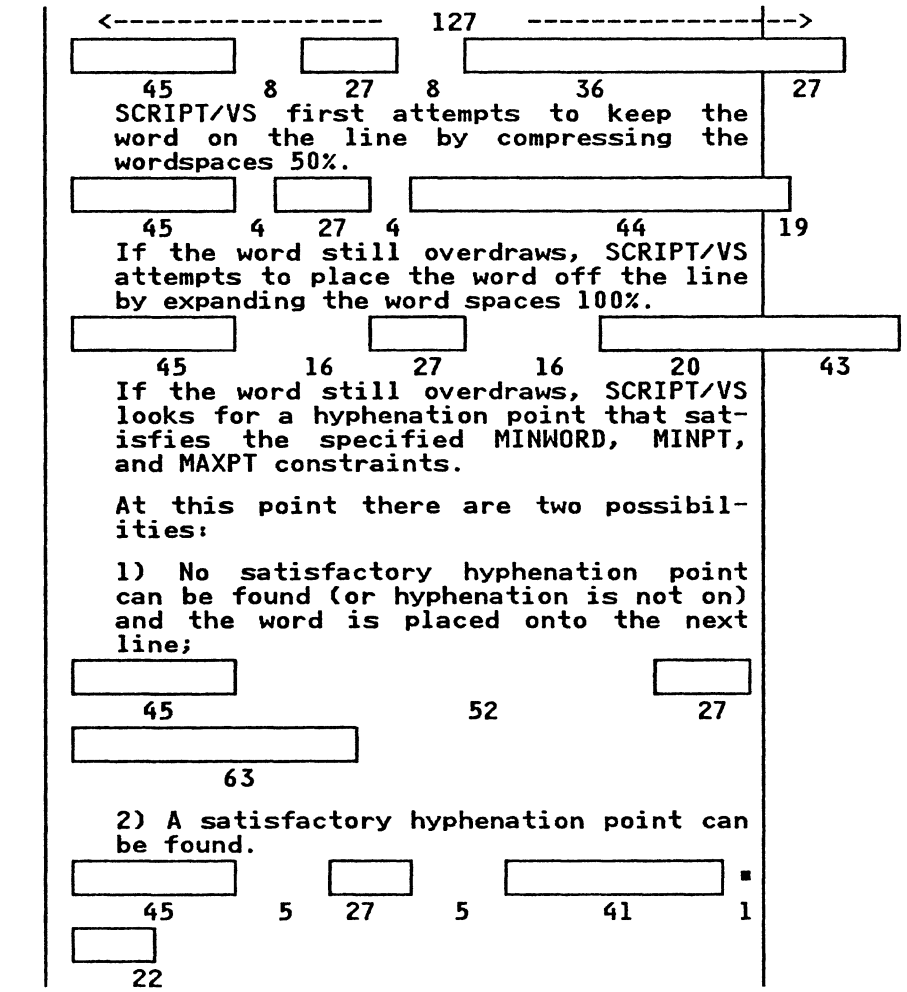


Figure 9. Adjusting an Overdraw Condition: This figure illustrates the steps taken to correct an overdraw condition.

More on Hyphenation

You can also instruct SCRIPT/VS to:

- Search a SCRIPT/VS dictionary to see if there is an entry for the word to be hyphenated
- Use an algorithmic hyphenator, if one is available, to hyphenate the word

Unless otherwise specified, SCRIPT/VS first searches for the word in the addenda, user and main dictionaries in that order. If the word cannot be found there, it will use an algorithmic hyphenator, if one is available, to perform the hyphenation.

If you do not want the addenda dictionary searched, specify

```
.hy noadd
```

If you do not want any of the dictionaries to be searched, specify

```
.hy nodict
```

If you do not want an available algorithmic hyphenator to be used, specify

```
.hy noalg
```

If you want the addenda dictionary searched, specify

```
.hy add
```

If you want any of the dictionaries to be searched, specify

```
.hy dict
```

If you want an available algorithmic hyphenator to be used, specify

```
.hy alg
```

Using an Algorithmic Hyphenator

Unless you use the NOALG parameter of the .HY [Hyphenate] control word, SCRIPT/VS attempts to use an algorithmic hyphenation routine to hyphenate:

- Words that cannot be found in the supplied language dictionaries
- All words if the NODICT parameter of the .HY control word was specified.

An algorithmic hyphenation routine for American 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, there may be occasions when you would like a specific 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,

```
.cl 22m
Guinevere's
.hw lighter--than--air
laughter was regularly heard
.hw through-out
the kingdom and caused her to be
crudely and
.hw un-cer-e-mo-ni-ous-ly
bounced into the heavens.
```

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
regularly       heard
throughout the kingdom
and caused her to be
crudely and unceremo-
niously bounced into
the heavens.
```

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 to indicate that SCRIPT/VS will print the necessary hyphens. "Unceremoniously" is hyphenated at one of the appropriate hyphenation points specified by the .HW control word. Note that the hyphenation points supplied by a .HW word apply only in this instance and nowhere else.

Hyphenation Points and Fallibility

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 starting with the first vowel in the word. Only one hyphenation point is recorded for each of the four groups. See "Chapter 25. Verifying Spelling" on page 289 for a complete description of the SCRIPT/VS dictionaries and their relation to hyphenation.

Chapter 8. Creating Vertical Space

This chapter describes how you can create vertical space in the text.

You can insert space between lines of text and control vertical space in the text by using a blank line, or with any of the following control words:

- .SP [Space]
- .SK [Skip]
- .BL [Blank Line]
- .LS [Line Spacing]
- .LO [Lead-Out]
- .SB [Shift Baseline]

Spaces and Skips

You can insert space between lines of text in your document by using the .SP [Space] and .SK [Skip] control words.

For example, the input lines

```
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 or bottom 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 or bottom of a column (or page), the blank lines

are 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 in a qualified space unit notation. 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.
```

Setting Line Space

On page printers, each font is designed with a default linespace value appropriate to the size of the font. For a particular line, default linespacing is determined by the size of the largest font used on that line.

You can use the .LS [Line Spacing] control word to change the vertical space separating lines of text and to establish limits for increasing or decreasing line spacing in your document for purposes of vertical justification.

You can establish fixed line spacing by entering:

```
.ls normal p15
```

On page printers, each line will have a depth of 15 pica points, regardless of the size of the characters on the line. On line devices, the depth of each line will be the nearest multiple of the vertical device unit. For example, at 6 lines per inch, each line of text occupies 1/6 of an inch; at 8 lines per inch, each line of text occupies 1/4 of an inch.

The BY keyword of .LS can be used to decrease or increase the default linespace value to set text more densely or sparsely. You can, for example, double-space your output by entering:

```
.ls by 2.0
```

Each line of text (and each .SK and .SP, when given in line spaces) will be twice as deep as normal, as shown in the following example:

```
The   following   lines   of   text

including any skips or spaces


will be twice as deep as normal. As

can be readily seen in this brief

but highly entertaining example.
```

On line devices, SCRIPT/VS will use the nearest multiple of the fixed line space value for that device as the depth of each line.

Shifting the Baseline

Fonts are designed so that the characters appear to rest on the normal baseline. If your output is for a page printer, you can use the .SB [Shift Baseline] control word to place characters above or below the normal baseline to create, for example, subscripts or superscripts.

Superscripts may be formatted with baseline shifts. For example, the expression *5-cubed* may be formatted as

```
5
.sb .15cm
.ct 3
```

The *3* will be placed .15 centimeters higher than the *5*.

5^3

A macro can be defined that will create subscripts:

```
.dm subs on
.se height1 = &dv'1
.bf small
.se height2 = &dv'1
.se diff = &height1 - &height2
.sb +&diff.dv
&*1
.sb -&diff.dv
.pf
.dm off
```

In this example, we first set a symbol (height1) to one vertical device unit in the current font, then begin a new font, small (assuming it is a valid, defined font and that it is in a smaller point size than the current font), and set a symbol to one vertical device unit in this new font. Next we set a symbol (diff) to the difference between &height1 and &height2, shift the baseline by that value, and enter the superscript number (&*1). When we are done, we enter .SB - &diff.dv to return the baseline to its previous position and .PF to return to the previous font.

Subscripts may also be formatted with baseline shifts. For example, the expression *x-sub-i* can be formatted as

```
x
.sb -p2
.ct i
```

The *i* will be placed two pica points lower than the *x*.

x_i

You can enter

```
.sb
```

to restore the normal baseline for subsequent text.

Formatting Fractions on Page Printers

Since a limited number of fractions exists in most of the fonts for page printers, you may have to construct fractions. One method of doing this is illustrated in the following steps:

1. Define an appropriate font for the fraction pieces and start the macro definition.

```
.df fraction type(6)
.ms on
.dm fraction on
```

2. Make sure that the parameters passed to the macro are valid. There must be exactly three parameters. The first and third parameters must be numeric and the second one must be a slash (/). We've chosen here to issue error messages if the parameters are invalid.

```
.if &*0 ne 3
.th .mg |E|FRACTION: Missing parameters: "&*."
.th .me
.if &T'&*1 ne N
.or &*2 ne /
.or &T'&*3 ne N
.th .mg |E|FRACTION: Invalid parameters: "&*."
.th .me
```

3. Calculate the amount of

   - Baseline shift needed to position the left hand side of the fraction. This amount is the difference between the normal font em-height and the fraction font em-height.
   - Insert the negative inline space necessary to kern (shift) the pieces properly with respect to the slash: (we've chosen one third of the width of a zero (the figure space) in the fraction font.)

```
.se *sb = &DH'1mv
.bf fraction
.se *sb = &*sb - &DV'1mv
.se *is = &DH'&W'0 / 3
.pf
.if &*sb le 0
.th &*1./&*3
.th .me
```

4. Format the left hand side in the appropriate font and baseline shift. Then restore the original font and baseline.

```
.bf fraction
.sb +&*sb.dv
&*1
.pf
.sb -&*sb.dv
```

5. Kern the amount calculated above, put out the slash and then kern again so that the right hand piece of the fraction is under the slash.

```
.is -&*is.dv
.ct /
.is -&*is.dv
```

6. Format the right hand side in the appropriate font and then restore the original font.

```
.bf fraction
&*3
.pf
```

7. End the macro definition.

```
.dm off
```

The method described above consists of using a baseline shift, inserting space, and defining and using fonts in order to format fractions for page printers. Figure 10 on page 110 shows how such a fraction is constructed.

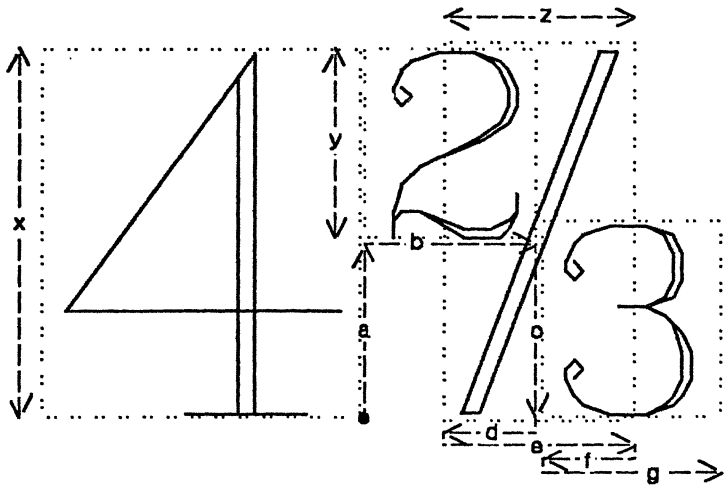


Figure 10. Example of Fractions Formatted on Page Devices.

Chapter 9. Vertically Justifying Text

Vertical Distribution, Formatting and Justification

Distribution

Use the .BC [Balance Columns] control word to distribute text across columns in order to achieve a *balanced* set of columns (the amount of text in each column is as close to equal as possible). 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. Text lines in a block will not be split across columns.

In order to get the proper distribution of text, you must consider:

- Top and bottom floats. If your floats are very large and your columns are small, proper distribution may not be possible.
- Keeps. There is no breaking of keeps. If your keeps are very large and your columns are small, proper distribution may not be possible.
- Widows. Remember that there should be no one-line widows.
- Skips and Spaces. Skips at the top or bottom of a column are discarded and not considered in vertical distribution. It should be noted that an apparent shortness in columns and/or insufficient space in the text may be due to such skips in your text. Spaces, however, are considered and, therefore, an apparent erroneous extra space at the top or bottom of columns may be due to spaces in your text.
- Multicolumn sections. In a multicolumn section, a .CC or .CB control word ends one set of balanceable columns and the new column becomes the first column in a new set of balanceable columns. Of course, if there is only one column in the set, no text is distributed.

In SCRIPT/VS, distribution is not guaranteed to result in columns that are exactly equal in depth. Short columns (more than one text line difference) can result from:

- User-entered unconditional column begins (.CB)
- Floats, skips, spaces, and so on (especially if they are comparatively large in size)
- Very short section depths
- Widow zones and keeps because they are not split across columns
- Variations in line spacing and font sizes.

Vertical Formatting

You can use the .FV [Format Vertically] control word to indicate how formatted lines of text should be placed within the columns of a section.

For example, if you specify

```
.fv top
```

text will be placed at the top of the columns in the section. This is the default value.

If you specify

```
.fv bottom
```

the text will be moved to the bottom of the columns.

If you specify

```
.fv center
```

the text will be centered in the columns.

Vertical Justification

If you want to vertically justify your text, that is, specify that the text in all columns of a section should extend to the same depth, if possible, then specify

```
.fv justify
```

The purpose of vertical justification is to adjust the vertical spacing in each column so that:

- In an unconditionally ended section, the depths of the columns within the section are as equal as possible
- In conditionally ended sections, the columns end at the bottom of the page.

Vertical formatting is applied separately to each section on the page. For example, because section breaks are performed before and after a level 1 (.H1) heading, the heading is vertically formatted within its own section. Since there is no additional space to be distributed within the section, the actual results for the section are the same regardless of the vertical formatting option.

For the last section on the page, however, there may be some additional space to be distributed within the section. If the page was ended unconditionally (as described below in "Section and Page Ending Considerations" on page 113), no distribution of white space occurs and so the vertical formatting does not change the appearance of text on the page. If the page is ended conditionally, extra white space is distributed in the last section according to values you may have specified with the .FV [Format Vertically] control word.

Section and Page Ending Considerations

Sections can be ended unconditionally by the ending of the primary input file, specifying a head level that causes a page or section break, and by any of the following control words:

- .PA [Page Eject]
- .CP [Conditional Page Eject] without a conditional value
- .CD [Column Definition]
- .SC [Single Column Mode]
- .MC [Multicolumn Mode]
- .SK [Skip] with the P parameter
- .SP [Space] with the P parameter.

Unconditional sections occur in multi-section pages or at the end of a chapter or other major division of a document. They will be justified to the depth of the deepest column or some intermediate point between the shortest, non-empty column and the deepest column in the section. If there is only one non-empty column in the section, no vertical justification will be done.

Sections can be ended conditionally by normal text overflow, and by specifying any of the following control words:

- .CB [Column Begin]
- .CC [Conditional Column Begin]
- .CP [Conditional Page Eject] with a vertical parameter.

Conditionally ended sections always end a page and the column(s) in them are always adjusted towards the bottom of the page. The object is to have the columns of the page (such as those in a chapter) end at the bottom of the page. The last page of a chapter is normally unconditionally ended and the column(s) in it will be set "short" as previously described.

If you want the columns to be justified towards the bottom of the page, then you must conditionally end the page. For example, if your page is 26cm deep, you could specify

```
.cp 26cm
```

to end the page conditionally instead of entering

```
.cp
```

which would cause the page to be unconditionally ended.

Other Considerations

Vertical justification is achieved, if possible, by making incremental and proportional adjustments to the sizes of of the following in the order given:

- Leadout points (set with the .LO [Lead-Out] control word)
- Skips (set with the .SK [Skip] control word)
- Spaces (set with the .SP [Space] control word)
- Text lines.

The exact value of the adjustments depends on:

- The number and sizes of any leadout points
- The number and sizes of any skips, spaces, and text lines
- The increment values specified by .LS INCREMENT at any given point
- The ranges specified by the SKIP, SPACE, and TEXT parameters of the .LS control word at any given point
- The size of the vertical escapement of the device for which the document is formatted.

SCRIPT/VS performs rounding so that the vertical spacing adjustments are made in whole increments of the values specified by the INCREMENT parameter of the .LS [Line Spacing] control word. The increment values may also be rounded so that they are an even multiple of the vertical escapement[31] of the device. Thus the values you specify for increments, ranges, and leadout size may be exceeded.

The use of leadout points before headings can be very effective since they already have spacing and some extra space will usually not be objectionable. For example,

```
.lo .5i
.h2 Heading Text
```

allows SCRIPT/VS to add up to an extra half inch of vertical space in front of the heading, if vertical justification has been requested with the JUSTIFY parameter of the .FV [Format Vertically] control word.

You could also consider allowing the sizes of skips to be varied. For example,

```
.ls skip .8 2.0
```

will allow SCRIPT/VS to compress skips by 80% or expand them by up to 200% of their original size[32]. If you have also used leadout points, they will be adjusted before the sizes of the skips.

---

[31] Escapement is the minimum amount by which a particular device can move in the vertical direction. For example, on a six lines-per-inch device it is 1/6 of an inch. This term is also referred to in this book as a vertical device unit.

[32] Because of rounding, the actual values used may be less than or more than the factors you specify.

If you do not want a specific skip to be adjusted, then you can use the A (absolute) parameter of the .SK [Skip] control word. In the following example, even though a skip adjustment range has been specified, the skip will never be adjusted because it has been specified as being "absolute":

```
.ls skip 1.5
The absolute parameter
.sk .5i a
means a skip (or space) is not eligible
for vertical adjustment.
```

After leadouts and skips, you may want to allow SCRIPT/VS to adjust the sizes of spaces. You could, for example, use

```
.ls space 1.2
```

which indicates that the size of spaces can be expanded as much as 120%, if necessary, in order to achieve vertical justification. Again, the A (ABSOLUTE) parameter of the .SP [Space] control word disallows a particular space from being expanded or compressed for vertical justification.

As a last resort, you may consider the TEXT parameter of the .LS [Line Spacing] control word. This is most appropriate for single column documents where variations in text linespacing are usually less noticeable (and therefore less objectionable) than for multi-column documents.

It is possible that SCRIPT/VS will be unable to achieve full vertical justification, if so you may need to consider the following potential problem situations:

- Extremely short columns in the section or page. About the only reasonable thing that you can do is to add more text or, perhaps, cause the previous page to end sooner so that more text is formatted on the short page.
- Insufficient number of objects to justify. You need to insert more leadout points or, perhaps, allow SCRIPT/VS to adjust skips (if you have not already done so).
- Insufficient size variations. Perhaps you should increase the size of the leadout points that you are using or you could increase the appropriate range factors.
- Extremely short or long columns. This may be due to large keeps and floats. You should reduce the text in the keep or float or split it into two or more smaller ones.

  An apparently unjustified column may actually be the result of a space (.SP) at the top or bottom of a column. If the results are not what you desire, rearrange the text and/or spaces in order to obtain more desirable results.

Chapter 10. Establishing Page Layout

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 Headings: Descriptive information that precedes the body of text on each page.
- Running Footings: Descriptive information that follows the body of text on each page, printed after footnotes, if any.
- 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.

See Figure 11 on page 118 for an illustration of the layout of a SCRIPT/VS output page. Control words used to specify the size or contents of each area are shown in parentheses.

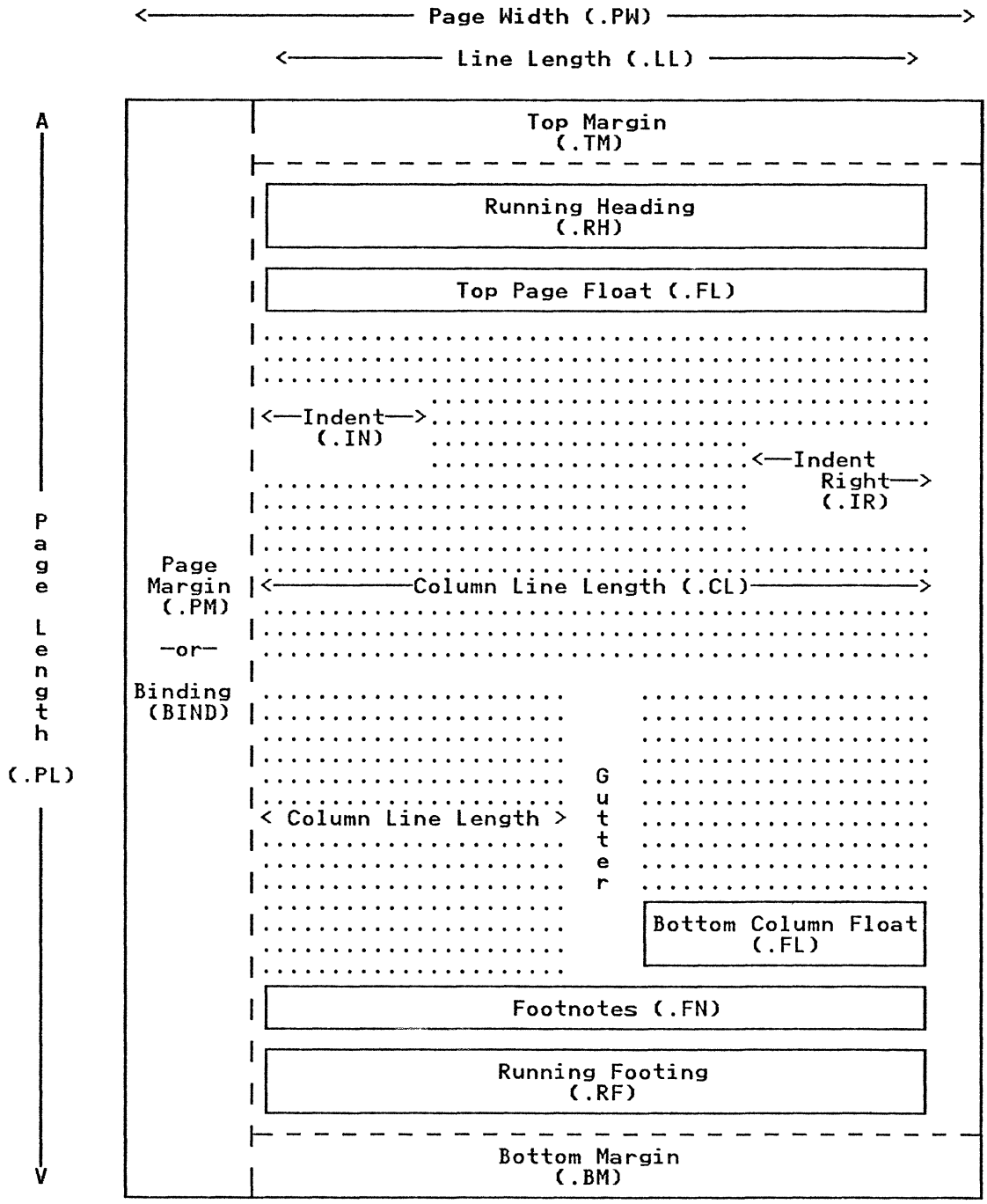


Figure 11. SCRIPT/VS Terms for Parts of the Page.

Default 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 the discussion of the DEVICE option of the SCRIPT command in the *Document Composition Facility: SCRIPT/VS Language Reference*).

When SCRIPT/VS formats output for logical line devices, each SCRIPT/VS page has default dimensions. For example, the page dimensions for a 1403N6 logical device are:

- 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 line devices, the values are 60, 80, and 120 respectively, because one inch of the form is reserved by the 3800 Printing Subsystem. One inch of the page length is reserved for top and bottom margins.
- 6 inches wide (60 characters at 10 pitch, 72 characters at 12 pitch, and 90 characters at 15 pitch).

See Figure 12 on page 120 for page printer default page dimensions.

Although the initial page length, page width, 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], .PW [Page Width], and .LL [Line Length] control words. Some things which you need to take into consideration are:

- The physical size of the paper on which you are printing SCRIPT/VS output.
- The number of lines printed per page depends on the linespacing of the fonts used.
- The number of characters per line depends on the fonts used in the line.
- The fact that the 3800 Printing Subsystem line device reserves one-half inch at the top and bottom of the page that is not counted in the page length.
- For page printers, you must reserve some white space at the bottom of the page to allow for descenders.

Page length includes all of the page that is accessible to SCRIPT/VS. For non-3800 line devices, this is the entire form (the vertical distance between perforations for continuous forms). The 3800 Printing Subsystem reserves 1/2 inch (12.7 mm) above and below the perforation, and makes it inaccessible for printing. Consequently, for 3800 logical devices, page length does not include 1/2 inch (12.7mm) at the top and bottom of the page.

Changing Page Dimensions

There are a number of control words that allow you to override default page dimensions. Among these are:

- .PM [Page Margins]
- .PL [Page Length]
- .PW [Page Width]
- .TM [Top Margin]
- .LL [Line Length]
- .BM [Bottom Margin]

Logical Device Type	Real Device Type	Lines per Inch	Page Size		Margins			Line Length	Class of Device
			Width	Length	Bind	Top	Bottom		
TERM 2741 3270	(1) 2741 3270	6	8.5i	11i	2	.5i	.5i	6i	
1403N6	1403	6	8.5i	11 i	1i	.5i	.5i	6i	line devices
1403N8		8	8.5i	11 i					
1403W6		6	13.5i	11 i					
1403W8		8	13.5i	11 i					
1403W6S		6	13.5i	8.5i					
1403W8S		8	13.5i	8.5i					
1403SW (2)		6	8.5i	11 i					
STAIRS		6	13.5i	11 i					
3800N6	3800	6	8.5i	10 i	1i	0	0	6i	
3800N8		8	8.5i	10 i					
3800N12		12	8.5i	10 i					
3800W6		6	13.5i	10 i					
3800W8		8	13.5i	10 i					
3800W12		12	13.5i	10 i					
3800N6S		6	11 i	7.5i					
3800N8S		8	11 i	7.5i					
3800N12S		12	11 i	7.5i					
3800W6S		6	13.5i	7.5i					
3800W8S		8	13.5i	7.5i					
3800W12S		12	13.5i	7.5i					
38PPN	3800-3	(3)	8.5i	10 i	1i	0	.125i	6i	page devices
38PPW			13.5i	10 i					
38PPNS			11 i	7.5i					
38PPWS			13.5i	7.5i					
38PPW90			10 i	13.5i	.5i	.5i	.5i	6i	
38PPNS90			7.5i	11 i					
38PPW270			10 i	13.5i					
3820A	3820	(3)	8.5i	11 i	1i	.5i	.5i	6i	
3820A90			8.5i	11 i					
3820A180			8.5i	11 i					
3820A270			8.5i	11 i					
3820L			8.5i	14 i					
3820A4			210mm	297mm					
3820B4			257mm	364mm					
3820B5			182mm	257mm					
4250A	4250	(3)	8.5i	11i	1i	.5i	.5i	6i	
4250B			11 i	17i					
4250L			8.5i	14i					
4250A3			297mm	420mm					
4250A4			210mm	297mm					

1 The physical device type corresponding to the TERM logical device can be either 2741 or 3270, depending upon the actual terminal type.

2 This is a 12-pitch device; all other 1403 devices are 10-pitch.

3 The linespacing for page devices is determined by the .LS [Line Spacing] control word and the fonts used in the document.

Figure 12. SCRIPT/VS Logical Devices: This table lists the logical devices that can be specified with the DEVICE option of the SCRIPT command, and the default page dimensions for each. The page size can be changed with the .PW [Page Width] and .PL [Page Length] control words. The page margins can be changed with the .PM [Page Margins], .TM [Top Margin], and .BM [Bottom Margin] control words.

All of these control words take effect on the next page.

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.

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 margins if they were established using the BIND option of the SCRIPT command. For example,

```
.pm 6
```

sets the page margin to six character spaces, whereas

```
.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,

```
.pm +9mm
```

increases the page margin by nine 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,

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

causes the formatted output to be shifted nine picas to the right for odd-numbered pages and six 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. If the BIND option was not specified, then the default bind is restored.

