SH20-9161-0

Program Product

Document Composition Facility: User's Guide

IBM

SH20-9161-0

Program Product

Document Composition Facility: User's Guide

Program Number 5748-XX9

IBM

The source for this document was marked up with GML tags and SCRIPT/VS control words, and formatted with SCRIPT/VS (program number 5748-XX9) under the CMS component of VM/370. The output was transmitted to OS/VS2 MVS via the VM/370 Networking PRPQ (program number 5799-ATA), and printed on the IBM 3800 Printing Subsystem. The fonts used were GT12 and GB12, which are provided with SCRIPT/VS for use with the 3800 Printer.

First Edition (July 1978)

This is the first edition of a new publication that applies to the IBM Document Composition Facility program product, program number 5748-XX9.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using this publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

© Copyright International Business Machines Corporation 1978

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

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

Use of SCRIPT/VS in a TSO or CMS environment requires the Foreground Environment Feature (feature number 6043 or 6044); use in a background environment requires the Document Library Facility program product (program number 5748-XXE).

Information on the background environment is for planning purposes only until the Document Library Facility is available.

The chapters of this publication contain:

- "Chapter 1. An Introduction to SCRIPT/VS" on page 1: A general description of SCRIPT/VS. This chapter includes a discussion of what SCRIPT/VS is and what it does.
- "Chapter 2. Using the SCRIPT Command" on page 13: A description of the SCRIPT command and each of its options.
- "Chapter 3. Basic Text Processing" on page 29: A description of how to specify basic text formatting functions, such as indention, tabs, blank space, forcing a new page, and formatting modes. This chapter also describes some guidelines for entering input lines in a SCRIPT/VS file.
- "Chapter 4. Defining a Page Layout" on page 51: A description of how to define the parameters of a page, such as page length, column width, line length, and page numbering. This chapter also describes how to have text repeated at the top and bottom of each page: running titles, headings, and footings.
- "Chapter 5. Multicolumn Page Layout" on page 65: A description of how to establish a multicolumn format for the body of a page.
- "Chapter 6. Head Levels and Table of Contents" on page 71: A description of how to specify and modify SCRIPT/VS head levels (that is, chapter and topic headings), and how SCRIPT/VS creates a table of contents from the head levels.
- "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77: A description of additional formatting features of SCRIPT/VS, including character translation, keeping text together, marking revised material, drawing boxes, and using fonts with the IBM 3800 Printing Subsystem. This chapter also describes footnotes, and conditional column and page ejects.
- "Chapter 8. The SCRIPT/VS Formatting Environment" on page 97: A description of the SCRIPT/VS formatting environment.
- "Chapter 9. Conditional Processing" on page 99: 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 10. Combining SCRIPT/VS Files" on page 105: A description of how to combine SCRIPT/VS input files, merge input from several files, and use SCRIPT/VS to interactively create an output document.
- "Chapter 11. Symbols in Your Document" on page 117: A description of the SCRIPT/VS symbol processing capability: how to name symbols, store them in a symbol library, use system symbols, and use symbol arrays. This chapter describes many useful applications for symbols.
- "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137: A description of the SCRIPT/VS macro processing capability: how to build a macro, use symbols within a macro, conditionally process parts of the macro, and store macros in a macro library.

- "Chapter 13. GML Support in SCRIPT/VS" on page 145: A description of how to name a GML tag, build an Application Processing Function (APF) associated with the tag, and map the tag to the APF. This section should be read in conjunction with the Document Composition Facility: Generalized Markup Language (GML) User's Guide.

- "Chapter 14. Using SCRIPT/VS with Other Programs" on page 153: A description of how to use SCRIPT/VS with other text processing programs, either as a post-processor used to format the output of another program, or as a preprocessor used to prepare SCRIPT/VS files for input to another text processing program.

- "Chapter