If the BIND option is not specified, the initial setting for the page margins is established by the logical device. In this case, this initial setting can be changed by specifying new values with the .PM control word. For example, if you specify,

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

the initial settings for the page margins are changed to nine picas for odd-numbered pages and six picas for even-numbered pages. NOBIND indicates the .PM control word should be processed only if the BIND option of the SCRIPT command was not specified. If the NOBIND parameter is not specified, the .PM control word unconditionally overrides the initial setting.

Changing the Page Length

Page length can be changed using the .PL [Page Length] control word. For example, you might specify

```
.pl 68
```

which will set the page length to 68 lines.

You may need to adjust a page dimension to handle a special situation in your document. Instead of changing the page length, you may be able to 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

```
.tm +3
```

To restore the original top 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 Page Width

You can specify the width of the output page by using the .PW [Page Width] control word.

The page width includes both the page margins, as determined by the .PM [Page Margins] control word (or binding, as established with the BIND option of the SCRIPT command), and the line length, as determined by the .LL [Line Length] control word. These relationships are illustrated in Figure 11 on page 118. The unbound margin of a page equals the page width minus the size of the binding and the line length. All text must be placed within the page, as defined with .PW.

Changing the Line Length

When you are changing the default dimensions of the page, you should consider the length of lines as well as the width of pages. The SCRIPT/VS line length default is based on the logical output device, and can be changed by using the .LL [Line Length] control word. For example, if you want a line length of eight inches, specify

```
.ll 8i
```

The .LL [Line Length] control word controls the width of the running headings and footings, and footnotes. Column line length, specified with the .CL [Column Line Length] control word, defaults to the .LL value, and controls the line length of each output text column. The starting position of the rightmost column plus the column line length is the effective width of the body. This can exceed the .LL value.

You can increase and decrease the value of the line length. For example,

```
.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 line length (not the line length) limits the number of characters that can fit on an output line in that column.

Establishing Top and Bottom Margins

For most logical devices, SCRIPT/VS includes space for top and bottom margins in the page length. The amount of space is based on the logical output device type. 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.

The .TM [Top Margin] control word specifies the amount of vertical space to be left at the top of output pages. The .BM [Bottom Margin] control word specifies the amount of vertical space to be left at the bottom of output pages.

The value given with the .BM control word and with the .TM control word should not be so large that the top margin and bottom margin together fill the entire page. The size of the top and bottom margins is not affected by line spacing.

Starting a New Page

As SCRIPT/VS formats text, it keeps track of how many lines it has put 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
or
.cp
```

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 ...
```

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

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 running heading and running 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 the pages in a document 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 even page eject mode. To specify even or odd 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 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
  ```

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
.fo off
.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. If there are at least three lines, as in the example above, 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 by itself, SCRIPT/VS ejects to the next page unless there is no data on the current page.

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 in the last column) is performed when there are fewer than the required number of lines left in the column.

Page Numbers

The page number symbol is, by default, the ampersand (&), but it can be changed using the PS parameter of the .DC [Define Character] control word. The page number symbol is replaced, wherever it appears in the running heading and footing, 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 is. You can use the .PA [Page Eject] control word to reset this counter if you need to. For example,

```
.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 with

```
.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, or footing. 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 parameter is useful for printing prefaces, forewords, and front matter. 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 ends decimal page numbering and ejects the page; the next page will be number 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, and so on. 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.

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 text, symbols, macros, logical functions, iterative processing, GML tags, and control words, enabling you to format the information to fit your needs.

Running headings and footings are processed in two phases:

- Definition phase: The entire heading or footing, including all text, symbols, macros, and GML tags, is saved for later processing. No control words are processed during definition phase. Symbol substitution and GML processing are not performed during definition phase.
- Processing phase: The saved definition is a macro and is processed as such. Only control words that cause a page eject are disallowed during the processing phase.

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

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

then the running footing for each page will contain the value of the symbol &title at the time the page is started.

A simple running heading which places text in the upper left hand corner of both odd and even pages can be specified as:

```
.rh on
The Text of the Heading
.rh off
```

If you wanted to center text at the top of each page for your running heading, you could specify:

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

You can also emphasize the security classification of your document by specifying:[33]

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

which places the running heading in a bold font.

A simple running footing which places text in the lower left hand corner of both odd and even pages can be specified as:

```
.rf on
The Text of the Footing
.rf off
```

A running footing which places the current page number in the right-hand corner of each page can be entered as:

```
.rf on
.sp 2
.ri Page &
.rf off
```

The page number symbol (&) will be replaced with the current page number on each page.

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

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

---

[33] 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 20. Defining the Formatting Environment" on page 219 for details.

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.

Because running heading and footing definitions can contain text, macros and control words, sophisticated headings and footings can be created to fill special requirements. For example,[34]

```
.rh on
.bx 1 &$LL
.fo center
.bf
Expiration Date:
.pf
.us January 22nd, 1985
.bx off
.sp 2
.rh off
```

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

```
Expiration Date: January 22nd, 1985
```

Running headings and footings appear in the body of a page flush with the text. Ordinarily, some space should be 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

```
.rh on
.bx 1m &$LL
.ce Parts List
.tp 3m 24m 49m
.bx 1m 14m 47m &$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

[34] The .BX [Box] control word is described in detail under "Chapter 14. Creating Rules and Boxes" on page 157.

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

```
.rh cancel      - or -      .rf cancel
```

If 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      - or -      .rf sup
```

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

```
.rh res      - or -      .rf res
```

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

Where to Define Headings and Footings

SCRIPT/VS formats running headings and running footings for each page before processing the body text for that page. Therefore, when you redefine a running heading or footing, you should make sure that it is redefined *before* a control word that causes a new page is encountered, because ordinarily it will not take effect until the *next* output page is processed.

If you want to alter a running heading or footing after the page has started, however, you can do so as follows:

1. Redefine the current running heading or footing to whatever new values you want it to have
2. And then specify

```
.rh execute      - or -      .rf execute
```

The new running heading or running footing for this page is processed immediately; that is, it takes effect on the current page. If the heading or footing contains any variable information, the latest values for those variables is used. Your new running heading or footing definition remains in effect until you redefine it again.

If the new running heading or running footing specified with the EXECUTE parameter is larger than the original running heading or running footing definition, then the following rules apply:

- If the new definition can fit on the page, it takes effect on the current page
- If the new definition does not fit, it takes effect on the next page.

If it is necessary for SCRIPT/VS to finish processing the current page before a running heading or footing is redefined, you can specify

```
.pa nostart
```

which ends the current page but does not start the next page. You can then redefine the running heading or footing for the next page. The next page will automatically be started when text for the body of that page is formatted.

Ordinarily, running headings and footings 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.

Chapter 11. Placing Text in Named Areas

The .DA [Define Area] and .AR [Area] control words can be used to place text at predefined places on the page.

The .DA [Define Area] control word is used to define *named* areas. With this control word you can specify the following:

- Type of area: page, body or section
- Horizontal displacement of upper left-hand corner of the area
- Vertical displacement of upper left-hand corner of the area
- Width of the area
- Depth of the area
- Font to be used in the area.

Page Areas

A page area can be placed anywhere on the page, as defined by the .PL [Page Length] and .PW [Page Width] control words.[35] The horizontal and vertical displacements given for a page area are measured from the upper left-hand corner of the page. All page areas are placed on the page when the page is ended. See Figure 13 on page 135 to see how the horizontal and vertical displacement of a page area is measured on the page.

Body Areas

A body area can be placed anywhere within the body of a page. The body of the page starts after the running heading (or after the top margin, if there is no running heading) and extends to the footnote, running footing, or bottom margin. Horizontally, the body of a page begins at the left margin as defined by the BIND option of the SCRIPT command and the .PM [Page Margins] control word. The horizontal and vertical displacements given for a body area are measured from the upper left-hand corner of the body of the page. All body areas are placed on the page when the page ends. See Figure 13 on page 135 to see how the horizontal and vertical displacement of a body area is measured on the page.

---

[35] On the 3800 Printing Subsystem the top and bottom half inch of the physical form is reserved by the printer and cannot be printed on.

Section Areas

A section area is a bit different from page and body areas. Section areas are placed on the page by specifying the PUT parameter of the .AR [Area] control word. When a section area is placed, the current section is ended and a new section is begun. A section area uses the upper left-hand corner of the new section as its origin. The vertical displacement is ignored for section areas. All section areas begin at the top of the section created for them. The horizontal displacement is used, and is measured from the left margin of the section as defined with the BIND option of the SCRIPT command and the .PM [Page Margins] control word. See Figure 13 on page 135 to see how the horizontal and vertical displacement of a section area is measured on the page.

Other Considerations

The horizontal and vertical displacement parameters are positional, they must be the second and third parameters on the .DA [Define Area] control word. Any valid space unit notation can be used for these two parameters. If the vertical displacement is omitted, zero is assumed.

If the type of area is not specified, PAGE is assumed.

Specifying Width

The WIDTH parameter of the .DA control word allows you to give a specific column line length for the *named* area. If not specified, the current column line length is used. Use this parameter to ensure that all of the text formatted in the area fits within the boundaries of the page. All text in the area will be formatted as if the value given on the WIDTH parameter had been given on a .CL [Column Line Length] control word at the beginning of the area.

Specifying Depth

SCRIPT/VS always formats all of the text in a *named* area. The DEPTH parameter of the .DA control word is used to determine how much formatted text will be placed on the current page, body, or section. If omitted, the area will be filled until there is no more text in the area, or the end of the page, body, or section is reached. Any text not placed is saved and may be placed on the next page, body, or section.

If the DEPTH value specified would exceed the bottom of the page, body, or section, the area will only extend to the bottom of the page, body, or section.

Specifying a Font

The FONT parameter of the .DA control word allows you to specify that a *named* area be formatted in a particular font. The font specified may be any fontname that can be used with .BF [Begin Font]. This font will be the initial font for the *named* area. If the FONT parameter is not specified, the current font will be used.

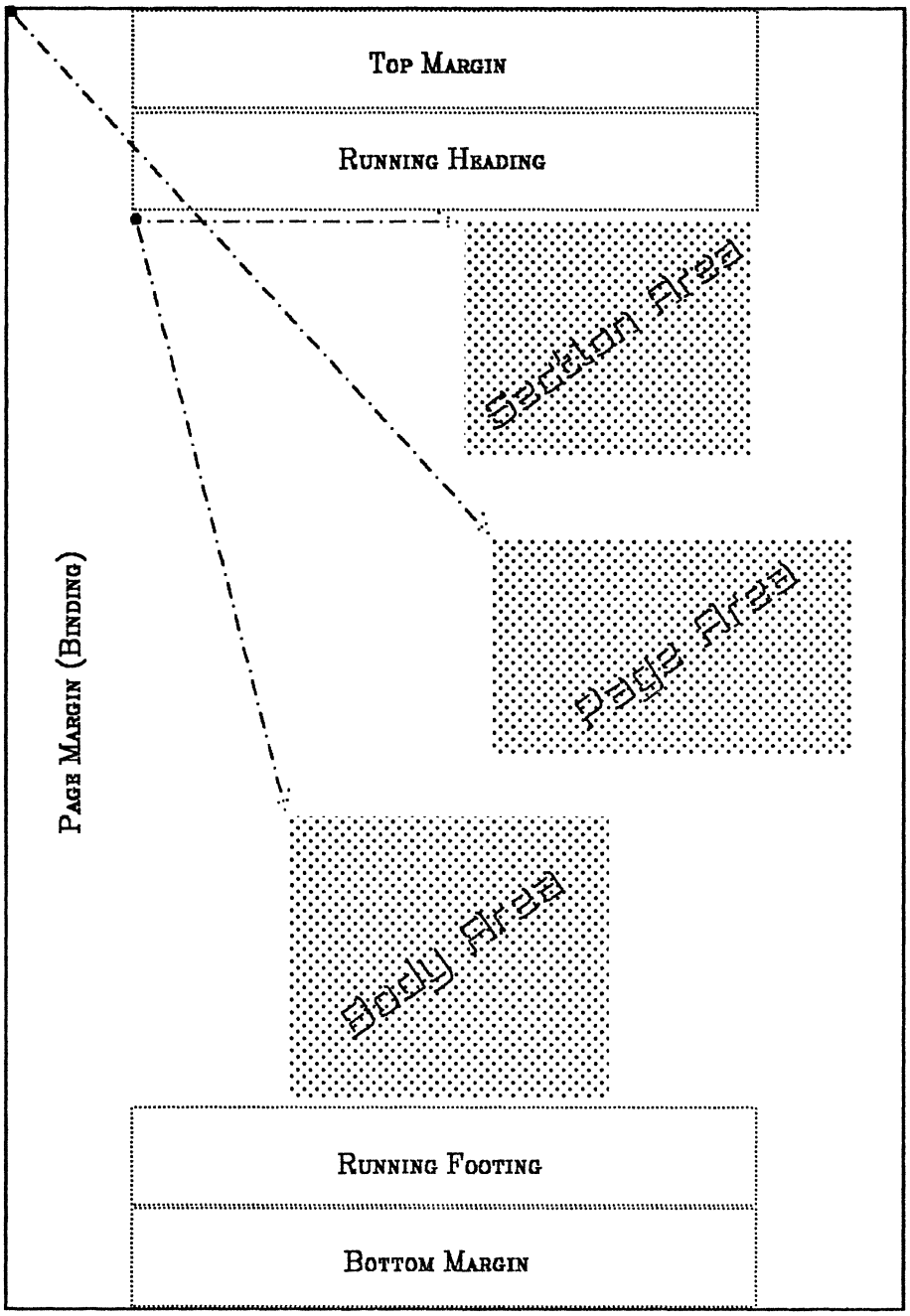


Figure 13. Measuring the Origin of Areas on a Page.

Putting Text in the Named Areas

Use the .AR [Area] control word to put text into a *named* area. The first parameter given is the name of the area that the following text is to go into. This is the name given with the .DA [Define Area] control word. For example,

```
.ar cortney on
```

causes all text, control words, GML tags and macros encountered until a .AR OFF control word to be put into the *named* area CORTNEY.

Specifying:

```
.ar cortney top
```

causes all text, control words, GML tags and macros encountered until an .AR OFF control word to be put into the area CORTNEY, ahead of anything that might already be in that area. If the area is empty, the ON and TOP parameters give the same results.[36]

Specifying:

```
.ar eric delete
```

will delete all the contents of the *named* area ERIC that have not yet been placed on a page. The *named* area is now empty.

Specifying:

```
.ar eric replace
```

will delete all the current contents of the *named* area ERIC that have not yet been placed on a page; then all text until the next .AR OFF control word is placed in the area. This parameter is equivalent to saying:

```
.ar eric delete
.ar eric on
```

All control words that are disallowed in keeps, floats and footnotes are also disallowed in *named* areas. If SCRIPT/VS encounters any of these control words while in a *named* area, the area is ended and the control word is processed. See the *Document Composition Facility: SCRIPT/VS Language Reference* for a list of these control words.

Placing the Named Area on the Page

Page and body *named* areas are placed on the page when the page is ended. Text can be added to an area several times before the area is placed on the page. Any text that does not fit on the current page, either because the DEPTH parameter was reached, or the end of the page or body was reached, will be saved and placed on the next page.

To place section areas on the page, you must explicitly specify .AR PUT. When you specify .AR PUT, the current section is ended and a new section is started. All section

---

[36] Specifying TOP does not necessarily mean that the following text will end up at the top of the area when the area is put on the page. If another .AR TOP control word is encountered for that area, the text following that control word will be put ahead of any text entered with a previous .AR TOP control word.

areas that are not empty are placed in the new section and as many sections as needed are created to place all the text of all the section areas. Also, as many pages as needed are created to place all of these sections.

Note: If you never specify .AR PUT, text formatted into section areas will never be placed on any page.

Skips that occur at the top of an area will be discarded when the area text is placed on a page. If the depth of the area is measured by use of &AD', the returned value will not reflect the presence of any top skip even though it has not yet been discarded.

A skip at the bottom of a page or body area is not discarded. If the placement of the skip (or any other object, such as a text line) would cause the depth of the area to be exceeded, the placement will be deferred until the next page. When this occurs, the short area is padded with vertical white space.

A skip at the bottom of a section area may or may not be discarded. If it will fit, the skip is kept; otherwise, it is discarded. After the section is complete, all of the section areas will be made equal in depth by padding the shorter areas with vertical white space.

Specifying Named Areas

The following .DA [Define Area] control word defines a page type area that originates at the upper left-hand corner of the page, is 3 inches wide, and is 5 inches deep:

```
.da piezie 0 0 page width 3i depth 5i
```

Since the default area type is PAGE, and the default vertical displacement is zero, the following .DA control word defines an area with the same specifications as the area PIEZIE above.

```
.da joe 0 width 3i depth 5i
```

The following .DA control word defines an area that covers the entire page:

```
.da paul 0 0 width &$PW depth &$PL
```

A body area that starts 3 centimeters in from the left margin, 2 centimeters down from the top of the body of the page, is 5 centimeters wide and extends to the end of the body would be defined as:

```
.da mick 3cm 2cm body width 5cm
```

To define two section areas to simulate a two column format with columns 26 characters

wide starting at the left margin and 30 characters from the left margin specify:

```
.da edgar 0 section width 26
.da lewis 30 section width 26
.ar edgar on
.fo left
Once upon a midnight dreary,
while I pondered, weak and weary,
Over many a quaint and curious volume of forgotten lore--
While I nodded, nearly napping,
suddenly there came a tapping,
As of some one gently rapping,
rapping at my chamber door.
"'Tis some visitor," I muttered,
"tapping at my chamber door--
Only this and nothing more."
.ar off
.ar lewis on
.fo right
The sun was shining on the sea,
Shining with all his might:
He did his very best to make
The billows smooth and bright--
And this was odd, because it was
The middle of the night.
.sk
The moon was shining sulkily,
Because she thought the sun
Had got no business to be there
After the day was done--
"It's very rude of him," she said,
"To come and spoil the fun!"
.ar off
.ar edgar top
RAVEN:
.sp
.ar off
.ar lewis top
WALRUS:
.sp
.ar off
.ar put
```

This will be formatted as:

RAVEN:

Once upon a midnight dreary, while I pondered, weak and weary, Over many a quaint and curious volume of forgotten lore-- While I nodded, nearly napping, suddenly there came a tapping, As of some one gently rapping, rapping at my chamber door. "'Tis some visitor," I muttered, "tapping at my chamber door-- Only this and nothing more."

WALRUS:

The sun was shining on the sea, Shining with all his might: He did his very best to make The billows smooth and bright-- And this was odd, because it was The middle of the night.

The moon was shining sulkily, Because she thought the sun Had got no business to be there After the day was done-- "It's very rude of him," she said, "To come and spoil the fun!"

To specify a page area that starts 4cm down from the top of the page and 4cm over from the left edge of the page, uses the default column line length, and an italic font (for page printers) specify:

```
.df italic type(futura 10 italic)
.da cleo 4cm 4cm page font italic
```

The text in the *named* area CLEO will use Futura 10 point italic as its initial font. Other font changes can be made in the area with the .BF control word.

To put an area in the upper left hand corner of the page specify:

```
.da fred 0 0 page width .5i depth 3
.ar fred on
something else...
.ar off
```

Named areas can overlap other *named* areas, and any other text that may be placed on the page. Choose vertical displacements, horizontal displacements, WIDTH values and DEPTH values, to avoid any unintentional overlapping. On devices that do not overprint, overlaying of text in this manner may produce unpredictable results.

Using the &AD' Symbol Attribute

The &AD' symbol attribute can be used to determine the depth of the unplaced text in a *named* area. This symbol attribute can be helpful in putting headings on areas each time they appear on a page. In the following example, the line "DIET continued:" will appear at the beginning of the *named* area CECIL on each page after the first that it appears on.

```
.rh on
.if &ad'cecil eq 0 .go skip
.ar cecil top
DIET continued:
.ar off
...skip
.rh off
.da cecil 0 0 body width 5i depth 3i
.ar cecil on
.fo left
.ce DIET
.sp
Some ladies smoke too much and
some ladies drink too much and some ladies pray too much,
But all ladies think that they weigh too much.
They may be as slender as a sylph or a dryad,
But just let them get on the scales
and they embark on a doleful jeremiad;
No matter how low the figure the needle happens to touch,
They always claim it is at least five pounds too much;
To the world she may appear slinky and feline,
But she inspects herself in the mirror and cries,
Oh, I look like a sea lion.
 ...
.ar off
.*
```

When the running heading is executed at the beginning of each page, the value of &AD'cecil will be the number of lines in the *named* area CECIL that have not yet been put on a page. If there is still text left in CECIL, the line "DIET continued:" will be inserted at the top of the area. When all of the text of the area has been placed on pages, the value of &AD'cecil will be zero; the condition will then be false: nothing more will be placed into the *named* area CECIL.

Using Named Areas with the 3800 Subsystem Model 1

If text or rules from the area overlay anything outside that area, misalignment of text and/or rules may occur.

Chapter 12. Composing Multiple-Column Pages

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

Note: Many of the examples in this chapter are formatted to the column line length of the page for demonstration purposes.

Defining Multicolumn Layout

You can define an output page with as many as nine columns of text. To define a multicolumn page layout, you should decide the number of columns that you want, the line length of each column, and the desired horizontal displacement for the left margin of each column.

The space between columns (the gutter) is determined by the relationship of the column line length to the column positions. Usually, the column line length 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.

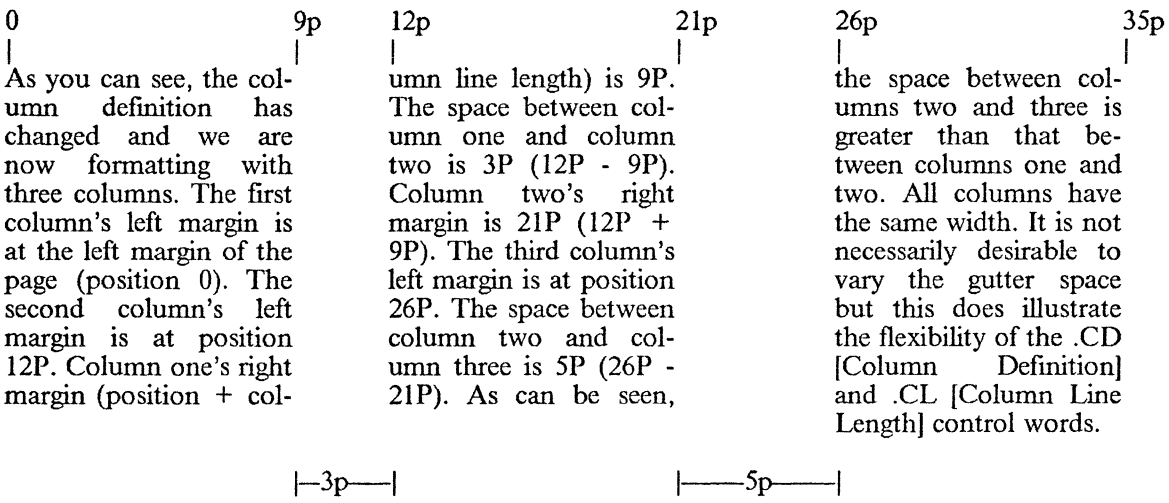
Once you have decided the dimensions and positions of your columns, the column definition can be specified using the following SCRIPT/VS control words:

- .CD [Column Definition], which provides for
  - Specifying the number of columns
  - Specifying the left margin position for each column
- .CL [Column Line Length] which provides for
  - Specifying the column line length for all columns

To define a multicolumn layout for three columns that have widths of 9P, and have left margins at the page's left margin, at 12P, and at 26P respectively, the following control words would be used:

```
.cl 9p
.cd 3 0 12p 26p
```

and would produce the following effect:



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 single column text layout, running headings and footings, 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 35p
```

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

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

Note: The .CD [Column Definition] and .CL [Column Line Length] control words take effect immediately on the next output line.

Page Sections and Section Breaks

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

Running Heading

Top Page Float

Body Text

Bottom Page Float

Footnotes

Running Footing.

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 11 on page 118 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 10. Establishing Page Layout" on page 117 for details on these component space values.

When a page is being formatted, 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. See "Chapter 9. Vertically Justifying Text" on page 111 for more details on balancing columns.

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.

Defining Columns

You can place text in a column by using a combination of the .CD [Column Definition] and .CB [Column Begin] control words.

You can use the .CD [Column Definition] control word to define a set of columns. Ordinarily, text flows from one column to the next as the columns are filled.

The .CD [Column Definition] control word causes a *section break* when it is processed. This means that all the text entered before the .CD [Column Definition] control word is processed and positioned on the page using the old definition before the new definition becomes active.

The gutter between columns is obtained by defining the column line length to a value less than the distance between column starting positions.

If you enter

```
.cd 2 0 21p
.cl 19p
```

you get two 19 pica columns with a 2 pica gutter between them.

The positions of the columns do not control how wide the columns are to be; you must set the column line length, using the .CL [Column Line Length] control word, to control this. If the current column line length is greater than the distance between columns, the text from a column can overlay the next column.

Whenever you use a .CD control word, you should specify positions for each column available. If you specify .CD n without specifying any positions and no previous column definition has been specified, the initial values 0, 46, 92, 0, 0, 0, 0, 0, and 0 are used.

You can also predefine columns without actually using them. For example, if you enter

```
.cd 2 0 10cm 20cm 30cm
.cl 8cm
```

four columns are defined, but text is formatted into only the first two of them. If you later enter

```
.cd 4
```

text is formatted into all four of the columns you defined with the earlier .CD control word. If .CD is specified with no other parameters, a section break is performed and the column definitions are unchanged.

Columns can be defined that overlay one another in whole or in part. The results may be undesirable on devices which do not allow overprinting.

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 11p 22p
.cl 9p
```

to format using only the first column.

This second section was produced by specifying

```
.cd 2
```

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

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 can 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 Line Length

Column line length remains in effect until explicitly changed by a .CL [Column Line Length] control word.

To make best use of the space on a page, column line length and column positions are usually changed when the number of columns changes. Usually the column line length value would be set to line length minus all gutter space, divided by the number of columns.

With a line length of 34P, and a gutter of 2P, two columns would be defined as:

```
.cd 2 0 18p
.cl 16p
```

This two-column data is formatted with a column line length of 16P to make the most 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 12p 24p
.cl 10p
```

This three-column data is formatted with a column line length of 10P to make the most 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 can 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 such a block will not be split across columns. see "Keeps" on page 191 for details on use of the .KP [Keep] control word.

You can use the .CB control word when you want to make subsequent text appear at the top of a new column. If the current column at the time .CB is encountered is the last column on the page, the column eject is the same as a page eject, because the next column is the first column of the next page.

The material following the .CB control word will be placed at the top of the new column, and will remain there, even if column balancing is in effect.

The .CB control word ensures that the text following it will appear at the top of a column:

```
.cb
This text will fall
at the top of a column ...
```

If a floating or delayed keep is waiting for the start of a new column, then the text that follows the .CB control word appears after the keep.

If both a top column float and a keep are waiting for the start of a new column then the top column float precedes the keep, which in turn precedes the text.

A column eject can 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 .CB control word. The other control words that can cause a column eject are:

```
.H0 - .H6 [Head Level 0 - 6]
.KP [Keep]
```

Suspending and Resuming Multicolumn Processing

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 line lengths, 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.

The .SC [Single Column Mode] control word

- Saves the current column definition
  - Column line length
  - Number of columns
  - Column positions.
- Defines a single column with a column line length equal to line length.

The .MC [Multicolumn Mode] control word restores the last-saved column definition.

You must specify the .SC [Single Column Mode] control word before you specify the .MC [Multicolumn Mode] control word.

Chapter 13. Creating Head Levels and Table of Contents

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[37] 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[38] . 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 can include such things as spacing above and below the heading, capitalization, underscoring, and font changes.
- If the heading requires a table of contents entry, the heading's text and current page number are saved in the DSMUTTOC file.

For example, if you enter a topic heading as

```
.h3 Symptoms
```

SCRIPT/VS uses the characteristics of 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.

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
```

---

[37] The word heading is used in this section to mean a topic heading that is printed as part of the text.

[38] The GML starter set provides tags with similar names and functions. This discussion is concerned only with the SCRIPT/VS control words.

The .DH [Define Head Level] control word allows you to redefine the characteristics of any head level to suit your needs. You can determine whether:

- The heading in the text should begin on a new page or cause a break.
- The heading should be placed in a separate section.
- The heading should be numbered with a decimal number associated with the head level.
- The heading should be eligible for hyphenation.
- The heading should never be hyphenated.
- The heading should or should not be formatted in hanging indent style if it occupies two or more lines.
- The heading should be right-aligned, left-aligned, centered, or aligned away from the presumed binding of the page.
- The heading should be capitalized or underscored, and which font it is to be printed in.
- Vertical space will precede and follow the heading, and how much
- 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 following:
  - 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 14 on page 151 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 space for the heading and at least two lines of text that follow it in the body of the document. If there is not enough space the heading is placed at the top of the next column. However, if the heading is defined to cause a section break, then SCRIPT/VS checks to see if there is space for the heading, plus a following line of space, plus 1/2 inch. If not, the heading is placed at the top of the next page. In this case, you should use a .CP control word prior to headings that cause section breaks but not page breaks.

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.

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 char-

acteristics, 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.

To center all level 1 headings and set them in the second font specified with the CHARS options of the SCRIPT command, enter

```
.dh 1 center font &$CHAR(2)
```

If only one font is specified with the CHARS option, the value of the symbol &$CHAR(2) will be null and the FONT keyword will be ignored if it is on the end of a line.

You might have requested a font you had previously specified with the .DF [Define Font] control word. The example above might then look like this:

```
.dh 1 center font emph
```

See "Chapter 15. Selecting Fonts" on page 175 for more details on defining and selecting fonts.

	.H0	.H1	.H2	.H3	.H4	.H5	.H6
New page for heading		yes					
Section breaks around heading		yes					
Heading alignment		out-side	left	left	left		
Space before heading	0	0	0	0	0	0	0
Skip before heading	0	0	3	3	3	1	1
Space after heading	0	5	2	2	2	0	0
Heading underscored		yes	yes		yes	yes	yes
Heading capitalized		yes	yes	yes		yes	
Break before heading		yes	yes	yes	yes		
Table of Contents entry	yes	yes	yes	yes			
Table of Contents only	yes						
Skip before T.O.C. entry		yes					
T.O.C. indention	0	0	0	2	4	6	8
Automatic hyphenation	0	yes	yes	yes	yes	yes	yes
Hanging indent	0	no	no	no	no	no	no

Figure 14. Summary of Initial Head Level Characteristics: This table lists the initial 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: By default, all headings and table of contents entries are printed in the current font and headings are subject to hyphenation.

To make level 2 headings result in exactly the same formatting as the default level 1 headings, you would enter

```
.dh 2 pa sect outside spaf 5 ts
```

your level 2 headings will then:

- Do a page eject before the head level (if not already at the top of a page).
- Cause a section break before and after the head level.
- Align the text of the heading against the outside margin of the column - away from the presumed binding edge of the duplexed page. This is equivalent to .FO OUTSIDE.
- Put five spaces after the head level.
- Space one line before a table of contents entry.

If you want to left-align (as in .FO LEFT) the text of a heading, you can enter:

```
.dh 2 left
```

If you want to right-align (as in .FO RIGHT) the text of a heading, you can enter:

```
.dh 2 right
```

If you want to align the text of the heading against the inside margin of the column - towards the presumed binding edge of the duplexed page (equivalent to .FO INSIDE) - you could enter:

```
.dh 2 inside
```

If you want to center the text of the heading, you could enter

```
.dh 2 center
```

If you do not want level 5 headings to be underscored or capitalized but you do want them to create table of contents entries, enter

```
.dh 5 nus nup tc
```

To restore the default characteristics at a later time, you can enter

```
.dh 5
```

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. 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 the 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 on, 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. Therefore, enter the text of a topic heading as it should appear in the table of contents. The specification of any hanging indent for topic headings does not also apply to the associated table of contents entry.

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 F /Sail and Rudder/ ./153/
```

When the input lines in the DSMUTTOC file are processed, the line appears in the table of contents as:

```
Sail and Rudder  . . . . . .  153
```

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 Head Three Text
```

inserts the .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 Head Three Text
```

inserts

```
.'SX F /.h3 Head Three Text/ ./154/
```

into the table of contents; the leading blanks are removed.

Printing the Table of Contents

Use the .TC [Table of Contents] control word to imbed the DSMUTTOC file. When the .TC control word is encountered, SCRIPT/VS:

- Ejects to a new page if it is not already at the top of a page.
- Prints the word CONTENTS as a level one heading 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 do not want a title at all, specify

  ```
  .tc /
  ```

  and a page eject will still be performed but no heading will be put on the page.
- 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.

In the CMS environment, the DSMUTTOC file is not deleted until the next time a new table of contents is started by another .PT control word.

In the MVS and VSE environments, unless preallocated, the DSMUTTOC file is deleted after the DCF run.

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 the discussion of the TWOPASS SCRIPT command option in the *Document Composition Facility: SCRIPT/VS Language Reference.*

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 does not 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 reserve a range of page numbers 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:

- Use the .PN [Page Numbering Mode] control word

  ```
  .pn roman
  ```

  to number table of contents pages with roman numerals

- Use the .RF [Running Footing] control word

  ```
  .rf even on
  Contents &
  .rf off
  .rf odd on
  .ri Contents &
  .rf off
  ```

  to put running footings on each table of contents page

- Use the .PA [Page Eject] control word

  ```
  .pa odd
  ```

  to ensure that the first page of the table of contents starts on an odd-numbered page.

Note: Because the .TC [Table of Contents] control word has a level one heading built into it, you should avoid redefining a head level one until after the .TC control word is processed.

Chapter 14. Creating Rules and Boxes

This chapter describes how you can create rules and boxes. It contains information about drawing:

- Horizontal and vertical rules
- Simple boxes
- Boxes with named rules
- Several types of boxes and including text within them
- Boxes with page printers.

Drawing Horizontal and Vertical Rules

You can use the .DR [Define Rule] to define rules. You can use these rules to underscore text (.UD), to create boxes (.BX), and to draw horizontal (.HR) or vertical (.VR) rules.

Defining Rules

With the .DR [Define Rule] control word you can define *named* rules of a specified weight for page printers or, for line devices, in a particular font previously specified with the CHARS option of the SCRIPT command or defined with the .DF [Define Font] control word.

For page printers, the default weight for horizontal and vertical rules, a rule called *boxrule*, is .3mm. If you want to redefine *boxrule*, and therefore change the default rule, you can do so by entering, for example,

```
.dr boxrule weight .4mm
```

Until you specify otherwise, the default horizontal and vertical rule weight then becomes and will remain .4mm.

If you want to, you can simultaneously specify rules so that whether you are printing on a line device, or on a page printers, your input is device independent. For example, if you specify

```
.dr thin weight .2mm font &$CHAR(1)
```

and then enter

```
.hr thin 1i for 2i
```

a one inch horizontal rule, .2 millimeters thick, will be printed by page printers. On line devices, this rule will be constructed of characters from the first font you requested with the SCRIPT command.

If you enter

```
.vr thin 4cm
```

a vertical rule, .2 millimeters thick, will be printed four centimeters from the left margin by page printers. On line devices, this rule will be constructed of characters from the first font you requested with the SCRIPT command.

You do not, of course, have to specify both types of rules but if your output may be directed to more than one type of device, making your rules device independent may be very useful and efficient. When SCRIPT/VS processes the rule definitions, it selects the one appropriate to the specified logical device and ignores the other.

Remember that when you request a particular font for a rule on a line device, you are implicitly requesting a box character set as well. If you wanted to specify a particular box character set, you might modify our previous example as follows:

```
.df font1 box APL font &$char(1)
.dr thin weight .2mm font font1
```

Then if you requested that the rule *thin* be printed on a line device, it would be printed in the first font you requested on the CHARS option of the SCRIPT command and it would be constructed of characters from the APL box character set.

For line devices, boxes and rules must be built with characters containing fragments of rules and rule intersections. For such devices, SCRIPT/VS assumes an appropriate box character set based on the logical device type and current font. You can override this assumption with the CHAR parameter of the .BX control word or the BOX parameter of the .DF control word, specifying any of the following box character sets:

APL APL characters
GPC 3800 GP12 font
TNC 1403 TN character set
TRM terminal character set
32A 3270 APL characters
32T 3270 text characters
38C SCRIPT/VS 3800 fonts.

Drawing Horizontal Rules

You can use the .HR [Horizontal Rule] control word to specify unnamed or *named* horizontal rules. The default for unnamed rules is a rule (named boxrule) .3mm thick for page printers, and the current font for line devices.

If, for example, you wanted a rule to be the width of the entire column, you can use the LEFT and RIGHT parameters of the .HR [Horizontal Rule] control word:

```
.hr left right
```

If you wanted a horizontal rule to print for only part of a column you could specify:

```
.hr 1i for 2i
```

or

```
.hr 1i to 4i
```

In the first case, a two-inch long horizontal rule will be drawn beginning one inch from the current left hand margin.

In the second case, a horizontal rule will be drawn beginning one inch from the current left hand margin and extending through four inches from the current margin.

You can also specify several horizontal rules with a single .HR [Horizontal Rule] control word. For example, if you specify:

```
.hr 1 for 5 10 to 15
```

then two horizontal rules will be drawn: one that starts in the first position of the current column and is five characters long and a second that begins in the tenth character position in the column and is printed up to and including the fifteenth character position.

--- ---

When defining two or more rules with one .HR [Horizontal Rule] control word, be sure that they are given in ascending order and that they do not overlap. The following examples

```
.hr 1i to 3i 2i to 4i
.hr 1i to 3i 3i to 4i
```

are *incorrect* because in each case the rules will overlap.

Also, keep in mind that there is a break before and after a .HR control word, so that you *cannot* insert a horizontal rule in the midst of text. For example, if you define a named rule, "thick," by specifying

```
.dr thick weight .8mm
```

and then enter

```
Here's some text;.hr thick 2i to 2.5i;more text following
```

you will get a break in the middle of the line and the rule will be printed on a line by itself and subsequent text (in this case the words "more text following") will be printed on the following line.

```
Here's some text
```

```
more text following
```

Using Named Horizontal Rules

If you have defined *named* rules with the .DR [Define Rule] control word, you can use them with the .HR [Horizontal Rule] control word to create rules of different weights on the same line for page printers. For example, if you had defined the rules *thin* and *thick*,

```
.dr thin weight .2mm
.dr thick weight .8mm
```

you could specify

```
.hr thick 1i to 2i thin 2.5i to 3.5i
```

to get:

If you had only specified one *named* rule in the example above,

```
.hr thick 1i to 2i 2.3i to 3.5i
```

then the second rule specification (2.3i to 3.5i) reverts back to the width of the last rule specified (in this case the rule *thick*) and you will get:

If no rulename had been given, the designated rules will be drawn using the default rule, *boxrule*.

On line devices, the font used for rules can not be changed on a given line. In other words, for a single set of horizontal rules, the first rulename specified with the .HR control word will be used for all segments of those horizontal rules and subsequent rulenames will be ignored.

Underscoring with Named Rules

If you want to explicitly position an underscore rule on a page printer, you can use the .UD [Underscore Definition] control word. For example, if you enter

```
.dr thick weight .5mm
.ud thick -p2
```

a rule is drawn two pica points below the baseline of underscored text:

a rule is drawn two pica points below the baseline of underscored text.

If the underscore rule is positioned above the normal baseline on a page printer, it may overlay text.

If, for example, you enter

```
.dr thin weight .3mm
.ud thin p2
```

a rule is drawn two pica points above the baseline and through the middle of the underscored text:

~~a rule is drawn two pica points above the baseline and through the middle of the underscored text.~~

For more details on underscoring text, see "Emphasizing Text" on page 186.

Drawing Vertical Rules

You can use the .VR [Vertical Rule] control word to specify unnamed or *named* vertical rules. A simple vertical rule can be drawn anywhere in a column. For example, if you specify

```
.vr 10
.sp 3
.vr off
```

a vertical rule will be drawn starting in the tenth character position of the current column and the rule will be three lines long. In this example, note that any vertical space unit could have been used and that you must end the vertical rule by specifying .VR OFF.

If you want a vertical rule to be flush left or flush right in a column, you can use the LEFT and RIGHT parameters of the .VR control word

```
.vr left
.sp 3
.vr off
```


or

```
.vr right
.sp 3
.vr off
```


respectively.

Using Named Vertical Rules

If you have defined a *named* rule, you can use it with the .VR [Vertical Rule] control word. For example, *named* rules defined as

```
.dr thin weight .3mm
.dr thick weight .6mm
```

for page printers can be used to create vertical rules. The following control word sequence, then,

```
.vr thin 15m thick 20m thin 25m
.sp 2
.vr 20m off
.sp 2
.vr off
```

results in:

Note from the above example that you can specify several vertical rules with one .VR [Vertical Rule] control word. You can also turn off vertical rules independently as shown in this example.

If you had not specified a particular *named* rule for every vertical rule, for example,

```
.vr thick 15m thin 20m 25m
```

then the second and third rules would both be drawn using the *thin* rule because rules without specific designations will default to the previously specified *named* rule (as in this example) or to the default font if no *named* rules are specified:

On line devices, the font used for rules can not be changed on a given line. In other words, for a single set of vertical rules, the first rulename specified with the .VR control word will be used for all of the vertical rules and subsequent rulenames will be ignored.

Vertical rules created with the .VR [Vertical Rule] control word which cross columns or pages will extend to the bottom of the section or page unless explicitly ended by a .VR OFF control word.

Aligning Vertical Rules

On page printers, you can also align vertical rules in various ways relative to a given horizontal position. For example, if you had defined a vertical rule as follows:

```
.dr thick weight 1mm
```

and you want the left edge of this vertical rule to align with a particular horizontal position, you would specify

```
.vr thick 20m lalign
.sp 3
.vr off
```

which results in:

This is the default alignment.

Note: In the previous example, and in the two following examples, the bullet above the vertical rule is used only as point of reference to more clearly show the alignment of the rule to the designated horizontal position.

In a similar manner, if you want the right edge of your vertical rule to align with a particular horizontal position, you would specify

```
.vr thick 20m ralign
.sp 3
.vr off
```

which results in:

To center your vertical rule at the given horizontal position you would specify

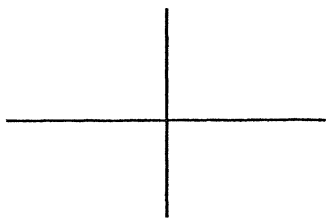
```
.vr thick 20m center
.sp 3
.vr off
```

which results in:

You can combine vertical rules with horizontal rules. To create an axis-like figure, you could specify

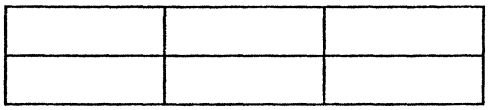
```
.vr 15m
.sp 2
.hr 5m for 20m
.sp 2
.vr off
```

which will produce the following figure:



Drawing Boxes

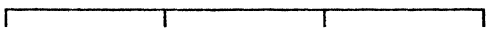
SCRIPT/VS can draw boxes around illustrations or text and can format charts with horizontal and vertical lines. Boxes drawn for page printers are formatted with horizontal and vertical rules. The control word that draws boxes and lines within boxes is the .BX [Box] control word. The three steps below define a box that would look like this:



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
   ```

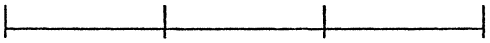
   This formats and prints a box top, with upper corners and descenders:

2. Each time you want a horizontal line within the box, specify the .BX [Box] control word with no other parameters:

   ```
   .bx
   ```

   results in

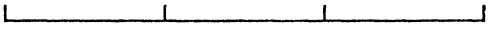
   

   The lines are drawn with intersections at the vertical rule character positions.

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 on the output line[39].

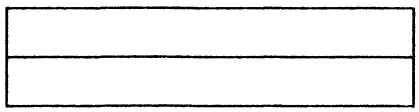
Creating Simple Boxes

Typically, a simple, basic box can be drawn as follows:

---

[39] The box may be considered to be overlaid on the formatted text. On some devices, like the 1403 printer, the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer vertical rules will cover up text characters which fall beneath them. On certain other devices, like the 3270 Display Station and the 3800 Printing Subsystem, the rule replaces the text characters.

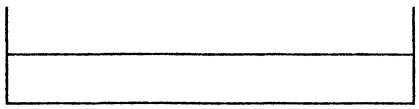
```
.bx 1 30
.sp
.bx
.sp
.bx off
```



If you did not want the initial horizontal line in your box, you could specify

```
.bx set 1 30
.sp
.bx
.sp
.bx off
```

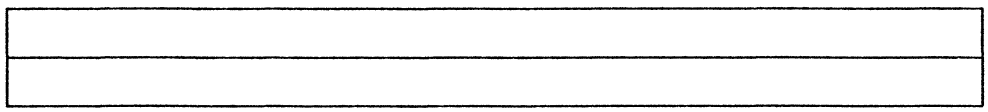
and your box would look like this:



If you want the box to extend horizontally from one side of the column to the other, you can simply specify

```
.bx left right
.sp
.bx
.sp
.bx off
```

and the box would look like this:



The box will be as wide as the currently defined column without you having to know exactly or guess the dimensions of that column.

Drawing Boxes with Named Rules

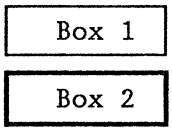
You can use *named* rules defined with the .DR [Define Rule] control word to draw boxes with rules of different weights. If you had defined rules *thin* and *thick* as follows

```
.dr thin weight .3mm font &$char(1)
.dr thick weight .6mm font &$char(2)
```

then you could specify

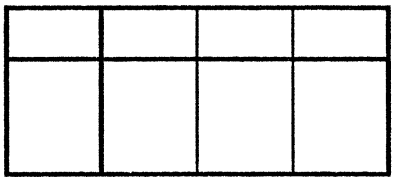
```
.ti ¬ 05
.tp 18m
.bx thin 15m 25m
¬Box 1
.bx off
.bx thick 15m 25m
¬Box 2
.bx off
```

and get the following two boxes:



On page printers, a single box can be drawn with rules of different weights. For example,

```
.bx thick 3p 6p thin 9p 12p thick 15p
.sp
.bx
.sp 3
.bx off
```



Note that in the above example:

- The first name given (*thick*) is the rule used for the horizontal rule
- The next two values (3p and 6p) are vertical starting positions for the box and they will be drawn with the *thick* rule designated
- The next name (*thin*) applies to the vertical rules beginning in positions 9p and 12p
- The last name (*thick*) applies to the last vertical rule that starts in position 15p.

On line devices, the font used for rules cannot be changed on a given line. In other words, for a single set of horizontal or vertical rules, the first rulename specified with the .BX control word will be used for all segments of those horizontal or vertical rules and subsequent rulenames will be ignored.

A Three Column Box

You can use the .BX [Box] control word to build a three-column table and use tabs to align text within the rules:

```
.ti ¬ 05
.tp 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.
		...

Centering Text within a Box

SCRIPT/VS constructs the corners and rules of boxes using 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 35m
.ce on
These lines
are centered within
this
lovely box.
.ce off
.bx off
```

when formatted for a terminal may appear as:

```
+---+------------------+---+
|   |    These lines   |   |
|   |are centered within|   |
|   |        this      |   |
|   |    lovely box.   |   |
+---+------------------+---+
```

However, when the same input lines are formatted for the 3800 Printing Subsystem, 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 by using the:

- CHAR parameter of the .BX control word
- BOX parameter of the .DF control word
- .DR control word.

(See "Defining Fonts" on page 178.)

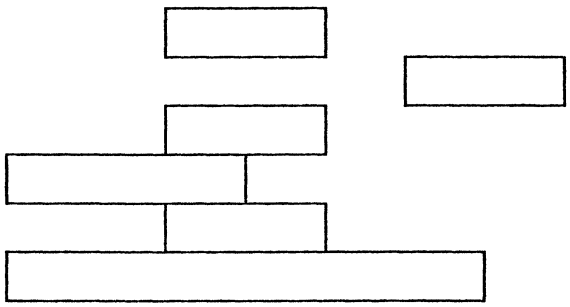
You can use SCRIPT/VS to produce many different box configurations, horizontal lines, and graphic structures. Some of the ways you can use the .BX [Box] control word are described below.

Stacking One Box on Another

You can stack several boxes by defining one box and then defining larger or smaller boxes, and you can do so without ending the definition of the first box. If you then define a box that is not contiguous with the first one, the first box is ended and the top of the second box is printed on the same line as the bottom of the first box. You can use these techniques to create a complex structure of boxes. For example, the lines

```
.bx 10m 20m
.sp
.bx 25m 35m
.sp
.bx 10m 20m
.sp
.bx 1m 15m
.sp
.bx 10m 20m
.sp
.bx 1m 30m
.sp
.bx off
```

result in:



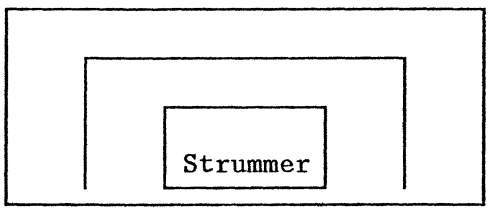
Drawing a Box within a Box

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 Strummer
.bx off
.bx can
.bx off
```

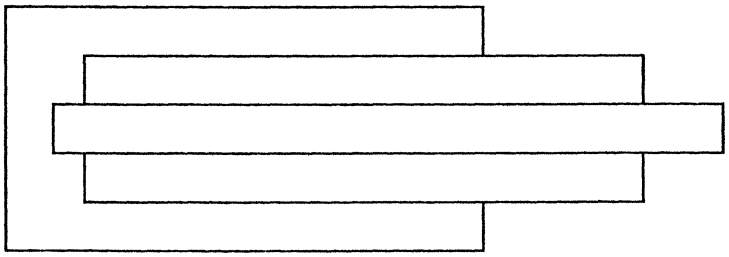
results in



When boxes are nested, 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 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
```

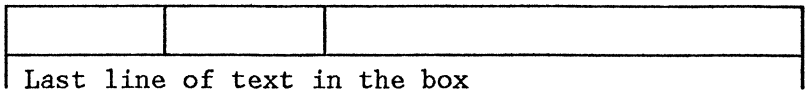
The slash indicates a discontinuity with no horizontal connection. These lines result in:

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
.in +2
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

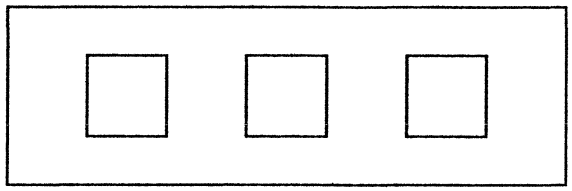
		First item in the box
		Second item in the box
		Third and subsequent items in the box....

Drawing the Middle Portion of a Box within Another (Larger) Box

You can draw a series of boxes 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 35m
.sp
.bx new 5m  10m / 15m  20m / 25m  30m
.sp 2
.bx off
.sp
.bx off
```

results in

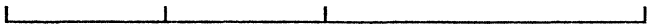


Drawing the Bottom Line (Only) of a Box

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 *and* you include the OFF parameter. For example,

```
.bx off 1m 10m 20m 40m
```

results in



Drawing Boxes with the 3800 Printing Subsystem Model 1

Special considerations apply to boxes when the output is being formatted for a 3800 Printing Subsystem. Because SCRIPT/VS does not provide three widths of each box character in each font, SCRIPT/VS performs monospace justification of text 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.
- When a vertical rule is overlaid on a text character, the rule replaces the character.
- Only monospace[40] fonts can be used within a box and all fonts used must be of the same pitch.

When using boxes and rules in a named area, if the boxes or rules overlay text from outside that area, misalignment may occur. Likewise, if text from an area is overlayed by rules or boxes from outside that area, misalignment may occur.

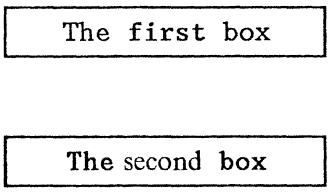
You can produce boxes of different line thicknesses containing text in several fonts. For example,

---

[40] All of the fonts distributed with SCRIPT/VS are monospace, with the exception of GP12.

```
.bx 1m 15m
.in +3
The
.bf GB12
first
.pf
box
.bx off
.sp 2
.bf GB12
.bx 1m 15m
The
.bf GT12
second
.pf
box
.bx off
.pf
```

results in:

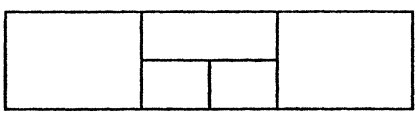


Boxes with a Different Top and Bottom

When you want SCRIPT/VS to draw a box having a different top and bottom, you can change the specifications with the CAN and SET parameters of the .BX [Box] control word. If a box is currently going, and a .BX control word with horizontal displacements and slashes is encountered, the previous box is ended with a box bottom which is the mirror image of the previous box top. The CAN and SET parameters redefine the box. When the box is ended by .BOX OFF or by another .BX control word, the box is ended with the horizontal rule drawn as specified on the .BX SET command. For example:

```
.bx 5 15 25 35
.sp 1
.bx can
.bx set 5 / 15 25 / 35
.bx 5 / 15 20 25 / 35
.sp 1
.bx can
.bx set 5 15 20 25 35
.bx off
```

results in:



Chapter 15. Selecting Fonts

With the Document Composition Facility you can take advantage of font capabilities available with various printers. On typewriter-like terminals you can stop the printing while you change typing elements. On line printers such as the 1403 you can specify underscoring, capitalization, and create boldface type by overstriking. Line printers such as the 3800 Printing Subsystem also allow actual font changes. For a page printer, you can use any font in its font library.

For the 3800 Printing Subsystem Model 1, you can select two fonts that you want to use by specifying them with the CHARS option of the SCRIPT command.

With page printers, you can select more than just two fonts. You can specify coded fonts with the CHARS option of the SCRIPT command, but entire families of fonts may be available for use with page printers as well. The font library contains these font families and you can access them with the .DF [Define Font] control word.

On page printers, you can request a particular coded font with all its defaults or you can use the TYPEFACE and CODEPAGE parameters of the .DF control word to specify parts of or variations on a particular font as well.

The latter specification is possible because page printer fonts consist of a code page (which contains the hexadecimal representation of a character in a given national language) and a font object (which is the representation of the character itself) both of which are accessible with the .DF [Define Font] control word.

Note: The Document Composition Facility (DCF) requires the following font program products be installed for the 4250 printer:

- 5771-AAR Monotype Times New Roman
- 5771-AAW Typewriter and Pi

and DCF requires the following font program products be installed for the 3800 Printing Subsystem Model 3, and the 3820 Page Printer:

- 5771-ABA Sonoran Serif
- 5771-ABC Pi and Specials.

You may tailor DCF and/or use the CHARS option of the SCRIPT command to point to typeface families other than the required ones listed here.

Selecting Initial or Default Fonts

When formatting a document you can take advantage of the printer's dynamic font storage and use different fonts in your document. You can use the CHARS option of the SCRIPT command to specify the fonts you want to use.

The CHARS option is specified as:

```
CHARS (font1 ... )
```

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

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 3800 Printing Subsystem Model 1, you can specify as many as four uppercase-only fonts, or two upper- and lowercase fonts. The CHARS JCL parameter must specify the corresponding character arrangement tables in the same sequence as the fonts specified with the CHARS option of the SCRIPT command.

Refer to the discussion of the PRINT option of the SCRIPT command in the *Document Composition Facility: SCRIPT/VS Language Reference* for details on printing documents formatted for the 3800 Printing Subsystem under TSO.

Fonts selected with the .DF [Define Font] control word for printing on the 3800 Printing Subsystem Model 1 are restricted to those fonts that you have specified with the CHARS option.

When formatting for page printers, there is no limit to the number of fonts you can specify with the CHARS option but you will most likely specify your fonts with the .DF [Define Font] control word instead. See "Defining Fonts for Page Devices" on page 179 for details on how to use the FONTLIB option.

Using Fonts

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

In addition to the upper-case only fonts, SCRIPT/VS provides sixteen complete upper- and lowercase fonts. You can also create your own fonts to use with SCRIPT/VS as long as the characteristics of these fonts are listed in a font table. (See the section on Device and Font Table Maintenance in the *Document Composition Facility: SCRIPT/VS Language Reference* for details on how to add a new font's characteristics to a font table.)

The IBM 3800 Printing Subsystem line device 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 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, 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.

With page printers the number of fonts you can specify with CHARS is unlimited but you must specify coded fonts and these fonts must be in the font library. You can also specify or describe any font in the library with the .DF control word.

More than one font can be identified with the .BF control word. The first font given which has been defined with the .DF control word or specified with the CHARS option

of the SCRIPT command is taken as the new font.[41] An error occurs only if none of the fonts given is valid.

Use the .BF [Begin Font] control word to change the current font to any font specified with the CHARS option. For example,

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

produces 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. As many as 16 fonts can be saved. Because the font stack is in the current environment, it can be affected by the .SA and .RE control words and any other control words that save and restore the environment.

You can use the .BF [Begin Font] control word to start any font that is either defined with the .DF [Define Font] control word, or listed in the CHARS option the SCRIPT command. If more than one font is specified with the .BF control word, the first valid font is used.

To eliminate dependence in the file on specific font names, use the SCRIPT/VS symbols &$CHAR(n) or the .DF control word instead of actual font names. For line devices, the previous example could be revised as:

```
This is a
.bf &$CHAR(2)
bold
.pf
word.
```

which prints as:

```
This is a bold word.
```

For page printers, the previous example could be revised as:

```
.df bold type(bold)
This is a
.bf bold
bold
.pf
word.
```

which prints as:

```
This is a bold word.
```

---

[41] For page printers, the font must be in the font library to be valid.

All SCRIPT/VS 3800 Printing Subsystem 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.

Defining Fonts

SCRIPT/VS extends the concept of fonts to include underscoring and capitalization on *all devices*, overstriking on impact printers, and stopping to change typing elements on typewriter terminals.

You can use the .DF [Define Font] control word to define *named* fonts for use with the .BF [Begin Font]. This allows you to alter the characteristics of the fonts specified with the CHARS option of the SCRIPT command and provides a means of identifying fonts descriptively. 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 Printing Subsystem Model 1, 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.

Defining Fonts for Impact Printers

When formatting for an impact printer, such as the 1403 printer, you can create boldface headings and emphasize important phrases by overstriking. You can define a named font using the .DF control word, specifying the OS parameter to indicate that the font is to be formed by overstriking the text four times:

```
.df boldface os rpt 4
```

To define a new font for 1403 output which causes capitalization and overstriking, specify

```
.df bold up os rpt 3
```

You can emphasize phrases by changing to a 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.[42]

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 stops typing to allow you to change elements. See "Interactive SCRIPT/VS Processing" on page 61 for a discussion of the use of the STOP parameter of the .DF [Define Font] control word.

Defining Fonts for Page Devices

For page printers a number of fonts may be available to you. These fonts are stored in a font library. Any font you request must be in this font library or SCRIPT/VS will not recognize it as a valid font. In order to properly select fonts, you need to know which fonts are available.

The font library consists of members or objects. In MVS, an object can be an actual member of a partitioned data set. In CMS, this object is simply a file whose filetype matches the name of the library. There are four types of objects in the font library:

Font Which provides both global font and individual character descriptive information.

Code page Which associates character names with code points.

Coded font Which is a combination of both a font and a code page. SCRIPT/VS requires both a font and a code page for formatting. A coded font is also one that is fully defined in terms of typeface, point-size, weight, width, attribute, and code page. Coded fonts are listed in the font library.

DCFINDEX Which contains one logical record for each set of page printer font objects in the font library that have a common typeface name.

Describing a Font

Most simply stated, a font is a set of characters in one typeface (such as Monotype Times New Roman) and one pointsize (such as 10 point). These two aspects of a font are described below.

---

[42] Underscoring of blanks is controlled by the .UD [Underscore Definition] control word; overstriking, even with the underscore character, affects only nonblank characters.

Typeface

A typeface is a specific set of style variations in one typeface family (such as Futura or Monotype Times New Roman). The major style variations are:

Posture The two most common typeface postures are roman, sometimes referred to as upright, and italic, sometimes referred to as cursive. Note that the roman posture is not to be confused with the typeface family Monotype Times New Roman.

Weight Weight is the variation in the width of the individual strokes of characters in a font that makes them appear to be bolder or lighter when they are printed. The common weights are light, medium, semibold, and bold.

Width Width is the variation in the width of characters that makes them appear to be narrower or wider when they are printed. The common widths are condensed, normal, and expanded. Width, as used here, does not refer to the width of individual characters.

Pointsize

The pointsize of a font (a point is 1/72 of an inch) refers to the height of the rectangle within which the largest character would fit. Of course, within a given font, the height of the individual characters vary in size, as do the ones you are now reading. IBM page printer fonts vary in size from 6 points to 72 points.

Code Pages

What is a code page? When you enter a character into a file, that character is stored as a hexidecimal code point in your file. The relationship between the hexidecimal code point values in your file and the actual character produced when the file is printed is defined by a code page.

IBM supplies code pages for the major IBM language groups. For example, there is a code page for French and a different code page for Spanish. These code pages are related to the national language keyboards that IBM supports. You should use the code page that most closely matches the keyboard that you are using, unless you are using a non-English hyphenation dictionary, in which case you should use the international codepage.

The code pages for use with the Pi and Light Italic fonts are a different kind of code page. These codepages have nothing to do with language groups or different kinds of keyboards. These fonts have a special set of characters, and thus need special code pages to use in relating those characters to specific hexidecimal code points.

Coded Fonts

A coded font is simply a member of the font library that relates a specific code page to a specific font. For each typeface family supplied by IBM, there is a coded font for each combination of code page and the 10-point font in that typeface family. As you will find out in a later section, DCF lets you define fonts for use without specifying a code page. This means it is not necessary to have a coded font for each combination of code page and font.

The Default Coded Font

When you specify the SCRIPT command option, DEVICE(4250x),[43] SCRIPT/VS selects an initial coded font to use in composing your document. If you do not use the CHARS option of the SCRIPT command, SCRIPT/VS will use a default coded font (AFTTR395). This coded font associates the U.S./Canada (English) code page with the Monotype Times New Roman 10 point, medium weight, normal width, roman posture font. The default coded font can be changed for your installation by changing the SCRIPT/VS logical device table as explained in Appendix B of the *Document Composition Facility: SCRIPT/VS Language Reference.*

You can tell SCRIPT/VS to select a different initial coded font by specifying a coded font with the CHARS option of the SCRIPT command. For example, specifying

```
CHARS (AFTFT383)
```

results in Futura, 10 point, medium weight, normal width, roman posture being used with the Belgium code page.

What Is in the Font Library?

The font library contains the fonts, code pages, and coded fonts for all of the typeface families that are available for your use.

There is a convenient way to find out what is available in your font library.

The DCF Font Library Index Program Report lists the contents of the font library.[44] The fonts are listed by typeface family name (such as Futura). The font characteristics listed here are:

Font identifier The font identifier is the name by which a font is filed in the font library. You will never need to use the font identifier when working with fonts.

Pointsize Pointsize is the height of the rectangle within which the largest character of a font would fit.

Weight Weight is the variation in width of individual strokes of of characters in a font that makes them appear to be bolder or lighter when they are printed.

Width Width is the variation in the width of characters in a font that makes them appear to be narrower or wider when they are printed.

Attribute Attribute is the heading to look under for the posture of the font. Italic, of course, means italic. The absence of italic means roman or upright.

Line space Line space refers to the vertical distance, in pels, between baselines when formatting in the font.

---

[43] See the SCRIPT/VS logical device table in the *Document Composition Facility: SCRIPT/VS Language Reference* for the various specifications of 4250x, depending on page size.

[44] The actual format of the listing is subject to change.

Figure space Figure space is equal to the width, in pels, of the number 0 in the font.

Word space Word space is the size of the horizontal space, in pels, to be used between words.

The code pages and coded fonts are also listed in the Font Library Index Report.

Specifying the Font Library

The FONTLIB option of the SCRIPT command is used to specify where the font library exists.

Normally, you do not need to specify this option, as SCRIPT/VS knows the name of the default font library to use if the FONTLIB option is not specified.

If your installation does not use the defaults, ask your systems programmer how to identify your library. You can find more information about the Font Library Index Program in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Defining Fonts by Characteristics

In addition to the methods of defining fonts already described, with page printers you can define fonts in even greater detail. Using the TYPE or CODEPAGE parameters of the .DF [Define Font] control word, you can describe a font by typeface, point-size, weight, width, and an attribute identifier. You can also specify a code page name. A code page contains the particular hexadecimal coding for each character of a font in a particular language.

Page devices can take advantage of large families of fonts - as long as they are defined in the font library - and these devices also give you greater flexibility in altering the parameters of any properly defined font.

When formatting for page printers, use the FONTLIB option of the SCRIPT command to identify the host system font library containing the fonts to be used.

In CMS, the FONTLIB is specified as one of the following:

```
FONTLIB ( filetype )
FONTLIB ( filemode )
FONTLIB ( filetype filemode )
```

The default is FONT4250 for the 4250 printer, FONT38PP for the 3800 Printing Subsystem Model 3 and FONT3820 for the 3820 Page Printer.

In TSO, each font description resides in a member of a partitioned data set. The FONTLIB option is specified as:

```
FONTLIB ( dsname )
```

The default is SYS1.FONT4250 for the 4250 printer, SYS1.FONT38PP for the 3800 Printing Subsystem Model 3, and SYS1.FONT3820 for the 3820 Page Printer.

In ATMS-III, font definitions reside in a host system font library. The FONTLIB option is specified as:

```
FONTLIB ( dsname )
```

The default is FONT4250 for the 4250 printer, FONT38PP for the 3800 Printing Subsystem Model 3 and FONT3820 for the 3820 Page Printer.

In batch MVS, font definitions reside in a host system font library. The FONTLIB option is specified as:

```
FONTLIB ( ddname )
```

ddname identifies a DD statement which gives the name of the host system font library.

The default is FONT4250 for the 4250 printer, FONT38PP for the 3800 Printing Subsystem Model 3, and FONT3820 for the 3820 Page Printer.

In batch VSE, font definitions reside in a host system font library. The FONTLIB option is specified as:

```
FONTLIB ( dlblname )
```

dlblname identifies a DLBL statement which gives the name of the host system font library.

The default is FNT4250 for the 4250 printer and FNT3820 for the 3820 Page Printer. The batch VSE environment is not supported for the 3800 Printing Subsystem Model 3.

In all environments, it is the user's responsibility to ensure that the font library used in printing a document is the same one used during formatting.

When formatting for page printers, you can define a *named* font by describing it. Although not all of the following parameters are necessarily available with any given font, you can specify a typeface (the style of the font, such as Monotype Bodoni[45]), point-size (the vertical height of the characters in the font, such as 6-, 8-, 10-point and so on), weight, width, attribute and code page.

Keep in mind that the font definitions described in the following examples show a hypothetical font description. In most cases, only some of the descriptive parameters will be available for any given font that you may want to define. Check the font library index listing for specific fonts and the particular parameter combinations that are available at your installation.

If, for example, you want to specify a particular typeface you can enter[46]

```
.df body type('monotype bodoni')
```

and when you begin the font, *body*,

```
.bf body
```

all subsequent text is set in Monotype Bodoni type provided it is available. Notice that the typeface name, Monotype Bodoni, was enclosed in quotation marks. These are required around the typeface name if it contains any blanks or parentheses. Because it was not specified on the .DF control word, the point-size is the same as the font that was in effect when we started our new font. Because no other parameters were specified, the weight is medium, the width and attributes are normal, and the code page is the same as the code page currently being used.

---

[45] Trademarks of The Monotype Corporation, Limited.

[46] This example and following examples use Monotype Bodoni as an example only. Monotype Bodoni is only availabe on the 4250 printer.

If you want to specify a point size for your font, you can enter

```
.df body type('monotype bodoni' 10)
.bf body
```

and all following type is printed in 10 point Monotype Bodoni.

If you had not specified the typeface, as in

```
.df body type(10)
.bf body
```

then subsequent text will be 10 point but it will be in the typeface of the previously specified font. Because no other parameters were specified, the width and attributes are normal, weight is medium, and the code page is the same as the code page currently being used.

Weight can be specified as:

- Ultralight
- Extralight
- Light
- Medium (the default if weight is not specified)
- Semibold
- Bold
- Extrabold
- Ultrabold.

If you want to specify a weight for your font, you can enter

```
.df body type('monotype bodoni' 10 semibold)
.bf body
```

and all following type is printed in 10 point Monotype Bodoni with a weight of semibold. Because no other parameters were specified, the width and attributes are normal, and the code page is the same as the code page currently being used.

Width can be specified as:

- Ultracondensed
- Extracondensed
- Condensed
- Semicondensed
- Normal (the default if width is not specified)
- Semiexpanded
- Expanded
- Extraexpanded
- Ultraexpanded.

If you want to add a width specification to your font, you can enter

```
.df body type('monotype bodoni' 10 semibold condensed)
.bf body
```

and all following type is printed in 10 point Monotype Bodoni with a weight of semibold and a condensed width. Because no other parameters were specified, the attribute is normal and the code page is the same as the code page currently being used.

If you want to add an attribute to your font description, that is, ask for your font to be italic, underscored, or outlined, you can enter

```
.df body type('monotype bodoni' 10 semibold condensed italic)
.bf body
```

and all following type is printed in 10 point Monotype Bodoni with a weight of semibold, a condensed width and in italics. Because no other parameters were specified, the code page is the same as the code page currently being used.

If, for another example, you want text printed in 10 point Futura italic and you want the characters printed as they would appear in different languages, specify a code page with the .DF control word.

If you specify

```
.df body type(futura 10 italic) codepage aftc0395
.bf body
```

your text will be printed in 10 point Futura italic, but the font library is also searched for the code page, AFTC0395 (which is the code page that contains the hexadecimal codes for U.S. American and Canadian English characters) in order to select the characters appropriate to the language you specified.

A convenient method of using type defined fonts is as follows. First, you could specify

```
.dm font on
.df font type(&*.)
.bf font =
.dm off
```

then, for example, you could enter

```
.font ''monotype garamond' 18 italic'
```

in order to define and begin the particular font you want.

Selecting Fonts for a Variety of Devices

If a document is formatted for a variety of devices, the fonts available may vary according to the device. When you specify a .BF control word in a document, you can either provide font definitions that are based on the device type or you can provide a list of fonts in the order of your preference. These techniques are especially useful if you are creating a document to print on different printers — or if, when you create the document, you might not know which device it will be printed on.

For example, you can specify

```
.df hi2 us
.if &$PDEV eq 1403 .df hi2 os rpt 3
.if &$PDEV eq 3800 .df hi2 font &$CHAR(2)
.if &$PDEV eq 3820 .df hi2 type(italic)
.if &$PDEV eq 4250 .df hi2 type(italic)
```

and then specify

```
.bf hi2
```

in our example, if the document is printed on a 1403 impact line printer, the printing is underlined by overstriking three times.

If the document is printed on a 3800 Printing Subsystem Model 1, the printing is underlined as a result of requesting the second font specified with the CHARS option, which, in our example, is an underlined font.

If the document is printed on a page printer, the printing is italicized as a result of requesting the current font in italics.

You may not know exactly which fonts will be available when a document is created. For example, you may prepare a document to be formatted for a 3800 Printing Subsystem without knowing what fonts will be used. If you want to ensure that a piece of text is set in a bold font, you can enter

```
.bf gb10 gb12 sb12
```

Subsequent text will be formatted in the GB10 font, if it was specified with the CHARS option of the SCRIPT command. If not, GB12 will be used if it was specified, and so on.

If you want a font change to apply only for a particular device and to be unused the rest of the time, you could specify

```
.if &$PDEV eq 4250
.th .df figfnt type ('prestige elite')
```

Then if you specified

```
.bf figfnt =
```

the *figfnt* font (in this case, prestige elite), will only be used when you are formatting output page printer.

The equals sign (=) in our example is required to restart the current font after it has concluded using the figfnt font. In this case a font change was desired only for the 4250 printer, all other devices should not change fonts. So, there is no point in defining the font for each possible device.

Emphasizing Text

You can emphasize text several different ways. Some methods of emphasizing a word or phrase are: uppercase, underscore, change of type weight (such as bold), italic, and intercharacter space.

Underscoring and Capitalization

Because underscoring on line devices requires backspacing and overstriking characters, the procedure can be particularly frustrating when you need to create a line that contains an underscored word or words. Instead of manually keying in the character/backspace/underline sequence, you can use either the .US [Underscore] control word or a combination of the .DF [Define Font] and .BF [Begin Font] control words to have a word or phrase underscored when it is printed.

For example,

```
.us This is very important.
```

prints as:[47]

This is very important.

You could also have entered

```
.df hi1 us
.bf hi1
This is very important.
.pf
```

and obtained the same result.

Because the .US [Underscore] control word does not cause a break, you can specify:

```
This line contains a very
.us important
concept for consideration.
```

and it results in:

This line contains a very important concept for consideration.

The .UP [Uppercase] allows you to capitalize text and the .UC [Underscore and Capitalize] control word allows you to both capitalize and underscore your text. Both of these functions can also be specified with combinations of the .DF [Define Font] and .BF [Begin Font] control words.

For example,

```
.up Chapter 10
                              - or -
.df hi2 up
.bf hi2
Chapter 10
.pf
```

---

[47] 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.

result in:

```
CHAPTER 10
```

Use the .UC [Underscore and Capitalize] control word or a combination of the .DF [Define Font] and .BF [Begin Font] control words when you want to both underscore and capitalize a line. For example, the lines:

```
.uc preface
                    - or -
.df hi3 uc
.bf hi3
preface
.pf
```

result in:

PREFACE

You can also affect a number of input lines with the .US [Underscore], .UP [Uppercase], and .UC [Underscore and Capitalize] and with the .DF [Define Font] and .BF [Begin Font] control words. For example, to underscore three input lines you would enter:

```
.us 3
Do not
destroy this letter
until
its expiration date,
which is January 22nd, 1985.
                                  - or -
.df hi1 us
.bf hi1
Do not
destroy this letter
until
.pf
its expiration date,
which is January 22nd, 1985.
```

both of which result in:

Do not destroy this letter
until its expiration date,
which is January 22nd, 1985.

Use the ON and OFF parameters of the .US [Underscore], .UD [Underscore Definition], .UC [Underscore and Capitalize] 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.
```

The same results could have been obtained if, using the fonts described in our examples above, we entered:

```
This is capitalized for
.bf hi2
emphasis
.pf
and
.bf hi3
emotional
.pf
impact.
```

You can use the .UD [Underscore Definition] control word to determine how automatic underscoring with the .US [Underscore] and .UC [Underscore and Capitalize] control words should be performed. You can indicate whether or not blanks are to be underscored and, on page printers, which named rule is to be used for underscoring and where it is to be located with respect to the baseline.

Because word spaces are initially underscored, you must specify the OFF parameter of the .UD [Underscore Definition] control word if you wish to turn off wordspace underscoring.

For example, when you underscore text by entering

```
.us on
```

all characters, including wordspaces, are underscored. But if you have entered

```
.ud off
```

wordspaces will not be underscored. Nonblank characters are always underscored, but tab expansions and spaces specified with the TO parameter of the .IS control word are never underscored.

You can also use the .UD [Underscore Definition] control word to explicitly position the underscore rule on page printers. For example, if you enter

```
.dr thud weight .6mm
.ud thud -p2
```

a rule .6mm thick is drawn two pica points below the baseline to underscore text:

a rule .6mm thick is drawn two pica points below the baseline to underscore text.

If the underscore rule is positioned above the normal baseline on page printers, it may overlay text.

If, for example, you enter

```
.dr thump weight .4mm
.ud thump p3
```

a rule .4mm thick is drawn three pica points above the baseline, through the middle of the text:

~~a rule .4mm thick is drawn three pica points above the baseline, through the middle of the text.~~

The 3800 Printing Subsystem Model 3 and the 3820 Page Printer fonts include underscoring information in the font objects. This built-in underscore definition will be used for these fonts unless you explicitly specify the .UD [Underscore Definition] control word with a rulename or position.

Each time a new font is started or restarted, the underscore definition is changed to use the underscore definition in the new font. However, this definition does not take effect until a new underscore rule is started for the next output line if underscoring of blanks is on, or for the next word if underscoring blanks is off.

For more details on drawing rules see "Drawing Horizontal and Vertical Rules" on page 157.

Using the .IC Control Word for Emphasis.

On page printers, you can also use the .IC [Intercharacter Space] control word to insert extra white space between characters of a word for emphasis. For example, if you enter

```
We must
.ic espace p6
emphasize
.ic espace 0
this word.
```

two extra pica points of horizontal white space is inserted between each pair of characters in the word emphasize:

```
We must e m p h a s i z e this word.
```

Chapter 16. Keeping Blocks of Text 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
- Placing a figure or diagram at the top or bottom of a column or page
- 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).

Keeps

When you wish to keep a specific group of lines, such as a figure or example, together, consider using:

- A regular keep, started with .KP ON, is placed in the current column if it will fit. Otherwise, a column eject is performed and the keep is placed in the next column. If necessary, a new page may be started to force the keep to be placed at the top of the page body.
- A floating keep, started with .KP FLOAT, is placed in the next available column if it does not fit in the current column. If the float does not fit into the current column, it is saved and the text that follows it in the input file is formatted and placed in the current column. Once a float has been placed, neither it nor the text that was moved before it can be rearranged for text distribution purposes.
- A delayed keep, started with .KP DELAY, is always placed in the next column, whether or not it fits in the current column. As with floating keeps, text following the keep in the input file can 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. The formatting environment is restored when the keep ends. See the *Document Composition Facility: SCRIPT/VS Language Reference* for a list of the formatting parameters saved and restored around keeps.

For example,

```
.kp on
.in p6
.ir p6
.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 can 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,
.kp float
   ....
   (drop in figure here)
   ....
.kp off
but this text can appear above or
below the figure, depending upon whether
the figure is moved to the next column.
```

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

v is an amount of vertical space. These kinds of keeps do not cause breaks. 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 1i
.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 can 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, 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 12. Composing Multiple-Column Pages" on page 141 for more information on column balancing.

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, except for v and v+v type of keeps. v and v+v type of keeps will combine their depths. 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 cause 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. See the *Document Composition Facility: SCRIPT/VS Language Reference* for a listing of these control words.

Floats

Figures and tables often are not related 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
.hr left right
.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 horizontal rule (Figure 3 on page 44 illustrates such a float.)

Floats can be specifically designated for odd- or even-numbered pages. For example,

```
.fl on page even
.im tblleft
.sp 2
.fl off
.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, ahead of any unordered floats.

When a single chapter of a document does not contain enough pages of text to accommodate all the floats defined within the chapter, you can specify

```
.fl dump
```

before beginning the new chapter. 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. See the *Document Composition Facility: SCRIPT/VS Language Reference* for a listing of these control words.

Widow Zones

When SCRIPT/VS is concatenating input text, it will automatically prevent single output lines at the beginning or end of a paragraph from being left alone at the bottom or top of a column or page. This is called widow zone control. If a paragraph spans two columns, at least two lines of the paragraph will appear in each column.[48] Widow zone control can be turned off by specifying

```
.wz off
```

Note: For purposes of widow zone control, SCRIPT/VS considers paragraphs to be delimited by breaks.

When a vertical, inline keep (.KP v or .KP v + v) is ended and widow zone processing is on, the ended keep is treated as the first two lines of a widow zone. When regular, floating, delayed and inline keeps are ended, SCRIPT/VS does not attempt to keep the last line in them together with the next line. This may cause one line widows to occur.

Widow zones are ended by any control that causes a break or by a line with a leading tab or blank. A widow zone is also ended whenever a line is encountered that is more than one-third the depth of the page body.

---

[48] When widow zone control is in effect, paragraphs of fewer than four lines will not be split between columns.

Chapter 17. Creating Footnotes

The .FN [Footnote] control word allows you to have text formatted and placed 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 following example will produce the footnote at the bottom of this page.

```
.fn on
** This line is going to
appear as a footnote
on this page.
.fn off
```

SCRIPT/VS prints a horizontal rule of 16 figure spaces, 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
**********
.sk
.fn off
```

Normal Footnote Placement

Because there is no maximum depth for a footnote, once a footnote is started, text is included in the footnote until a .FN OFF control word is encountered, or 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 the footnote refers to (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 and no attempt is made to associate it with a callout line or widow.

** This line is going to appear as a footnote on this page. Unless otherwise indicated, footnotes are generally aligned against the left page margin. In this book, offset style is used and footnotes have been adjusted to be aligned with the offset text. This also applies to the footnote leader.

A line or widow containing a footnote callout will be placed on the page if there is sufficient space for all of the following:

- The line or widow
- The footnote leader
- At least two lines, counting skips and spaces, of the last footnote referenced in that line or widow. If the footnote is only three or less lines deep, then the entire footnote must fit on the page.

If there is insufficient space on the page for the line or widow, the line or widow and its associated footnote 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 as will fit. The remainder of the footnote will be placed on a subsequent output page.

In placing footnotes, SCRIPT/VS will, if necessary, attempt to split footnotes only if they are four or more lines (including skips and spaces) deep. 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 a vertical space generated by a single control word (for example, .SP 3) are considered to be single lines.

When a footnote is split, or cannot be placed on a page (for example, the first of two footnotes called out on a page is greater than the space allowed for footnotes on that page), the remainder will be allowed to float to the next available page.

Whenever a new page is started, footnotes that were allowed to float from previous pages are placed on the new page. In placing footnotes that were floated from previous pages, SCRIPT/VS will attempt to reserve space 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.

The .FL (Float) DUMP control word causes SCRIPT/VS to place all floats, including footnotes, before resuming input text processing.

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 occur when:

- The page depth is very small
- The footnote leader is very large
- 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.

If a footnote begins with one or more skips and the footnote is the first one to be placed on the page, the size of the first skip will be made zero.

Other Footnote Considerations

You can mark up a 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 20. Defining the Formatting Environment" on page 219 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.

****

[2] This is the next footnote in this section.

Chapter 18. Translating Characters

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 performs 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
- Expanding individual input characters into character strings.

Translating Output Characters

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 cannot, for example, 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 the 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. For example,

```
.tr * af
```

Each occurrence of an asterisk in your file is translated, on output, to the bullet character (•) that has the hexadecimal code AF.[49] For example, the input line

```
* Pay attention to this point.
```

results in:

```
• Pay attention to this point.
```

You can specify as many translation pairs with one .TR [Translate Character] control word as your input line allows. For example,

---

[49] The hexadecimal codes for each character for line devices are shown in the *Document Composition Facility: SCRIPT/VS Language Reference*. The hexadecimal codes for page printer fonts depend on the code page being used. See the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer font catalogs for the appropriate code points.

```
.tr 0 b0 1 b1 2 b2 3 b3 4 b4 5 b5 6 b6 7 b7 8 b8 9 b9
```

causes the characters 0 thru 9 to print as their corresponding subscript characters if they are available in the current font. For example, the formula: [50]

```
X2+Y2=Z3 prints as: X₂+Y₂=Z₃
```

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.

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 cannot be displayed in the output. The positions occupied by the translated characters can 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 (*) to bullets (•) only if output is going to the printer. The .IF [If] control word is discussed in detail in "Chapter 22. Processing Logical Statements" on page 253.

Translating Input Characters

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. 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:

---

[50] Superscript characters are not available in all fonts on all devices.

- 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. Remember that *each* 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 *0* in the above control word would be translated to @ before the control word is processed. The only way to restore the 0 to its original definition is by issuing .TI [Translate Input] with no parameters.

- 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
```

Capitalizing Text

SCRIPT/VS provides several means of capitalizing text. They are:

- The UPCASE option of the SCRIPT command, described in detail in the *Document Composition Facility: SCRIPT/VS Language Reference.*
- The .UP [Uppercase] and .UC [Underscore and Capitalize] control words, described in detail in the *Document Composition Facility: SCRIPT/VS Language Reference.*
- 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 in "Chapter 6. Composing Lines" on page 71
- The &U' symbol attribute, described in "Chapter 21. Processing Symbols" on page 223

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 can 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.

Translating Strings of Characters

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 can 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 can contain both text and control words. For example,

```
.dc cw off
.ts < /;.bf ;"/
.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 can 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 *all* occurrences of the character, and are performed before any other processing of the line.

If you specify .TS IGNORE, this causes SCRIPT/VS to ignore the width of specified characters when it is measuring the text to put on an output line. For example, if you specify

```
.ts a ignore
```

this causes SCRIPT/VS to assign a width of 0 to the character "a" when SCRIPT/VS measures text.

Prefixing Input Lines

You can use the .PX [Prefix] control word to replace control characters at the beginning of input lines with control words, macros, or other strings.

With the .PX control word, you can get control over lines that start with a particular character. For example, a complier listing might have printer carriage control characters in the first position of each output line. A new page might be signalled by a line that starts with the ASA control character 1.

You can write a macro called NEWPAGE, and then cause the macro to get control whenever SCRIPT reads a line that starts with the control character one. To do this, you would use the .PX control word:

```
.px 1 /.NEWPAGE/
```

This will cause every 1 in position one to be deleted and replaced with the character string .NEWPAGE. Because the line now starts with a period, followed by the name of a defined macro, SCRIPT will give control to that macro just as though the original input line had started with .NEWPAGE.

Only lines read from an input file are examined for prefixing; macro lines are never prefixed. Prefixing is performed before symbol substitution and is performed even when symbol substitution is off.

To treat input lines beginning with an asterisk (*) as comments, you can prefix them with the SCRIPT/VS comment characters:

```
.px * /.*
```

Any input line beginning with * is prefixed: the * is replaced with .*.

You could turn off the input line prefixing as begun in the previous example by entering:

```
.px * off
```

If an input line does not begin with a currently defined prefix control character, you can use

```
.px null /.*
```

to add a period and an asterisk (.*) to the front of subsequent input lines (until turned off with a .PX OFF control word) without disturbing the first character in the input lines.

Chapter 19. Creating an Index

SCRIPT/VS enables you to automatically produce an index, such as the one contained in this publication. You must 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. If the INDEX option of the SCRIPT command is not specified, all .PI control words are ignored. 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 index can contain multilevel entries and cross references. SCRIPT/VS generates 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.

Placing the Index in a Document

Use the .IX [Index] control word to indicate where you want the index to be placed. When .IX is encountered, SCRIPT/VS:

- Starts a new page, if it is not already at the top of a page
- Prints the word *INDEX* as a heading.

If you want a different title for the index page, you can specify it as

```
.ix Subject Index
```

this indicates that *Subject Index* is to be used for the title instead of *INDEX*.

If you do not want a title at all, specify

```
.ix /
```

and SCRIPT/VS will generate the index without a title.

TWOPASS Considerations

In order to have correct page numbers in the index, pages must be numbered the same way on both passes. On the first pass, the index is empty. On the second pass, it can contain several pages of information.

You should reserve a range of page numbers for the index. This ensures that the page following the index will have the same page number for both passes so that the page

number for the index entries collected during the first pass will be correct during the second pass. For example, if the index will require three pages, you can reserve the current page number and the next two page numbers by specifying:

```
.ix 3 Index
```

Another way to ensure consistent page numbering between the two passes is to explicitly reset the page number with a .PA [Page Eject] or .PN [Page Numbering Mode] control word *before* any head level or .IX [Index] control word is encountered that requires knowledge of the page number.

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 third-level index entries with the same first- and second-level terms are collected and formatted alphabetically. These third-level terms are then placed in the index immediately following the second level-term to which they apply.

A description of weasels, for example, can contain the following:

```
.pi /weasels/training
   ...
.pi /weasels/care and feeding
   ...
.pi /weasels/breeding
```

The index entry for *weasel* looks 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
      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 there are a large variety of topics in a document, 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.[51] 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
      dental hygiene  19
      exercise  15
      nutrition  16
   training  14
```

---

[51] The REF parameter is valid only if at least two terms are specified.

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
```

Because there are no nonreference terms under the entry *circular definition,* the cross reference is prefixed only with *See.*

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 entry is, by default, 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
```

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. The page numbers come from the three subsequent occurrences of that term.

Handling Special Characters

Index terms often contain special characters that, 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 because 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 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, because 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
```

because the blank precedes *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
```

Initially, there are no characters that are ignored for index sorting.

There may be occasions when you want special characters treated as 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
```

because 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
```

Initially, only hexadecimal 40 is treated as a blank when sorting index entries.

Explicitly Specifying Sort Keys

Occasionally, you may want to place an index entry in some section of the index independent of the entry's default 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 that 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
```

The term *Austria-Hungary* is placed 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, because 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 comes before any other .PI control word with "walrus" as its index term.

Creating the Index

When you specify the .IX [Index] control word, SCRIPT/VS formats the index by creating and executing the .IE [Index Entry] control words generated 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
```

generates the first-level term, *Pines*, with two second-level terms, *Japanese Black* and *Lodgepole*.

When you specify .IX, SCRIPT/VS formats these terms by creating and executing the following control words:[52]

```
.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
```

---

[52] The blank between the control word name and the first parameter does not have to be specified. It is omitted in the .IE [Index Entry] control words generated by .IX [Index].

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.[53] The .IX control word generates and executes a header for each section of the index for which there are entries.

---

[53] Because 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.

Part 3. SCRIPT/VS Programming Facilities

This section of the book contains information about the programming facilities of SCRIPT/VS.

Included in this section are the following chapters:

- Chapter 20 - Defining the Formatting Environment
- Chapter 21 - Processing Symbols
- Chapter 22 - Processing Logical Statements
- Chapter 23 - Processing Macros
- Chapter 24 - Processing GML
- Chapter 25 - Verifying Spelling.

Chapter 20. Defining the 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 environment, which contains parameters that define the entire page
- The translation tables associated with the .TI [Translate Input] and .TR [Translate Character] control words.

The Formatting Environment Parameters

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 used for body text and initializes the active environment for formatting running headings and running footings.

   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 floats exist and will fit, they are placed on the page.

   The output page's running headings and footings, 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 output destination.

The Running Heading and Footing Environments

When running headings and footings are started, SCRIPT/VS:

- Saves the current formatting environment
- Restores the initial formatting environment
- Modifies the environment to reflect changes that have been made with the .DC and .GS control words.

When the running heading or footing ends, the saved formatting environment is restored.

The Keep, Float, Footnote, and Named Area Environments

When keeps (other than inline keeps), floats, footnotes, and *named* areas are started, SCRIPT/VS saves a copy of the active environment and then modifies it by:

- Clearing the values of the .OF [Offset] and .UN [Undent] control words
- Restoring the indention to the basic .IN [Indent] value currently in effect.

In addition, for page floats and footnotes, the column line length is set equal to the line length value.

When the keep, footnote, float, or *named* area ends, the saved copy of the active environment is restored.

Saving and Restoring the Current Formatting Environment

The .SA [Save Environment] and .RE [Restore Environment] 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 environment
- The .TI and .TR translation 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. You could use the .SA [Save Environment] and .RE [Restore Environment] control words, as in the following example,

```
.sa
.in
Distribution list for special publications:
.sk
.in 3m
.ti ¬ 05
.tp 2i 2.5i
.us Name ¬Dept ¬Address
 .
 .
 .
.* End of distribution list
.re
```

to avoid having to restore the main document's values.

Named Environments

The .SA control word saves environments in a stack or by *name*. The .RE [Restore Environment] control word restores the SCRIPT/VS environment to the values that were in effect at the time of the corresponding .SA control word.

To save the current environment by *name*, enter

```
.sa barnes
```

The current environment is saved as BARNES.

The .SA control word only *saves* a copy of the values of these SCRIPT/VS variables in the current environment, it does not change any of these variables.

Because .SA does not change any of the SCRIPT/VS variable settings, all variables should be explicitly set to the appropriate values unless the current settings are known. For example, you can explicitly set indention to 0 and then restore it to whatever it was previously.

The .RE control word restores the SCRIPT/VS formatting environment from a *named* saved environment or 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.

To restore the most recently saved *unnamed* environment, enter

```
.re
```

To restore the *named* environment BARNES, enter

```
.re barnes
```

If there is no currently active .SA control word, the .RE control word restores the initial values.

Chapter 21. Processing Symbols

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 current value of the symbol. 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 specify

```
.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 name can be up to ten characters long and can contain upper and lowercase characters, numbers, and the characters @, #, and $.

The symbol 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 value contains blanks or special characters, enclose the entire value in single quotation marks (as shown in the example above).

The .SU [Substitute Symbol] control word causes SCRIPT/VS to stop or resume the substitution of defined set symbols. The .SU control word causes a specified number of subsequent input lines, including 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 value but it is reset to OFF with .SU OFF or with .SU n, after n lines have been read.

Even when symbol substitution has been turned off with the .SU [Substitute Symbol] control word, symbols on a .SE control word line will be substituted if they are not enclosed in single quotation marks.

Some examples of valid symbol definitions are:

- A numeric value:

```
.se number = 25
.se add = 1
```

- A character string:

```
.se text1 = 'IBM 1403 Printer'
.se TEXT1 = 'IBM 1403 Printer'
```

- A character string that includes a quoted phrase:

```
.se type 'prepared on a "word processing" machine
.se type = 'prepared on a "word processing" machine
```

- A SCRIPT/VS control word:

```
.se break = '.br'
```

- The value of another symbol:

```
.se printer = '&text1'
```

You can perform integer arithmetic with symbols:

- To increment it:

```
.se incr = &add + 1
.se next = &number + 1
```

- To decrement it:

```
.se prev = &number - 1
```

- To divide it:

```
.se half = &number / 2
```

- To multiply it:

```
.se mult = &add * 10
.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 the terminal during SCRIPT/VS processing in interactive environments.

For example, a symbol called *name* could be set with the following control word:

```
.se name = 'Ray Hicks
```

The same symbol could also be set this way:

```
.rv name = '
```

At this point, SCRIPT/VS issues a read to your terminal and you can enter the material to be used as the value of the symbol. In this example, you would enter:

```
Ray Hicks
```

You must use single quotation marks in the same circumstances where they would be required in a .SE control word.

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 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 the first character (character 1) and continuing for six 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*.

The SUBSTR (substring) function also can be used to extract characters from a character string that is not another symbol 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).

The INDEX and SUBSTR parameters can be used together to process control information. For example, the SYSVAR option of the SCRIPT command can be used to specify formatting parameters and you can define SYSVAR *C* to establish the number of columns. You can validate that a permissible value has been given with the INDEX parameter:

```
.se x = index '-1-2-' '-&SYSVARC.-'
.if &x eq 0 .mg /e/SYSVAR C invalid.
```

If the value given in &SYSVARC is valid, you can use the symbol &x, set with the INDEX parameter and the SUBSTR parameter to convert synonyms of valid values to a standard value:

```
.se cols = substr '1-2' &x 1
```

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 name with its value. A symbol name is ended either with a blank, a period (.), or the end of the input line. If the symbol name ends with a blank, the blank is treated as a normal input character and is left in the input line. If the symbol name ends 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 &item1..
```

results in an end-of-sentence period concatenated with the value of the symbol named &item1. 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 scheduled
for shipment on &shipmo &shipday., 19&shipyr..
```

In this example, values for the symbols are substituted and the printed sentence appears as:

```
The name of our product is Whizbanger, which is scheduled
for shipment on January 22nd, 1985.
```

If you do not place an end-of-symbol 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.

SCRIPT/VS limits an input line to a maximum of 256 characters. Because symbol substitution is performed *before* the line is evaluated, with symbol substitution on, the complete input line including the substituted symbol values should not exceed 256 characters. If the substituted line exceeds 256 characters, it will be split and the remaining characters will be treated as a separate input line.

One exception to this processing occurs with symbol arrays. If an input line contains a symbol array, for example, &sym(*), one or more elements of the array will be segregated into additional input lines rather than splitting the line at 256 characters. However, if the line contains a single element of a symbol array, for example, &sym(1), then the symbol will be treated as a simple symbol.

Lines split during symbol substitution processing will be processed as if they were separate lines in the input file. Therefore, it is possible even with formatting off, for a single input line to appear as two or more output lines in the formatted document. This may result in errors if a split line contains a script control word.

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 238.

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.
- 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.
```

- 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.[54] 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 that maps that hexadecimal code to the character &.

  You can then use the symbol &amp wherever you want an ampersand to appear.[55]

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.[56] A character string cannot be a symbol name if:

---

[54] Make sure the hexadecimal code you are using is not already being used or you may have unpredictable results.

[55] This technique has been used in marking up this book whenever a text ampersand is required.

[56] Except in a running heading or running footing where the ampersand (&) will be mistaken for the page number symbol.

- It has not been defined as such with a previous .SE [Set Symbol] or .RV [Read Variable] 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 that you have 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.

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.

Attributes of a Symbol Value

SCRIPT/VS provides you with the ability to determine some of the characteristics of a symbol in your input file, such as:

- Existence (&E')
- Length (&L')
- type (&T')
- Current value (&V')
- Width (&W')

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 space value into horizontal device units (&DH')
- A space value into vertical device units (&DV')
- A numeric symbol value to its roman numeral character string equivalent (&R' or &r')
- A character string to uppercase (&U')
- A character representation of a hexadecimal string to that string (&X')

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 the symbol attribute &A', which converts a numeric value to an alphabetic character string, have different meanings when specified in uppercase and lowercase. These attributes are only substituted if symbol substitution is on.

&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 results in the string B
- &A'26 results in the string Z
- &A'27 results in the string AA
- &A'28 results in the string AB
- &A'705 results in the string AAC
- &a'28 results in the string ab

The largest number that can be converted is 65535. Numbers higher than this return a zero.

If the character string to be converted is not a decimal integer number, the result is zero (for example, &A'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 value or the number of characters in any character string. 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 roman numeral equivalent of the number:

- &R'87 causes the string LXXXVII to be substituted.
- &R'19&SYSYEAR causes the string MCMLXXXV to be substituted (in 1985).

- &r'87 causes 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).

If the character string to be converted is not a decimal integer number, the result is zero (for example, &R'zorch = 0).

&T' analyzes the symbol type. It also replaces the character string with:

- N, if the value is numeric.
- 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
```

Note: The capitalization that takes place is the same as all other uppercase conversions and is controlled by the .TU [Translate Uppercase] control word.

&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 *character string* 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 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. Because &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 *character string* &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, because &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 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).

&W' yields the width of a character string in figure spaces, measured in the current font. If, for example, you wanted to offset some following text by the width (in horizontal device units) of a name within a symbol, you could specify:

```
.se width = 100
.of &width.dh
```

and text following the .OF control word will be offset by the value, in horizontal device units, of the name found within &home.

&X' converts a hexadecimal notation into a character 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•[TRANSFORM²]≠0
```

If an even number of hexadecimal digits is not specified or an invalid hexadecimal digit is encountered, the value of the &X' attribute is zero.

Space Unit Symbol Attributes

SCRIPT/VS provides the following space unit symbol attributes:

&DH' Which converts a space unit value into the nearest number of horizontal device units. The space value can be the value of a symbol. For example, if you wanted to set a symbol to convert the width of the current page number symbol to the nearest number of horizontal device units, you could specify:

```
.se width = &dh'&w'&
```

&DV' Which converts a space unit value into the nearest number of vertical device units. The space value can be the value of a symbol. For example, if you wanted to set a symbol to the height of one vertical device unit, you could specify:

```
.se height = &dv'1
```

&AD' Which indicates the depth of material within a specified *named* area. If the specified areaname is not the name of a currently defined area, or if there is nothing in the area at the moment, then 0 is returned. The value returned does not include the depth of skips at the top and bottom of the area. See "Chapter 11. Placing Text in Named Areas" on page 133 for more details on the &AD' symbol.

&SW' Which indicates the width of a named segment. If the specified segment does not exist, or if NOSEGLIB was specified on the SCRIPT command, or if you are formatting for a line printer, then 0 is returned. See "Imbedding Segments in Your Documents" on page 55 for more details on the &SW' symbol attribute.

&SD' Which indicates the depth of a named segment. If the specified segment does not exist, or if NOSEGLIB was specified on the SCRIPT command, or if you are formatting for a line printer, then 0 is returned. See "Imbedding Segments in Your Documents" on page 55 for more details on the &SD' symbol attribute.

Note: The size of device units varies widely from one device to another, ranging from about six units to the inch to 600 units to the inch. If you have converted numbers into device units so you can do integer arithmetic with the values, be sure to tell SCRIPT that the number is in device units when you use it in a control word.

For example if you have calculated an indention in horizontal device units in the symbol &hin, you can then use the value with:

```
. in &hin. DH
```

If you do not specify that the value is in horizontal device units by appending DH, the size of the resulting indention may be much larger than intended.

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 can occupy as many lines as required.) In ATMS-III, a library member is a document or a subdocument.

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 does not. 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 will be searched for the definition of an undefined symbol only when .LY ON or .LY SYM is specified.

Note: Searching the library for symbol values and macro definitions uses a lot of processing time. This is especially true for the forward referencing of symbol values, a case in which there are normally many potentially unresolved symbols. For this reason, the .LY control word is provided to control library lookup. If a symbol is found in the library, it is defined in the SCRIPT/VS symbol table and processing time used for this purpose is thereby reduced.

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 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 the *Document Composition Facility: SCRIPT/VS Language Reference.*

Most 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 symbol names 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
```

These symbols could, for example, yield the following dates and times:

```
DATE: 01/22/85
TIME: 12:50:02
```

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.

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, 38PP, 3820, 4250

SCRIPT Command Options		
Symbol	Description	Value
&$BE	Even bind [3,4]	0 - nnn
&$BO	Odd bind [3,4]	0 - nnn
&$CHAR(n)	Fonts [5]	1 - 4 characters
&$DCF	SCRIPT/VS release level	3.0
&$DDUT	Utility file redefenition [6]	0, 1
&$INDX	Indexing [6]	0, 1
&$LIB	Macro library available [6]	0, 1
&$PARM	Command options [7]	8 - 256 characters
&$PASS	Current pass number	1, 2
&$PRT	Current page print switch	ON, OFF
&$SYS	Environment	CMS, TSO, VS2, VSE, CICS
&$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 on 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 15. SCRIPT/VS System Symbol Names (Part 1 of 2)

Page Characteristics		
Symbol	Description	Value
&$BE	Bind even (BIND)	0 - nnn
&$BM	Bottom margin (.BM) [8]	0 - nnn
&$BO	Bind odd (BIND)	0 - nnn
&$CL	Column line length (.CL) [9]	1 - nnn
&$FM	Footing margin (.FM) [8]	0 - nnn
&$IN	Left indention (.IN) [9]	0 - nnn
&$IR	Right indention (.IR) [9]	0 - nnn
&$LC	Internal line counter [8] [10]	0 - nnn
&$LL	Line length (.LL) [9]	0 - nnn
&$OF	Offset (.OF) [9]	0 - nnn
&$PL	Page length (.PL) [8]	1 - nnn
&$PW	Page width (.PW) [8]	1 - nnn
&$TM	Top margin (.TM) [8]	0 - nnn

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
&$EGML	GML end-tag delimiter [11]	(default: "::")
&$ENV	Formatting environment	BODY,FL,FN,KP,RF,RH,IBP,AR,FNL
&$FNAM	Current input file name	eight characters
&$GML	GML tag delimiter [11]	(default: ":")
&$KP	Keep in process	ON, OFF
&$LNUM	Last line number read	0 - nnn
&$LST	Line started	0,1
&$PN	Page number [12]	1 - nnn
&$PS	Page number symbol [11]	(default: "&")
&$RB	Required blank [11]	(default: hexadecimal 41)
&$RET	Return code from .SY [13]	0 - nnn
&$SU	Symbol substitution enabled	ON, OFF
&$TAB	Tab character	hexadecimal 05
&$TAGD	GML delimiter of last tag	&$GML, &$EGML
&$VR	Vertical rules in effect	ON, OFF

[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, widow zones and partial lines.

[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 .SY is not supported.

Figure 15. SCRIPT/VS System Symbol Names (Part 2 of 2)

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, January 22, 1985.
```

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, January 22, 1985.
```

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. You can obtain the current value of the parameter 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.

You can use this technique to ensure proper results even though some formatting parameters can be changed by control words. For example, the following sequence produces a box the width of the output page:

```
.se indent = &$IN
.se rindent = &$CL - &$IR
.if &indent eq 0 .se indent = 1
.bx &indent &rindent
.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 formatted 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 headings or footings that have been placed on the page, nor does it include keeps, widows, or partially filled output lines that have not been placed in the column at the time of symbol substitution.

The value of &$LC at the time of symbol substitution cannot accurately reflect the final position of surrounding text on the page if that text is in a keep, float, or widow, if there is a partially filled output line, or if column balancing is in effect. The value of &$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.

The &$DCF Special Symbol

The &$DCF special symbol indicates the release level of the module being executed. The value will be 3.0 for Release 3, level 0.

The &$DDUT Special Symbol

The &$DDUT special symbol indicates whether the NODDUT or DDUT command option is in effect. The value is either 0 or 1. It is 0 when the NODDUT command option has been given on the command line. It is 1 when the DDUT command option has been given.

If neither option has been specified, the value of &$DDUT is 0, because NODDUT is the default. This special symbol is meaningful only in the CMS and TSO environments.

In ATMS-III and batch, &$DDUT is always set to 1, indicating that DDUT is the default command option of the pair in these environments.

The &$GML Special Symbol

The &$GML special symbol reflects the value of the current GML tag delimiter as set by the .DC [Define Character] GML tag.

The &$EGML Special Symbol

The &$EGML special symbol reflects the value of the current GML end-tag delimiter as set by the .DC [Define Character] GML tag.

The &$ENV Special Symbol

The &$ENV special symbol indicates the current formatting environment. The range of possible values includes:

AR	*Named* area
BODY	Body of page
FL	Float
FN	Footnote
FNL	Footnote leader
IBP	In between pages
KP	Keep
RF	Running footing
RH	Running heading

The &$LST Special Symbol

The &$LST special symbol indicates whether or not an output line is started. The value is either 1 or 0. It is 1 when text has been formatted (line started), but the line has not yet been completed. It is 0 when a line has been promoted into the current column and no more text has been formatted yet. The value of &$LST is 0, for example, following a .BR [Break] control word.

The &$PASS Special Symbol

The &$PASS special symbol indicates if the current formatting pass is the first or second pass. A value of 1 or 2, respectively, is returned.

The &$PASS system symbol can be used with the &$TWO system symbol to determine whether or not the current formatting pass is the last pass. For example, you could specify

```
.if &$TWO = 0 .se lastpass = yes
.el .if &$PASS eq 2 .se lastpass = yes
.el .se lastpass = no
```

Then if the TWOPASS option of the SCRIPT command wasn't specified, the current formatting pass is the last pass. Else, if the TWOPASS option was specified, the last pass will be indicated if the &$PASS symbol equals 2.

The &$PRT Special Symbol

The &$PRT special symbol indicates whether the current page will be written to the output destination or discarded. Pages are always discarded on the first of two passes when the TWOPASS option of the SCRIPT command is used. Pages can be discarded when the PAGE option of the SCRIPT command is used. The value is either ON or OFF.

The &$TAGD Special Symbol

The &$TAGD special symbol indicates whether a GML tag or end-tag was last processed. The value is equal to either &$GML or &$EGML.

The &$VR Special Symbol

The &$VR special symbol indicates whether vertical rules are currently being drawn, due either to the .BX [Box] or the .VR [Vertical Rule] control words. The value is either ON or OFF.

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 or as a comparand for an .IF control word line, always use the uppercase symbol name and character-string values.

For example, &SYSVARA can be used to bypass parts of the document and &SYSVAR2 can be used to terminate processing before completion:

```
.if &SYSVARA eq ATYPE .go aproc
.if &SYSVAR2 eq NOGO .qu
...aproc
```

When you use &SYSVAR symbols, you should 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 the *Document Composition Facility: SCRIPT/VS Language Reference.*

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. As many as 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 4. Combining SCRIPT/VS Input Files" on page 47 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 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 23. Processing Macros" on page 261 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. Each time a macro is called, a new set of local symbols is established for it. These symbols are deleted when the macro ends.

Unlike other symbols, local symbols, when undefined, are replaced during symbol substitution with the null string.

Setting a Symbol to 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. You can define an array symbol 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. The ASEP parameter of .DC allows you to define up to *four* characters that 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 in separating array elements must be specified, including blank characters (as 40).

The initial array separator characters are the comma (,) and the blank. The characters given with ASEP replace the previous array separator characters.

Suppose you have defined a symbol array containing the names of geographical locations:

```
.se place() = Goldsboro
.se place() = Kunsan
.se place() = Misawa
```

When the symbol &place(*) is found, it is replaced with the entire symbol array and the elements of the array are separated by the array separator characters. For example, the initial setting of the array separator characters is a comma (,) and a blank, so if you enter

```
At: &place(*)
```

the result of symbol substitution will be

```
At: Goldsboro, Kunsan, Misawa
```

You can specify up to four characters with the ASEP parameter. You can insert more than four characters, including control words, between array elements by setting the array separator characters to a symbol. For example,

```
.se x = ' and '
.dc asep &x .
At: &place(*)
```

The intermediate result of symbol substitution is

```
At: Goldsboro&x. Kunsan&x. Misawa
```

The final result of symbol substitution is

```
At: Goldsboro and Kunsan and Misawa
```

If you want to set the array separator to cause a break, specify:

```
.'se *x = ';.br;'
.dc asep &x .
```

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 symbol name and the element index number in the form

```
&name(n)
```

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 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 to 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 is 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 values are substituted.

The array element number can be another symbol. For example,

```
.se elem = 1
.se array(&elem) = &
```

No blanks can 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.

An example of a complex symbol is:

```
.se x = ry
.se ry = 4
&&x
```

which resolves as follows:

1. &ry
2. 4

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 indicates which element should be set next if you did not 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 but the individual elements are not reset.

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
```

n is any nonnegative integer. 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 cause substitution of all elements in the array (except element zero), or you can cause substitution for 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. 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 cannot 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.

For more information about the control word modifier, see the SCRIPT/VS control word description section of the *Document Composition Facility: SCRIPT/VS Language Reference.*

Defining Text Variables

There are certain times when you may want to define and assign a value to a particular text variable. For example, you might want to print a special character, such as the Greek alpha character, that is not on your keyboard; or you might want to override a delimiter character so that it is printed as ordinary text; or perhaps you might want to change fonts within a GML heading without having to rewrite the APF that defines this heading. The .DV [Define Variable] control word provides a simple way to resolve all of these text variable concerns.

Producing Special Characters

For many applications, you may need to be able to select any arbitrary character from any arbitrary font, and to do so whenever you want. This can be a problem, however, because the special character you need may not be on your keyboard. Such special characters may include:

- Accented characters for languages other than English
- APL characters
- Greek characters.

Producing a Greek Alpha Character

One way of producing unkeyable characters, such as the Greek alpha character, in SCRIPT/VS is to use a code page defined for the set of characters you wish to use. In this way, you can allocate, using the .TR [Translate Character] control word, the keys you do have to represent whatever characters you wish. This is not a complete solution, however, because a required character might still be in a different font than that used for surrounding text.

If you want to be able to produce a Greek lowercase alpha character, one approach in SCRIPT/VS is to define a symbol called **alpha** that can be used by entering the symbol &alpha. in your input file. If, for example, the alpha character is at code point hexadecimal 41 in a font called **greek**, then the alpha symbol could be defined as follows:[57]

```
.'se alpha '&$CONT.&$CW..bf greek;&X'41.;.pf;
```

Because this coding frequently does not produce correct results, a better approach is to use the .DV control word. The same Greek alpha character, which is hexadecimal 41 in the **greek** font, can be defined with .DV as follows:

```
.dv alpha font greek /$X'41
```

In this example, the symbol **alpha** is defined to associate a font (**greek**) with a text string (hexadecimal 41).

The symbol **alpha** may also be used in the text of a heading as shown in the following example

```
.h1 the &alpha symbol
```

---

[57] A continuation character must have been previously specified, and a font named **greek** must have been previously defined with the .DF control word.

or as part of a split text control word line such as the following:

```
.sx /&alpha.//greek/
```

See "Using Defined Variables to Change Fonts" on page 251 for more details on using defined variables.

Overriding Delimiter Characters

Another potential problem in SCRIPT/VS is the occasional need to treat one of the special delimiter or escape characters as text. The following characters have special meaning to SCRIPT/VS and need a mechanism to override them if they are to appear as ordinary text characters:

- A *period* (.) delimits a control word when it is in column one of an input line or when it follows a control word separator.
- The *ampersand* (&) delimits a symbol.
- The *GML delimiter* (:) delimits a tag.
- The *page number symbol* (& by default) is replaced by the current page number in running headings and footings.
- The *control word separator* (; by default):
  - Allows several control words to appear on one line, or
  - Delimits a control word sequence, or
  - Causes a line to be treated as two logical input lines when it occurs as the first character in a symbol's value.
- The *required blank* (hexadecimal 41 by default) is automatically converted to white space in the output.

In SCRIPT/VS, there are ways to produce any of these characters as ordinary text, but a different mechanism is required to produce each one. In some cases, it is necessary to shut off the normal function of a character in order to use it as text. With text variables, a single mechanism can be used to define symbols for all of these characters, thereby making them available as text characters anywhere in the document. This can be done without shutting off the character's normal function.

In SCRIPT/VS, symbols are commonly defined in a GML profile to allow special delimiters to be used as text characters. For the ampersand, which is the symbol delimiter and also the default page number symbol, the symbol &amp. is defined to be used where a text ampersand is needed. It is done as follows:

```
.se amp &X'08
.tr 08 &
```

In other words, the symbol &amp. is replaced by a hexadecimal 08 by substitution, and then all X'08's are translated to ampersands on output.

A simpler method of producing the &amp. symbol is to use the .DV control word. With text variables, for example, all symbols like &amp. are defined in this form:

```
.dv amp text /&
```

The **TEXT** form of .DV is used for a character string that has no associated font and that is merely defined as known text. This method has the following benefits:

- No code point need be given up for this symbol
- The translate table is not needed
- The symbol may be used anywhere freely, including in a running heading or footing.

Using Defined Variables to Change Fonts

In SCRIPT/VS there is no mechanism whereby a font can be changed in the middle of certain elements that require all of the text on the same line, such as topic headings or the content of split text control words. You can not, for example, change fonts in a heading in SCRIPT/VS unless you disregard all of the built-in heading and table of contents functions, and rewrite these functions yourself as macros.

A text variable defined with .DV, however, can have a font associated with it, and it can be used freely anywhere, including in a topic heading, table of contents, index entry, or split text. If the text variable requires a change of fonts, this is done automatically as part of the text variable support.

If you want to add a special character from the Pi font to the font of a heading, for example, you could enter:

```
.df pifont type('Pi font Sans Serif' 14 codepage AFTC0363)
.dv special font pifont /&x'fc
.h1 &special. Words to the Wise
```

The result of these lines is that the telephone character will be printed as the first character in the heading.

Chapter 22. Processing Logical Statements

SCRIPT/VS provides several methods for processing input logically or conditionally. You can write input files and macros that are capable of making simple decisions and taking action based on the result. With logical processing techniques, you can do the following:

- 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.)
- Write macros that 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 basic logical processing techniques in conjunction with other techniques that are not discussed here. "Chapter 23. Processing Macros" on page 261 contains information about the mechanics of writing macros, and "Chapter 21. Processing Symbols" on page 223 discusses symbol substitution. Individual control words are described in the SCRIPT/VS control word description section of the *Document Composition Facility: SCRIPT/VS Language Reference.*

There are three basic logical 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 value to determine whether to process an input line or ignore it. To make this conditional test, you can use the .IF [If] control word alone or in conjunction with the .AN [And], .OR [Or], .TH [Then], and .EL [Else] control words. Using the .IF [If] control word alone 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 long,[58] 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:

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 can 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,

```
.if &street eq Broadway .se branch = Commercial
.if &street ne Broadway .se branch = Warehouse
```

causes the same results as:

```
.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,

[58] The entire input line, after substitution, cannot be longer than 255 characters. When comparing symbols that can potentially have long values or contain blanks, we recommend that the .IF control word be performed with substitution off, as described in "Special Techniques for Conditional Processing" on page 256.

```
.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 can be contained in the .NOTES macro or the SALARY file do not affect this series of control words because 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 = Fayetteville .an &state = Arkansas .se zip = 72701
```

which causes the symbol &zip to be set to 72701 if both conditions are true.

Similarly, you can have a situation where only one of multiple conditions must be true for one type of processing to be done. In this case, you might specify

```
.if &city eq Knob Noster .or &city eq Warrensburg
.th .carpool &city
.el .se city =
```

The macro .CARPOOL will be invoked if the value of the variable &city is either *Knob Noster* or *Warrensburg*; 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 current 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 master files. You can provide for slight differences among the files by setting the same symbol to a different value in each master file and using that symbol to determine how processing is to be done in the imbedded file.

The .GO function, on the other hand, can be relatively inefficient. You should restrict its use to situations where it best achieves the required results. When the label follows the .GO in your input file, processing is most efficient if that label is not far from the .GO;

when the label comes before the .GO in your input file, processing is most efficient if that label is near the beginning of the file.

Label processing in macros is much more efficient than in files. However, it is most efficient to branch to a label that is early in a macro because the search for labels always begins at the top of the macro.

The SYSPAGE and SYSOUT Comparands

There are two comparands that you can use with the .IF [If] control word family: SYSPAGE and SYSOUT. They are keywords, not symbols. Therefore, they should not be prefaced with an ampersand (&).

- SYSPAGE tests whether the page currently being formatted is an even- or odd-numbered page (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/
```

- SYSOUT tests whether the destination of the output is the line printer (PRINT), page printer (PAGE), 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 prefix the symbol name with a character-prefix to avoid a possible syntax 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 is null, /&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 because 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 a document might be read by several different audiences, you can customize it for each. 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	Either Class B or Class C (Not Class A)
5	Either Class A or Class C (Not Class B)
6	Either Class A or Class B (Not Class C)

At the beginning of the 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 the 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.

Because the .CS [Conditional Section] control word does not 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.

Logical Processing With Symbols

With set symbols, you can do logical processing in several ways. The simplest of these is to have a symbol that resolves to one control word or another depending on the specific, applicable 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 21. Processing Symbols" on page 223 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 logical processing. The length attribute and the following string or symbol are replaced with the length of the string or symbol during substitution. See "Chapter 21. Processing Symbols" on page 223 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, zeroes need not be added. If it is four digits long, you need one zero, and so on. 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 is concatenated with the number itself when &num is substituted.

Chapter 23. Processing Macros

SCRIPT/VS allows you to define your own processing controls, called macro instructions. 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 first two characters of the macro are treated as a control word.

When to 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,

```
.su off
.dm increment /.se x = &x + 1 /.se y = &x
.su on
```

After SCRIPT/VS processes the macro

```
.increment
```

&x and &y have equal values, because 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. Although seven head levels, .H0 - .H6 [Head Level 0 - 6], 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. Because SCRIPT/VS processes macros as control words, an undefined SCRIPT/VS macro may be 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. You can write the following paragraph macro:

```
.su off
.dm para /.sk /.in 3 for 2 /&*
.su on
```

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

```
.para 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.

The simplest way of defining a macro within a document is this:

```
.dm echo on
.ty ===
.ty &*
.ty ===
.dm off
```

The inline form (ON/OFF) of the .DM [Define Macro] control word allows you to define macro lines on separate input lines. For example, you could define the .PARA macro as follows:

```
.dm para on
.sk
.in 3 for 2
.dm off
```

All of the input lines between the .DM PARA ON and the .DM OFF will be put into the macro definition. Substitution and input translation will *not* be performed on these lines until the macro is invoked. The .DM OFF control word line must begin in column one of the input record and cannot contain the control word modifier. The inline form of the .DM control word requires that you completely define the macro each time.

This type of macro definition includes an implied .SU OFF and .SU ON and .GS TAG OFF and .GS TAG ON. Symbols and GML tags in the macro definition are not resolved but are instead saved as part of the macro and are substituted whenever the macro is executed. This eliminates the need to surround a macro with .SU OFF and .SU ON and with .GS TAG OFF and .GS TAG ON when using this form of the .DM control word.

Another method of defining macros involves using the subscripted form of the .DM control word. For example,

```
.su off
.dm para(5) /.sk
.dm para(10) /.in 3 for 2
.dm para(15) /&*
.su on
```

The macro line number in parentheses is also called the subscript. If the number is omitted from the parentheses, SCRIPT/VS uses an increment of 10, starting at 10. Macro line numbers, if included, do not have to be defined in any particular order. However, when the macro is used, it is executed in subscript order, which is not necessarily the sequence in which the macro lines were entered.

Each line can be defined separately and each line can be given an explicit line number. Each macro line can contain several control words, separated by control word separators. If so, the control word separator scan must be prevented. This can be accomplished by using the control word modifier or the .DM control word.

```
.su off
.dm echo(1) /.ty ==
.dm echo(2) /.ty &*
.dm echo(3) /.ty ==
.su on
```

If line numbers are not given explicitly, they will be automatically generated, using an increment of 10:

```
.su off
.dm echo() /.ty ==
.dm echo() /.ty &*
.dm echo() /.ty ==
.su on
```

Whenever the single line or subscripted forms are used, remember to turn symbol substitution off during the definition so that symbol names will be saved as part of the macro. You also need to turn GML processing off with .GS TAG OFF if there are GML tags in the macro.

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 .PARA macro, you can issue:

```
.dm para(10) /.in 5 for 1
```

or you can cause the .PARA macro to start an inline keep by specifying

```
.dm para(12) /.kp 6
```

Be careful when you mix the forms of the .DM control word. You can use the single line form and the subscripted form within an inline macro definition but you cannot use an inline macro definition within another inline macro definition.

How Values Are Substituted for Symbols within a Macro Definition

Macro definitions almost always contain symbols and these symbols are generally meant to be substituted anew each time the macro is executed. You can ensure that symbol substitution is turned off when you define a macro within a document and that it is turned on when you execute it by using the .DM macroname ON form to define macros. This form explicitly disables symbol substitution for the definition:

```
.dm count on
.se x = &x + 1
.ty &x
.dm off
```

The symbol &x will be saved as part of the macro definition and substituted whenever the macro is executed. Each time the macro is executed the value of x will be different.

If you use the subscripted or the single line form of the .DM [Define Macro] control word and 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 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 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.

When you use the inline form of the .DM control word you do not have to be concerned with when symbol substitution will be performed on the macro lines. It will not be done when the lines are read into the macro definition, only when the macro is actually executed.

Conditional Macro Processing

Macros can be defined to conditionally format a document using the .IF [If] control word family. For example, you can 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 that will execute certain control words only if the document is being formatted for specific individuals:

```
.dm canbe on
.if &who eq Geoff .or &who eq Dot
.th &*
.dm off
```

When you specify

```
.canbe .im pg$sym
```

the file PG$SYM will be imbedded only if the document is being formatted for either Geoff or Dot. The symbol &who has presumably been set elsewhere.

You can also use conditional processing to highlight lines of text differently depending upon the device for which the document is being formatted. The following macro will cause a line of text to be put in uppercase, if the document is being formatted for a 3270 terminal. For all other devices, the line will be underscored.

```
.dm hilite on
.sk
.if &$PDEV eq 3270
.th .up &*
.el .us &*
.sk
.dm off
```

Macro Naming Conventions

A macro name can be up to 10 nonblank characters long, without imbedded blanks or special characters, and is not case sensitive. That is, the macro name ABC is the same as the macro name abc. The name can be the same as the two-letter name of a control word, in which case its definition supersedes the function of the control word. When you enter a macro name as part of your input file (after you have defined it), enter it as though it were a control word, with a period in column 1.

Local Symbols for Macros

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 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 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 or tokens that make up the character string. The symbol &*1 contains the first token, the symbol &*2 contains the second token, 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 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.

Processing Local Variables

Macros can contain conditional and iterative processing and can use local variables. You can use the .GO [Goto] control word to branch to another portion of a macro on certain conditions. For example, to process each token specified with a macro invocation separately you could enter:

```
.dm macro on
.se *1 = 1
...loop
.process &*&*i
.se *i = &*i + 1
.if &*i le 2 .go loop
.dm off
```

Terminating a Macro

Ordinarily, processing of a macro ends after the last line of the macro has been processed; control returns to the file or macro that invoked the macro.

The .ME [Macro Exit] control word can be used to end processing of a macro prematurely:

```
.dm score on
.sk 1
.if &place eq inline .me
.hr left to right
.sk 1
.dm off
```

If the value of the symbol &place is *inline*, the .ME control word causes control to return immediately to the macro's caller, without processing 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 then is processed as if it had been part of the calling macro or file. This allows a macro to set its caller's local symbols. For example,

```
.dm macro on
...
.me .se *rc = 4
.dm off
```

Here the .ME control word function of prematurely ending the macro is superfluous, because 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 on
.se *i = &*1 + 1
.if &*i gt &*0 .mg ||CASE index error.|
.el .me .go &*&*i
.dm off
```

The CASE macro can be invoked with an index number (0, 1, 2, or 3) and a list of labels:

```
.case &function open read write close
```

The CASE macro uses 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 on
.ec .sk &*
.il 5
.dm off
```

redefines the .SK [Skip] control word, to skip 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 (that is, to execute the control word rather than the macro).

When macro substitution is off (.MS OFF) and you want to execute a macro (whether or not the macro 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 *does not* modify the control word function; it is used, as a macro, *instead of* 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, if you want to redefine the .SK [Skip] control word to skip lines and indent to the first output line:

```
.dm sk on
.ms off
.sk &*
.ms on
.il 5
.dm off
```

Because we turned macro substitution off with the .MS OFF control word, the third line of the macro invokes the .SK control word, rather than reinvoking the .SK macro.

Sometimes turning off macro substitution is not an adequate solution to the problem of an endless loop. For example, you can cause the .IM [Imbed] control word to type the name of the imbedded file whenever it is imbedded by defining a .IM macro:

```
.dm im on
.ty Imbedding &* . . .
.ms off
.im &*
.ms on
.dm off
```

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 have the input line treated as a control word even though a macro of the same name might be defined. For example, the following lines

```
.dm im on
.ty Imbedding &* . . .
.ec .im &*
.dm off
```

redefine the .IM [Imbed] control word, preventing an endless loop while still allowing for macro substitution in the imbedded file.

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
.*
.dm fignum on
.se figctr = &figctr + 1
.se fig@&*1 = &figctr
.se fig#&*1 = &
Figure &fig@&*1..
.dm off
.*
.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 fred figure in the symbol &fig@fred
- Saves the page number of the fred figure 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 have 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 FIGNUM macro could have been defined like this:

```
.se figctr = 0
.*
.dm fignum on
.se figctr = &figctr + 1
.se *sx '?Figure &fig@&*1..? .?&fig#&*1..?
.dm figlist() |.sx &*sx
.se fig@&*1 = &figctr
.se fig#&*1 = &
Figure &fig@&*1..
.dm off
.*
.dm figref /Figure &fig#&*1 on page &fig#&*1
```

Notice that the FIGLIST macro is actually defined by the FIGNUM macro. You can use the other forms of the .DM control word within an inline form of .DM, but an inline form of .DM within an inline form will cause the initial macro definition to end.

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 that will format a figure number and page number.

Note: The lines of the FIGNUM macro that build the FIGLIST macro appear before the lines of the FIGNUM macro that set the symbols referred to 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 processes .SE [Set Symbol] control words that refer to 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

SCRIPT/VS has several implicit formatting functions. 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 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 processing the .LB [Leading Blank] control word when a leading blank is encountered, and by processing 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.

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 as:

```
.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], .LT [Leading Tab], and .BL [Blank Line] 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 processed.

Specifying a Macro Library

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 number if the user's 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
```

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 the SCRIPT/VS symbol table, so that a second occurrence of the macro or symbol does not require a library search. If no LIB option is specified, the symbol name or macro is searched for in the default library (if it exists).

When a macro name cannot be resolved (because there was no previous definition set with a .DM [Define Macro] control word), SCRIPT/VS can 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 can be members of the same library.[59]

You can use the macro library in two ways:

- To explicitly set a macro name. Use the LIB parameter of 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 .LY ON or .LY MAC have been specified.

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.

Because searching macro libraries for unresolved symbols is expensive in terms of processing time, we recommend 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.

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.

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.

When you are placing the definition of a macro into a macro library, be sure none of the lines of the macro is preceded by a .DM [Define Macro] control word.

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.

---

[59] 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.

A macro can be created or changed by editing a file with a file name 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 the *Virtual Machine Facility/SP: CMS User's Guide.*)

You can use the LIB option of the SCRIPT command to specify as many as eight macro library names.

```
script test ( lib (mylib yourlib)
```

The filetype for all of these libraries must be MACLIB. When an undefined macro is encountered, the libraries will be searched in the order specified on the LIB option of the SCRIPT command. If no library name is specified using the LIB option, a default name of DSMGML3 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.[60]

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.R30.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.

Because 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 before invoking SCRIPT. When doing this, you must use the file name (ddname) of SCRPTLIB. If you want SCRIPT.R30.MACLIB to be searched for macro definitions, you must include it in the concatenation when defining SCRPTLIB. Otherwise, it will not be searched. (For more information on concatenating libraries, see the *OS/VS2 TSO Terminal User's Guide.*)

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 can be concatenated in this manner, and SCRIPT.R30.MACLIB is not included in the concatenation. If the LIB option is not specified and a DDname of SCRPTLIB is not allocated, SCRIPT.R30.MACLIB is used.

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

---

[60] We recommend 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.

function can be used to delete members or list member names. Because 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.

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. Furthermore, the search is performed only against the permanent storage of the users whose operator numbers are specified in the LIB list.

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 and 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.[61]

```
script * ( lib (myopnum youropnum)
```

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

When you are going to access macros 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, rather than prodPARA and prodKEEP.

In a Batch Environment

In a batch environment, the Document Library Facility must be used to invoke SCRIPT/VS. Therefore, any macro libraries that are required for processing a document

---

[61] 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.

must be created as sequential data sets and brought into the Document Library Facility's Document Library by the IMPORT command before they can be accessed by SCRIPT/VS. For more information on how to use the IMPORT command to bring documents into the Document Library using the Document Library Facility, and how to access documents stored in the Document Library, see the *Document Library Facility Guide*.

Chapter 24. Processing GML

Generalized Markup Language (GML) can be used to describe the structure and elements of your document without regard to the particular processing that can 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. The Document Composition Facility actually provides two languages: 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; GML, on the other hand, is made up, basically, of adjectives that describe the structure and elements of a document.

The Document Composition Facility 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 implement 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. See the *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide* for more details on the starter set.

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.

Note: The *unmodified* GML starter set is a supported component of the Document Composition Facility. The GML starter set requires the Typewriter and Pi Specials (5771-AAW) font for the 4250 printer and the Pi and Specials (5771-ABC) font for the 3800 Printing Subsystem Model 3 and for the 3820 Page Printer.

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 can 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-tag 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 equal 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.

```
:h1. stitle='Programmer''s Guide'.DCF: Text Programmer's Guide
```

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 equal 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 is no room for complacency.
```

The line of text following the markup/content separator 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 can be null, if no text appears between the end of markup and the next tag. If there is text between the end of markup and the next tag, then the residual text is the first line of text. 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.".

Normally, residual text is formatted along with any other text following the markup. See "Residual Text Processing" on page 283 for more details on the treatment of residual text.

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 can 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 can 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.

Note: GML markup cannot span input files.

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 can be one or two characters. If it is a single character, it can 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 can 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 can be entered as:

```
<h1 id=gml>GML Support in SCRIPT/VS
```

SCRIPT/VS Processing of GML

This section describes the functions available in SCRIPT/VS to recognize GML markup and associate the tags and attributes with APFs.

Automatic GML Processing

When SCRIPT/VS processes a document and encounters a GML tag, the following processing sequence occurs:

1. Any attributes not processed by the previous tag are purged.
2. A search is made for an application processing function (APF) for the tag. This APF (written in the SCRIPT/VS formatting language and usually a macro) may be:
   - An APF associated with the tag by the .AA [Associate APF] control word
   - An APF with the same name as the tag.
3. The input is scanned for attributes and value attributes, if recognized by the tag, and saves them for processing
4. The residual text line is identified and saved.
5. The APF is invoked.

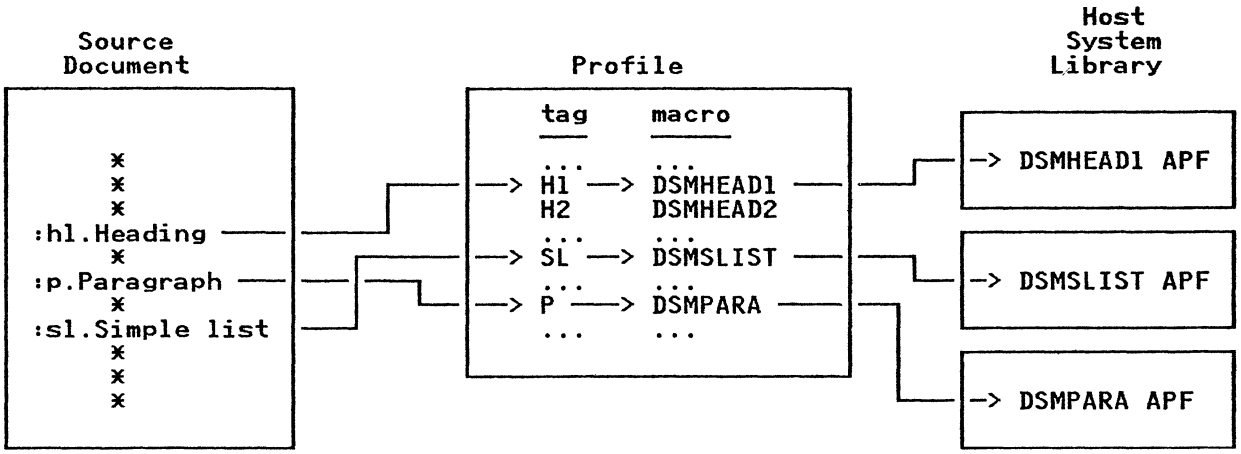


Figure 16. Processing Documents with GML: The profile provides the mapping between tags, which identify elements of text in the source document, and APFs, which provide formatting functions.

6. Any residual text is processed.

See Figure 16 for an illustration of how SCRIPT/VS processes documents containing GML tags.

Attribute Scanning Rules

The .GS [GML Services] RULES control word can 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
  - 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 can 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;[62] GML end-tags will not recognize attributes at all.

The attribute scanning rules for tags given with .GS RULES can be overridden for specific tags with the .AA [Associate APF] control word. For example, in the GML starter set,

---

[62] Invalid attributes are most commonly text, encountered when an optional markup/content separator has been omitted.

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

indicates that the APF for the *ol* tag is the DSMOLIST macro, and that value attributes are allowed for this tag. The APF for the *ol* end-tag is the DSMELIST macro and because no attribute scanning rules are specified, those given with .GS RULES for end-tags will be used.

The attribute scanning rules for .AA and .GS RULES are described in the SCRIPT/VS control word description section of the *Document Composition Facility: SCRIPT/VS Language Reference.*

Attribute Processing

Within the APF for a tag, for example,

```
:fig id=fred place=inline frame=box
```

the .GS [GML Services] EXATT control word, specified as,

```
.gs exatt
```

can be used to process all of the attributes that have been found and placed in the attribute stack.

The attributes are processed by APFs of the same names as the attributes. In other words, id is processed by the ID APF, place is processed by the PLACE APF, and frame is processed by the FRAME APF.

It is also possible to selectively process attributes. If the APF that processes the fig tag specifies

```
.gs exatt frame id as @idf
```

the frame attribute is processed by the FRAME APF and the id attribute is processed by the @IDF APF. The place attribute is not processed unless specified on another .gs exatt control word line.

The value of the attributes is provided to the APFs as parameters. For example, box is provided to the FRAME APF as the parameter &*1 and the value fred is provided to @IDF as &*1.

The APF for the fig tag can also specify

```
.gs exatt width
```

but because the width attribute was not specified with the tag, the width macro is not processed.

Attribute processing is described in further detail in the description of the .GS [GML Services] control word in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Value Attribute Processing

Value attributes are presented to the APF for the tag as the parameters &*1, &*2, and so on. The number of value attributes is provided in &*0.

If a simple list tag were encountered, for example,

```
:sl compact.
```

the value compact would be provided in &*1 to the APF that processes the SL tag. The APF could test whether this symbol was compact and then proceed accordingly.

In the following example,

```
.if /&U'&*1 eq /COMPACT
.th (do something)
.el (do something different)
```

the &U' symbol attribute was used to ensure that the uppercase form of &*1 would be used for comparisons because we do not know whether it was entered in upper- or lowercase. The symbol &*1 is a null symbol if the COMPACT attribute is not specified and it will cause an error if it is used alone on the IF control word, so it is prefixed here with a slash (/).

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 occurred in the middle of a word, the processing (such as starting a new font, for example) will not break the word. Because of this continuation, if the user is running with the SPELLCHK option, the first word of residual text may not be spellchecked.

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 is removed from the document and is placed in the symbol &line. If you do not want the residual text to be removed from the document but only want to have a copy of it placed in the symbol &line, you can specify

```
.gs copy line
```

When an APF explicitly retrieves the residual text, it is the APF's responsibility to provide continuation or other special treatment which can be required, such as turning on literal mode for the residual piece.

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.

GML scanning 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, the lines

```
.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,
```

If you want to insert text from an APF, use the INSERT parameter of the .GS control word. For example, if you specify

```
.gs insert Figure 7
```

then the text, Figure 7, will be inserted, using proper continuation, before any residual text that may exist for the tag associated with this APF.

The continuation of text that comes before and after a GML tag can be affected by other markup on the input line. For example if you entered,

```
.sp; text1 :tag text2
```

the .SP [Space] control word causes a break between *text1* and *text2*. The resulting output may appear as:

```
text1
text2
```

Any control words entered at the beginning of an input line that contains both text and GML tags, will have this effect.

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 can be a macro or a control word, can 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 is the FIGURE macro, and the APF for the fig end-tag is 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 281. The .AA control word is described in more detail in the *Document Composition Facility: SCRIPT/VS Language Reference*.

Class Mapping

A single character that will be added to the front of a tag name to produce an APF name can be specified with the .GS [GML Services] PREFIX control word. For example,

```
.gs prefix @
```

With this class mapping in effect, the APF for the *figlist* tag is 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.

Creating Your Own GML Tag

Before you create a tag of your own, you will need to:

- Identify the need for a tag
- Decide on a tag name
- Identify the tag's function
- Define an APF to process the tag
- Enable the tag.

Suppose, then, that you need a frequently used disclaimer that contains some constant and some variable text.

```
Any Similarity To Living Or Dead Persons
Of The Same Name Is Purely Coincidental.

For Legal Questions Contact: John Doe
```

In our example, the variable text is the name of the person to contact, the rest of the disclaimer will always be the same.

Your first step would be to give names to your tags.

- :disc - to start the structure
- :edisc - to end the structure.

The markup might look like this:

```
:disc. John Doe
:edisc.
```

Your second step is to define the APF for the :disc tag. This APF will:

- Get to the top of a page
- Turn on highlighting
- Start a box
- Insert some space
- Left align the text of the message
- Print the text of the message

It might look like this:

```
.dm lnote on
.pa nostart
.bf hi2
.bx 1 45
.sp 1
.in +2
.ir +2
.fo left
Any Similarity To Living Or Dead Persons
Of The Same Name Is Purely Coincidental.
.sp 1
For Legal Questions Contact:&rbl.&$cont.
.dm off
```

Note that the two symbols in the last line of our disclaimer text (&rbl. and &$cont.) are there to put a required blank between the text and the variable name to follow, to keep that variable name and the text on the same line.

Then the APF for the end tag should:

- Turn normal formatting back on
- Leave some more space

- Complete the box
- Turn off highlighting

It might look like this:

```
.dm elnote on
.fo
.sp 1
.bx off
.pf
.dm off
```

Your last step would be to integrate the tags into your document or into a profile that you created that then imbeds DSMPROF3 (the profile for the GML starter set that, among other things, turns on tag and macro processing, enables the macro library, and associates GML tags that you have enabled with the .AA [Associate APF] control word to their proper APFs). In the example, you would associate the tags :disc and :edisc like this:

```
.aa disc lnote elnote
```

Your complete tag description might look like this:

```
.*
.dm lnote on
.pa nostart
.bf hi2
.bx 1 45
.sp 1
.in +2
.ir +2
.fo left
Any Similarity To Living Or Dead Persons
Of The Same Name Is Purely Coincidental.
.sp 1
For Legal Questions Contact:&rbl.&$cont.
.dm off
.*
.dm elnote on
.fo
.sp 1
.bx off
.pf
.dm off
.*
.aa disc lnote elnote
```

With your tag complete, you can now specify

```
:disc. John Doe
:edisc.
```

and you will get the disclaimer

```
Any Similarity To Living Or Dead Persons
Of The Same Name Is Purely Coincidental.

For Legal Questions Contact: John Doe
```

printed at the top of your next page with the variable text (in this case, John Doe) added as a result of the processing of your tag.

Chapter 25. Verifying Spelling

SCRIPT/VS can automatically verify the spelling of words. When this function is activated, each word in your document will be checked for correct spelling. The SCRIPT/VS dictionaries, described later in this chapter, are used for spelling verification, and also hyphenation.

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.

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. If no .DL control word has been encountered, the default dictionary for your installation will be used. Spelling errors are listed with other errors found during formatting, using the .UW [Unverified Word] control word.

Whenever misspelled words are found in an input line, the .UW control word is executed with the misspelled words as parameters. 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. When 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.

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 you may want to write the words to a file to use later as an addition to your dictionary.

The following .UW macro will do both functions:

```
.dm uw on
.mg //Unverified Words: &*
.du add &*
.wf .du add &*
.dm off
.ms on
```

After you have formatted a document containing this macro with the SPELLCHK option of the SCRIPT command, the DSMUTWTF file will contain a list of all unverified words, prefixed with .DU ADD. This file must be edited to remove any truly misspelled words and can then be renamed and imbedded the next time the document is formatted to create an addenda dictionary.

For purposes of spelling verification, a *word* is a string of two to fifty-five characters delimited by *word delimiters*. The WORD parameter of the .DC [Define Character] control word specifies characters that are to be interpreted as word delimiters during spelling verification. The spelling of each string marked with word delimiter characters is separately verified. A list of the default word delimiter characters is in Figure 17 on page 291.

Characters given with WORD will be added to the current word delimiter set when ADD is specified; they will be deleted when DEL is specified.

The following example shows how the WORD parameter is used to separate individual words for spelling verification. A hyphen (-), for instance, is initially a punctuation character, so the term *in-laws* will be processed by spelling verification as a single word. However, if you enter

```
.dc word -
```

the hyphen will be treated as a word delimiter, and the term *in-laws* will be checked as two separate words.

The backslash (\) is initially neither a word delimiter nor a punctuation character, so the term *APL\360* will be verified as a single word. The backslash is not keyable on some terminals, but it can be identified as a word delimiter by entering its hexadecimal code on any terminal:

```
.dc word e0
```

The term *APL\360* will now be processed as two separate words.

The PUNC parameter of the .DC [Define Character] control word specifies characters that are to be recognized as punctuation for spelling checking. When punctuation characters occur within a word, they will be retained when the word is checked against the dictionary; if they occur at either end of a word, they will be removed before checking takes place.

The initial punctuation characters are the hyphen (-) and single quote ('). Characters given with PUNC will be added to the current set of pf punctuation characters if ADD is specified; they will be deleted if DEL is specified.

For example, the slash (/) is initially a word delimiter character, so when the term *SCRIPT/VS* is processed by spelling verification, *SCRIPT* and *VS* are checked separately as two different words. If you enter

```
.dc punc /
```

the slash will now be treated as a punctuation character, and the term *SCRIPT/VS* will be checked as a single word. If you enter the term */rubbish/*, the slashes will be removed, since they occur at the ends of the word, and the word *rubbish* will be verified.

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 addenda and main 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 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 synthesizes a .UW [Unverified Word] control word and executes it with all of the unverified words from a single input line.

Code	Character	Code	Character	Code	Character
05	Tab	4F	|	6F	?
11	Special Blank [1]	5A	!	7A	:
12	Special Blank [1]	5B	$	7E	=
13	Special Blank [1]	5C	*	7F	"
16	Backspace	5D	)	8B	{
40	Blank	5E	;	9B	}
41	Required Blank [2]	5F		A0	~
4A	¢	61	/	AD	[
4B	. (Period)	6B	,	AF	•
4C	<	6C	%	BD	]
4D	(	6D	_		
4E	+	6E	>		

[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. The code point assignment of the required blank can be changed with the .DC [Define Character] control word.

Figure 17. Characters that Delimit Words for Spelling Verification: The .DC [Define Character] control word can be used to make other characters word delimiters.

Note: Because stem processing (the process of removing the prefix and suffix of a word) 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 addenda and main 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
```

- No 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.

When spelling verification is performed, each occurrence of every word in the document being formatted is checked against the active dictionaries. This can result in a significant increase in the processor time required to format a document.

Often it is sufficient to perform spelling verification only twice: once, when the document is first created, to find entry errors, acronyms, and valid words that are not in the dictionaries; and, again, just before the final formatting runs, to catch any errors made while updating or revising the document.

Spelling 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* will both be verified after stem processing.

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.

Spelling verification may fail on the first word of a GML Tag's residual text due to possible continuation from a previous tag.

The SCRIPT/VS Dictionaries

There are three types of SCRIPT/VS dictionaries that are used for hyphenation and spelling verification:

- Read-only dictionaries of words provided by IBM with SCRIPT/VS. Each contains about 10,000 words. Because suffixes and prefixes are removed from a word, if necessary, 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. The algorithms used in building dictionaries require that they contain more than five words and that they are relatively dissimilar.
- Addenda dictionaries you create for a specific document using the .DU [Dictionary Update] control word. Addenda dictionaries contain words that are not in the main or user-created dictionaries but are frequently used in a specific document. 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 because it is rebuilt in storage every time SCRIPT/VS processes a document that requires it. An addenda dictionary can contain a maximum of 804 words.

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. If a dictionary is not specified, the default dictionary for your installation is used.

The hexadecimal code points for accented characters in the SCRIPT/VS spelling checking and hyphenation dictionaries are listed in Figure 18.

DUTCH UC/lc	CANFARDEINACNH UC/lc	FRENCH UC/lc	GERMAN UC/lc	ITALIAN UC/lc	SPANISH UC/lc	Character Name
/45 /44 /42 63/43	 64/44 62/42 	 /44 /42 	 63/43	 /44 	/45 	"A" Acute "A" Grave "A" Circumflex "A" Diaeresis (Umlaut)
/48	68/48	/48				"C" Cedilla
/51 /54 /52 73/53	71/51 74/54 72/52 73/53	/51 /54 /52 /53		/51 /54 	/51 	"E" Acute "E" Grave "E" Circumflex "E" Diaeresis (Umlaut)
   77/57	 76/56 77/57	 /56 /57			/55 	"I" Acute "I" Grave "I" Circumflex "I" Diaeresis (Umlaut)
					69/49	"N" Tilde
/CE /CD /CB EC/CC	 EB/CB EE/DF	 /CB EE/DF	 EC/CC	 /CD 	/CE 	"O" Acute "O" Grave "O" Circumflex "O" Diaeresis (Umlaut) "OE" Digraph
   FC/DC	 FD/DD FB/DB FC/DC	 /DD /DB /DC	 FC/DC	 /DD 	/DE /DC	"U" Acute "U" Grave "U" Circumflex "U" Diaeresis (Umlaut)
			59			Ess zet

Figure 18. Code Point Assignments for Accented Characters: Accented characters in the SCRIPT/VS Spelling Verification and Hyphenation dictionaries are represented using the hexadecimal code points shown under each language for uppercase (UC) and lowercase (lc) characters.

Building a User Dictionary

A user dictionary is created using the dictionary maintenance procedures that are described in Appendix E. of the *Document Composition Facility: SCRIPT/VS Language Reference.* 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 long and includes

- The appropriate hyphenation for the word
- 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 (using the .DL control word) to be accessible to SCRIPT/VS.

Because it is concatenated to the main dictionary, SCRIPT/VS treats it as part of the main dictionary. Therefore, whenever you specify, with the .HY [Hyphenate] or .SV [Spelling Verification] control words, 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. Refer to Appendix E in the *Document Composition Facility: SCRIPT/VS Language Reference* for information about creating and maintaining user dictionaries.

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 if it is other than the default dictionary at your installation. This will associate the addenda dictionary with the main dictionary for that language. For example,

```
.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.

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,

```
.du add Ty-pog-ra-phy
.du del Paul Ri-ver-front
```

adds the word *Typography* 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,

```
.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: More than one language can be used when processing a document. However, only one language can be active at a time.

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 ] )
```

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.
- If you want to use a user library and the default library, you must specify both on the TLIB option.

The TLIB option is valid only in CMS.

Searching a SCRIPT/VS Dictionary

Any time SCRIPT/VS encounters a word that needs to be hyphenated, it searches the SCRIPT/VS dictionary 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.

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 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.

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 will 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*.

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[63] prefixes that SCRIPT/VS checks during stem processing: 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
```

---

[63] 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: 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 processes 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				

German Prefixes

The following German prefixes are processed by SCRIPT/VS during stem processing.

ABER	ABTAST	ABSATZ	AB
ACHTUND	ACHT	AKZEPT	ALLGEMEIN
ALT	ANLAGE	ANLEIHE	ANLERN
ANTI	ANTRAGS	AN	AUFTRAG
AUF	AUSFUHR	AUSSEN	AUS
BAGATELL	BAHN	BANK	BAR
BAUSPAR	BEDARFS	BEI	BEREIT
BESCHLUSS	BESITZ	BESTELL	BETRIEBS
BILANZ	BINNEN	BRIEF	BRUTTO
BUCH	BUDGET	DAR	DA
DEFIZIT	DEPOT	DIENST	DIREKT
DISKONT	DOLLAR	DOPPEL	DREH
DREIUND	DREI	DRITTEL	DRITT
DRUCK	DURCHFUHR	DURCH	EIGENTUMS
EINFUHR	EINKAUFS	EINLAGE	EINNAHME
EINSATZ	EINSPRUCHS	EINUND	EINZEL
EIN	ELEKTRO	ENERGIE	ENTGEGEN
ERSATZ	ERWERBS	EUROPA	EXEKUTIV
EXISTENZ	EXPORT	FABRIK	FACH
FAHRT	FAHR	FEHL	FEIN
FERNMELDE	FERN	FEST	FILIAL
FINANZIELL	FINANZ	FLUGZEUG	FLUG
FLUT	FORT	FRACHT	FREI
FREMD	FRIST	FRUEH	FUENFUND
FUENF	FUERSORGE	FUNKTIONAERS	GANZ
GARANTIE	GAST	GAS	GEBAEUDE
GELD	GEMEINDE	GEMEIN	GENERAL
GEPAECK	GESAMT	GESETZ	GEWAEHR
GEWERBE	GEWERBS	GEWINN	GE
GIRO	GLEICH	GRENZ	GROSS
GRUENDSAETZ	GRUND	GUT	HAFT
HALBJAHRES	HALB	HAND	HAUPT
HAUS	HEIMAT	HEIM	HERAB
HERAN	HERAUF	HERAUSGE	HERAUS
HERBEI	HEREIN	HERNACH	HERUEBER
HERUM	HERUNTER	HERVOR	HER
HILFS	HINAB	HINAN	HINAUF
HINAUS	HINEIN	HINDURCH	HINUEBER
HINUNTER	HINWEG	HINZU	HIN
HOCHSCHUL	HOCH	HOECHST	HONORAR
HUNDERT	IMPORT	INDIVIDUAL	INDUSTRIE
INITIATIV	INNEN	INSOLVENZ	INTERZONEN
INVENTAR	INVESTIV	INVESTMENT	IN
JUGEND	JUSTIZ	KABEL	KAMPF
KAPITAL	KARTELL	KASSE	KAUF

KERN	KLAGE	KLEIN	KLINIK
KLIMA	KOLLEKTIV	KOMMUNAL	KOMPROMISS
KONJUNKTUR	KONKURRENZ	KONKURS	KONTO
KONTROLL	KONZERN	KRAFT	KREDIT
KURIS	KULTUR	KULTUS	KUNSTSTOFF
KUNST	KURS	KURZ	KUR
LANG	LEHR	LIZENZ	LOHN
LOS	LUFT	LUXUS	MACHT
MAGNET	MANAGEMENT	MARKT	MATERIAL
MEHR	MERK	METALL	MIET
MIKRO	MINDEST	MINIMAL	MISS
MITTEL	MIT	MODELL	MONOPOL
MONTAN	NACHFRAGE	NACHSCHUB	NACHWUCHS
NACH	NATIONAL	NETTO	NEUNUND
NEUN	NEU	NICHT	NOMINAL
NORD	NORMAL	NOT	NUTZ
OBEN	OBER	OPTIMAL	ORTS
OST	PACHT	PAKET	PARALLEL
PARTEI	PATENT	PAUSCHAL	PERSONAL
PFAND	PFLICHT	PLAN	PORTO
POST	PREIS	PRESSE	PRIVAT
PROBE	PROBLEM	PROFIT	PROJECT
PROTEST	PROZENT	PROZESS	PUNKT
QUER	RAND	RANG	RATS
REFORM	REGEL	REGIONAL	REIN
REISE	REKLAME	REPARATUR	REPRAESENTATIV
REPORT	RESERVE	REST	RISIKO
ROHSTOFF	ROH	RUECKWAERTS	RUECK
RUHEGELD	RUHE	RUNDFUNK	RUND
SACH	SALDO	SAMMEL	SCHATZ
SCHECK	SCHEIN	SCHLUSS	SCHNELL
SCHREIB	SCHRIFT	SCHUL	SCHUTZ
SECHSUND	SEE	SELBST	SICHT
SIEBENUND	SOFORT	SOLL	SONDER
SOZIAL	SPAR	SPERR	SPEZIAL
SPITAL	SPRACH	STADT	STAHL
STANDARD	STAMM	START	STIMM
STOER	STRAF	STRAHL	STREIK
STROM	STRUKTUR	STUECK	SUBSTANZ
SUED	SUPER	SYSTEM	TARIF
TAT	TAUSCH	TEIL	TELEFON
TELEGRAM	TENDENZ	TERMIN	TEST
TEXTIL	TON	TOTAL	TRANSPORT
TREUHAND	TREUHAENDER	UEBERNAHME	UEBERSCHUSS
UMLAUFS	UMSATZ	UMTAUSCH	UMWELT

```
UM              UNFALL          UN              UR
VALUTA          VERDIENST       VERGLEICHS      VERKAUFS
VERKEHR         VERLADE         VERLUST         VERMAECHTNIS
VERSAND         VERTRIEBS       VIEL            VIERTEL
VIERUND         VIER            VOLL            VORAN
VORAUF          VORAUS          VORJAHRES       VORSORGE
VORWAERTS       VOR             WACHSTUMS       WAHL
WANDEL          WECHSEL         WEG             WEHR
WEIHNACHTS      WELT            WERBE           WERT
WEST            WIDER           WOHL            WOHN
ZEHN            ZENTRAL         ZIEL            ZINS
ZIVIL           ZOLL            ZURUECK         ZUSATZ
ZUSCHUSS        ZUWACHS         ZU              ZWECK
ZWEIG           ZWEIUND         ZWEI
```

Spanish Prefixes

The following Spanish prefixes are processed by SCRIPT/VS during stem processing.

CONTRA	RECOMP	PRES#	PRES$	SUBS#
SUBS$	TRANS	ANTI	COMP	CONP
PRE>	REM#	REM$	REN#	REN$
RES#	RES$	CON	DES	IMB
IMP	INB	INP	IRR	PRE
RE>	SUB	IN	RE	

Part 4. Appendixes

This part of the book contains additional information about SCRIPT/VS. Included in this section are the following appendixes:

- Appendix A - Using SCRIPT/VS with Other Programs
- Appendix B - Improving System Performance.

Appendix A. 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.

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 19 on page 311.

Specifying STAIRS/VS Output

Use the DEVICE option of the SCRIPT command to specify STAIRS/VS output. (For details, refer to the description of the DEVICE command option in the *Document Composition Facility: SCRIPT/VS Language Reference.*)

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. These blocks are *fixed* length records of 1008 bytes and they must be directed to either an F (fixed-length records) or FB (fixed-length, blocked records) type data set. The use of a V (format-V records) or VB (variable-length, blocked records) data set may cause a system error.

If you specify DEVICE(STAIRS) and PRINT, FILE, or TERM, the document is written to the specified destination for proofreading.

Restrictions Imposed on Formatted Output

The STAIRS/VS program recognizes only the text in the body of a document. Consequently, running 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.

Whenever a paragraph has a paragraph identifier that is equal to or lower than the previous paragraph identifier, STAIRS/VS will create a new logical document. This is referred to as *stepdown.* This may or may not be the result of a user error. In either case, SCRIPT/VS will flag it as an error in *proofing* mode output as described below.

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 are flagged when output is being prepared in *proof* format, and are 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.
```

A line is considered to be a sentence delimited by *full stops*: a full stop character followed by two blanks. As described in "Chapter 3. Marking Up Documents with SCRIPT/VS" on page 37, 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 Identification

STAIRS/VS requires that a 3-character identifier be associated with each paragraph of a document placed in its data base.

STAIRS/VS paragraph identifiers 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:

```
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
```

causes 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 that 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 can also be provided with the .SO [STAIRS/VS Output] control word.

Offset Dec.	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

* CTF record blocks are not cleared or blanked out. The record length must be read in order tc determine the end of the valid data.

Figure 19. STAIRS/VS Condensed Text Format (CTF) Records: Each record has a fixed length of 1008 bytes and contains up to 13 lines of text.

The ATMS Conversion Routine

The ATMS-to-SCRIPT/VS conversion routine is composed of three separate element

- The conversion program, which runs as a processor under the control of the Do ment Library Facility. This processor scans ATMS documents for ATMS-II ATMS-III formatting controls and substitutes SCRIPT/VS symbols that inv similar or equivalent formatting functions.
- The ATMS conversion profile, DSMATMS3, which is used when invok SCRIPT/VS to format documents that were converted from ATMS. The profile fines to the formatter the substitutions required for the symbols generated by conversion program.
- The library of SCRIPT/VS macros that are used to emulate the original AT functions.

The conversion routine is designed to convert most ATMS controls, GML tags, and i plicit keying conventions to similar or equivalent SCRIPT/VS symbols. The output the conversion routine can then be formatted using DSMATMS3 and the library SCRIPT/VS macros supplied to emulate the original ATMS functions.

Some functions in ATMS are not directly convertible. Editing of the document may 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
- The r option of the lde and ldr controls.

In many of these areas the most noticeable difference is that the SCRIPT/VS equivalent of the ATMS function can cause a line break.

onversion 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 the *Document Library Facility Guide* for details.

yphenating 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.

ɔnversion Program Operation

The ATMS file(s) in FT00 format can 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.

on-Format Command Conversion

The following describes the conversion of each ATMS nonformatting control.

ıd of Imbedded 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 refer to the macros through the ATMS !m syntax. To accomplish this, the subdocuments containing the macros can 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(DSMATMS3) SYSVAR(A SUBDOC))
```

The IBM-supplied ATMS profile document (DSMATMS3) 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 before the start of formatting of the primary document. This limit of 10 names can be changed by the user by altering DSMATMS3 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 DSMATMS3.

The conversion macros are defined in DSMATMS3 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 recognizes these conditions in !tf (formatted mode) and inserts 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 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 (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 cannot 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 can not be used in split text lines.

Triplets and Backspaces

In ATMS there is an entry convention involving backspaces for characters that do not occur on the keyboards but that 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 converts 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 20 on page 316.

Hexadecimal Code	Character	Hexadecimal Code	Character
4A		&X'4a	
4C		&X'4c	
4F		&X'4f	
5F		&X'5f	
6E		&X'6e	
8B		&X'8b	
8C		&X'8c	
8F		&X'8f	
9B		&X'9b	
9F		&X'9f	
AB		&X'ab	
AC		&X'ac	
AE		&X'ae	
AF		&X'af	
B0		&X'b0	
B1		&X'b1	
B2		&X'b2	
B3		&X'b3	
B4		&X'b4	
B5		&X'b5	
B6		&X'b6	
B7		&X'b7	
B8		&X'b8	
B9		&X'b9	
BB		&X'bb	
BC		&X'bc	
BE		&X'be	
BF		&X'bf	

Figure 20. Character Codes Recognized by ATMS-III Conversion: The triplet (character-backspace-character) conventions for special characters defined in the *ATMS-III Terminal Operations Guide* are recognized and translated into a single hexadecimal character.

ATMS Control - SCRIPT/VS Symbol Relationship

Figure 21 on page 317 identifies the ATMS controls and the SCRIPT/VS symbols to which they are converted. The substitution for the SCRIPT/VS symbols is contained in DSMATMS3 and should be looked at in conjunction with this list. The contents of each macro that is eventually invoked by the substitution is contained in DSMATMS3 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.
textl!lst;xtext2	&@LST @textl@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
!tfnl;n2;n3;n4	&@TF nl n2 n3 n4
!tfe	&@TFE
!thh	&@THH
!thm;n	&@THM n
!tib;nl;n2	&@TIB nl n2
!tif	&@TIF
!til;nl;n2;n3	&@TIL nl n2 n3
!tir;nl;n2;n3	&@TIR nl n2 n3
!tj	&@TJ
!tle	&@TLE
!tls	&@TLS
!tm;nl;n2	&@TM nl n2
!tnj	&@TNJ
!tnp	&@TNP
!tpd;nl;n2	&@TPD nl n2
!tps;nxx	&@TPS Nxx
!trs;n	&@TRS n
!tset;id;val;style	&@TSET ID val STYLE
!tss	&@TSS
!ttab;nl;...;nm	&@TTAB nl ... nm
!ttab-;nl;...;nm	&@TTABM nl ... nm
!ttab+;nl;...;nm	&@TTABP nl ... nm
!tts	&@TTS
!tu	&@TU
!tuc	&@TUC
!tufnn	&@TUFnn
!tufcnn	&@TUFCnn
!tuhnn	&@TUHnn
!tuhcnn	&@TUHCnn
!tw;nl;n2	&@TW nl n2
!twz;n	&@TWZ n
!x	null

Figure 21. ATMS-III Controls to SCRIPT/VS Conversion

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

You can use SCRIPT/VS as a preprocessor if 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.[64]

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 that 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.

For example, the GML starter set APFs for ordered lists and list items automatically generate numbers (or letters) for the items in an ordered list. This is convenient, because it permits the list to be revised without renumbering all the items.

You can create a modified version of the APFs that 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

---

[64] See "Chapter 24. Processing GML" on page 277 for details about profiles, APFs, and mapping tags to APFs. See "Chapter 21. Processing Symbols" on page 223 for details about symbols, and "Chapter 23. Processing Macros" on page 261 for details about macros.

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 that 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
.wz off
.fo off
```

You might also need to translate special characters that 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. Among these are symbols used in the following ways:

- As abbreviations for lengthy character strings
- As references to generated information that is not format-dependent (such as a figure or section number -- but not a page number)
- To enter unkeyable characters that 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 can 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 and carriage control characters 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.

Note: The .OC [Output Comment] control word is ignored for page printers.

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 the SCRIPT command options section of the *Document Composition Facility: SCRIPT/VS Language Reference.*

Although 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 can require record lengths and formats, or other characteristics, which differ from those produced by SCRIPT/VS. You might have to use a utility program, or code your own, to handle such interface requirements.

Appendix B. Improving System Performance

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, and 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 verification
- 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 that are set later in the document.[65]

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 can 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 checked against the active dictionaries.

---

[65] These are called *forward references*, because the value of the symbol is used before the symbol is defined.

Because spelling verification significantly increases the processor time required to format a document, it should be used only occasionally. Often it is sufficient to perform spelling verification only twice: once, when the document is first created, to find entry errors, acronyms, and valid words that 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 significant amounts of both virtual storage and processor time, because the index entries are kept in storage and sorted dynamically.

Because an INDEX is often not needed for draft copies of a document, the INDEX option can simply be omitted; the index terms specified in the document will be ignored.

SCRIPT/VS in the ATMS-III Environment

In the ATMS-III environment, CICS provides facilities for creating and editing documents, and SCRIPT/VS can be used to format these documents. If you are an ATMS user, you can invoke SCRIPT/VS to format your documents in any one of three ways:

- On-line formatting: SCRIPT/VS can 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 can then be transferred to a printer or reviewed at your terminal.
- Using a peripheral queue: Requests can 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 can 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 can be used to regulate SCRIPT/VS in the CICS environment. The parameters 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 that can be obtained per document, beyond an initial 64K byte block.

SPP Limits the number of output pages that can 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 that can 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 that can be produced when formatting documents on-line from a terminal. TSP gives the number of pages permitted.

Further information on tuning ATMS-III in the CICS environment can be found in the *ATMS-III Program Reference Manual.*

Glossary

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 can 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.*

API: See *application programming interface.*

application processing function (APF): In GML processing, the processing that is performed when a document element or attribute is recognized. In SCRIPT/VS, an APF is implemented as a sequence of control words, possibly intermixed with text and symbols, in one of three forms: macro definition, value of a symbol, or imbedded file.

application programming interface (API): An external, published interface that can be programmed to by another application.

attribute: A characteristic of a document (or document element) other than its type or content. For example, the security level of a document or the depth of a figure.

attribute label: In GML markup, a name of an attribute that is entered in the source document when specifying the attribute's value.

back matter: In a book, those sections (such as glossary and index) that are placed after the main chapters or sections.

balancing: In multicolumn formatting, the process of making column depths on a page approximately equal by re-distributing the text in the columns. See also *vertical justification.*

baseline: An imaginary horizontal line upon which most of the letters in a line of text appear to rest.

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.

body: (1) Of a printed page, that portion between the top and bottom margins that contains the text. (2) Of a book, that portion between the front matter and the back matter.

boldface: A heavy-faced type. Also, printing in this type.

bottom margin: On a page, the space between the body or the running footing, if any, and the bottom edge of the page.

break: An interruption in the formatting of input lines so that the next input line is printed on a new output line.

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 space: The horizontal size of a character. This size depends upon which font the character is from and on which physical device the character is printed.

character spacing: The space between characters in a word.

cicero: In the Didot point system, a unit of 0.1776 inch (4.512 millimeters) used in measuring typographical material.

CMS: An interactive processor that operates within VM/370.

code page: A font library member name that gives the association between code points and the character names of a font.

code point: An eight-bit binary code representing one of 256 possible characters.

coded font: (1) The combination of a code page and a font library. (2) A font that is fully described in terms of typeface, point-size, weight, width, and attribute.

column balancing: The process of redistributing lines of text among a set of columns so that the amount of text in each column is as equal as possible.

column width: The width of each text column on a page. Specified with the .CL [Column Line Length] control word. (In multicolumn formatting, all columns on a page usually have the same width.)

command: A request from a terminal or specified in a batch processing job for the performance of an operation or the execution of a particular program. For example, a request given at a terminal for SCRIPT/VS to format a document or for an editor to edit a line of text.

comment: A control word line that is ignored by SCRIPT/VS. Such lines begin with either .* or .cm.

composed text: Text that has been formatted and that contains control information to direct the presentation of the text on page printers.

compositor: A person or program that composes text.

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.

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.

destination: The physical device to which data is sent.

dictionary: A collection of *word stems* that is used with the spelling verification and automatic hyphenation functions.

Didot point system: A standard printer's measurement system on which type sizes are based. A Didot point is 0.0148 inch (0.376 millimeter). There are 12 Didot points to a cicero. See also *cicero* and *point.*

document: (1) A publication or other written material. (2) A machine-readable collection of lines of text or images, usually called a source document. See also *output document* and *source document.*

document conversion processor: A computer program that processes a machine-readable document that includes formatting controls written in one formatter language, to produce a machine-readable document that includes formatting controls appropriate for another formatter language.

document library: A set of VSAM data sets, accessible in a batch environment, which contain documents and related files.

dot leader: A set of periods that fills in the space between two pieces of split text such as a chapter title and its page number in a table of contents.

duplex: A mode of formatting appropriate for printing on both sides of a sheet.

EBCDIC: Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

edit: To create or modify the contents of a document or file. For example, to insert, delete, change, rearrange, or copy lines.

editor: A computer program that processes commands to enter lines into a document or to modify it.

eject: In formatting, a skip to the next column or page.

em: A unit of measure usually equal to the width or the height of the character "m" in a particular font.

en: A unit of measure usually equal to one-half the width of an em. For many typefaces, lower case characters tend to average the width of an en.

epifile: The second portion of a profile (after a .EF control word) that is processed *after* a main document has been processed.

escapement: The space unit of movement (either vertical or horizontal) that is built into a physical device. For the 1403 printer with a 10-pitch train, the horizontal escapement unit is 1/10th of an inch; for the 4250 printer, that value is 1/600th of an inch; and for the 3800 Printing Subsystem Model 3 and the 3820 Page Printer, that value is 1/240th of an inch.

extended symbol processing: The processing of a symbol whose value causes the remainder of the line to be stacked and later processed as a new input line.

factor: A dimensionless scalar value used to form a product with another value. Factors can also be expressed as percentages.

figure space: (1) The width of the figure zero (0) is commonly used as the figure space of a given typeface. This is the definition of figure space as it is used in the Document Composition Facility. (2) A unit of measure equal to the width of the "en" space in a particular font.

fill character: The character that is used to fill up a space; for example, blanks used to fill up the space left by tabbing.

float: (1) (noun) A keep (group of input lines kept together) whose location in the source file can vary from its location in the printed document. (2) (verb) Of a keep, to be formatted in a location different from its location in the source file.

flush: Having no indention.

fold: (1) To translate the lowercase characters of a character string into uppercase. (2) To place that portion of a line that does not fit within a column on the next output line.

font: 1) An assortment of type, all of one size and style. 2) A font library member that contains characters that must be used in conjunction with a code page font library member.

font object: Refers to a member of a font library. In CMS, a font object is a file whose filetype matches the name of the font library. In MVS, a font object can be a member of a partitioned data set (PDS).

font set: The set of fonts to be used in formatting a source document.

footing: Words located at the bottom of the text area. See also *running footing*.

footnote: A note of reference, explanation, or comment, placed below the text of a column or page, but within the body of the page (above the running footing).

foreground: The environment in which interactive programs are executed. Interactive processors reside in the foreground.

format: (1) (noun) The shape, size, and general makeup of a printed document. (2) (verb) To prepare a document for printing in a specified format.

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 can 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.

hanging indention: The indention of all lines of a block of text following the first line (which is not indented the same number of spaces). Specified with the .OF [Offset] or .UN [Undent] control word.

head-level: The typeface and character size associated with the words standing at the beginning of a chapter or chapter topic.

heading: Words located at the beginning of a chapter or section or at the top of a page. See also *head-level* and *running heading.*

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.*

horizontal justification: The process of redistributing the extra horizontal white space at the end of a line of text in between the words and letters of the line so as to exactly fill the width of the column with the text.

impact printer: A printer, such as the 1403 and the 3211, in which printing is the result of mechanical impacts.

indent: To set typographical material to the right of the left margin.

indention: The action of indenting. The condition of being indented. The blank space produced by indenting. Specified with the .IN [Indent], .IR [Indent Right], .UN [Undent], .OF [Offset], and .IL [Indent Line] control words. See also *hanging indention.*

initial caps: Capital letters occurring as the first letter of each word in a phrase. To set a phrase in initial caps is to capitalize the first letter of each word in the phrase.

initial value: A value assumed by SCRIPT/VS for a formatting function until the value is explicitly changed with a control word. The *initial value* is assumed even before the control word is encountered, whereas the *default* value is assumed when the control word is issued without parameters. See also *default value.*

inline space: An amount of horizontal white space in a line that usually occurs between two words.

input device: A machine used to enter information into a computer system (for example, a terminal used to create a document).

input line: A line, as entered into a source file, to be processed by a formatter.

interactive: Pertaining to an application in which entries call forth a response from a system or program, as in an inquiry system. An interactive system might also be conversational, implying a continuous dialog between the user and the system. Interactive systems are usually communicated with via terminals, and respond immediately to commands. See also *foreground.*

interactive environment: The environment in which an interactive processor operates.

intercharacter space: Extra horizontal white space inserted *between* characters of a word. This space is in addition to the space included as part of the characters by the designer of the font.

interword space: See *word space.*

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.

justification: See *horizontal justification* and *vertical justification*)

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.

ligature: A single character (piece of type or font raster) that represents two or more input characters: ff and ffi are examples of characters that may be represented by (printed as) a ligature.

line device: Any of a class of printers that accept one line of text from the host system at a time.

SCRIPT/VS supports such line devices as the 1403, 2741, and 3800.

line space: The vertical distance between the baseline of the current line and the baseline of the previous line.

line spacing: See *line space.*

logical output device: The combination of a physical output device and such logical variables as page size and number of lines per vertical inch (for line devices). A specification of 1403W6 is an example of a logical output device.

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.

mark up: (verb) (1) To determine the markup for a document. (2) To insert markup into a source document.

markup: (noun) Information added to a document that enables a person or system to process it. Markup can describe the document's characteristics, or it can specify the actual processing to be performed. In SCRIPT/VS, markup consists of GML tags, attribute labels and values, and control words.

nonimpact printer: A printer, such as the 3800 Printing Subsystem, in which printing is not the result of mechanical impacts, but is instead produced by another process such as laser beam, ink-jet, or electro-erosion. The 3800 Printing Subsystem, for example, uses a laser based technology and the 4250 printer uses an electro-erosion process.

object.: A sequential collection of control records that represent documents, pages, fonts and so on.

offset: (verb) To indent all lines of a block of text, except the first line. (noun) The indention of all lines of a block of text following the first line.

option: Information entered with 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 that have been formatted, or otherwise processed, by a document processor. The output document can be printed or it can be filed for future processing.

output line: A line of text produced by a formatter.

page printer: Any of a class of printers that accept composed pages, constructed of composed text and images, among other things. SCRIPT/VS supports such page printers as the 4250 printer, the 3800 Printing Subsystem Model 3, and the 3820 Page Printer.

page segment: See *segment.*

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 3800 Printing Subsystem and the 4250 printer. On the 3800 Printing Subsystem Model 1, one pel equals approximately 1/180th inch. On the 3800 Printing Subsystem Model 3 and the 3820 Page Printer, one pel equals approximately 1/240th inch. On the 4250 printer, one pel equals approximately 1/600th inch.

physical output device: A physical device, such as a terminal, a disk file, a line printer, or a nonimpact printer. The 1403 printer is an example of a physical output device.

pica: A unit of about 1/6 inch used in measuring typographical material. Similar to a cicero in the Didot point system.

pitch: A number that represents the amount of horizontal space a font's character occupies on a line. For example, 10-pitch means 10 characters per inch, or each character is 0.1 (1/10) inch wide. 12-pitch means 12 characters per inch, and 15-pitch means 15 characters per inch.

point: (1) A unit of about 1/72 of an inch used in measuring typographical material. There are twelve points to the pica. (2) In the Didot point system, a unit of 0.0148 inch. There are twelve Didot points to the cicero.

profile: (1) In SCRIPT/VS processing, a file that is imbedded before the primary file is processed. It can be used to control the formatting of a class of source documents. When processing GML markup, the profile usually contains the mapping from GML to APFs and the symbol settings that define the formatting style. (2) In the 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: (1) A straight horizontal or vertical line used, for example, to separate or border the parts of a figure or box. (2) A solid black rectangle of a given width, extending horizontally across the column or vertically down the column.

running footing: A footing that is repeated above the bottom margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the body of the page (text area).

running heading: A heading that is repeated below the top margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the body of the page (text area).

section: When an output page has two or more single-column parts with the same or different column-widths, or a single-column part and a multicolumn part, or two or more different multicolumn parts, each part of the output page is called a section.

segment: An object containing composed text and images, prepared before formatting and included in a document when it is printed.

set size: The set size of a given typeface determines the number of characters that will fit in a line of a given width when it is printed or set.

small caps: Capital letters in the same style as the normal capital letters in a font, but approximately the size of the lowercase letters.

source document: A machine-readable collection of lines of text or images that is used for input to a computer program.

space: A blank area separating words or lines.

structured field: A self-identifying string of bytes, analogous to a logical record. A structured field consists of an introducer, which identifies and characterizes the structured field, and data or parameters.

symbol: A name in a source document that can be replaced with something else. In SCRIPT/VS, a symbol is replaced with a character string. SCRIPT/VS can interpret the character string as a number, a character string, a control word, or another symbol.

symbol substitution: During formatting, the replacement of a symbol with a character string that SCRIPT/VS can interpret as a value (numeric, character string, or control word) or as another symbol.

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) that is entered in the source document to identify it. For example, *:p.* might be the tag used to identify each paragraph.

terminal: A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

text line: An input line that contains only text.

token: A string of characters that is treated as a single entity. In SCRIPT/VS, a parameter passed to a macro in one of the local variables &*1, ... &*n.

top margin: On a page, the space between the body or running heading and the top edge of the page.

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.

typeface family: A collection of fonts of a common typeface that vary in size and style.

typeset: (1) (verb) To arrange the type on a page for printing. (2) (adjective) Pertaining to material that has been set in type.

type posture: A typeface style variation indicating whether a typeface is upright (as in roman) or slanted to the right (as in italic or cursive).

type size: The vertical height (point size) of a given typeface, such as 10 point.

type style: Style variations in a typeface. Among these variations are posture, weight, and width.

type weight: The relative thickness of the strokes of a typeface. Usually described in such terms as light, demi bold, bold, and so on.

type width: The horizontal size (set size) of a given typeface. The width may be given in units of measurement, such as set 9 point, or it may be descriptive: ultra condensed, condensed, expanded, and so on.

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.

unit space: The minimum amount of additional spacing acceptable for purposes of horizontal justification, as specified by the font designer.

uppercase: Pertaining to capital letters, as distinguished from small letters; for example, A, B, G rather than a, b, g.

vertical justification: The process of redistributing the extra vertical white space at the end of a column in between the lines of text, so as to make each column in a set appear to be equal in depth.

widow: One or two lines or words at the end of a paragraph that are printed separately from the rest of the paragraph.

word space: The horizontal white space placed between words. This is sometimes referred to as an interword blank.

word spacing: The space between words in a line. See also *word space.*

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

3

4

Document Composition Facility:
SCRIPT/VS Text Programmer's Guide Release 3
Order No. SH35-0069-2

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

You may use this form to communicate your comments about this publication, its organization, or subject matter with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or the IBM branch office serving your locality.

	Yes	**No**
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- How do you use this publication:

As an introduction to the subject?	☐
For advanced knowledge of the subject?	☐
To learn about operating procedures?	☐
As an instructor in class?	☐
As a student in class?	☐
As a reference manual?	☐

- What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

Attention: Information Development
Department 580

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Products Division
P. O. Box 1900
Boulder, Colorado 80301

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM

Document Composition Facility: SCRIPT/VS Text Programmer's Guide Release 3 Printed in U.S.A. SH35-0069

Document Composition Facility
SCRIPT/VS Text
Programmer's Guide

SH35-0069-2

File Number S370-20
Printed in U.S.A.

SH35-0069-2

IBM®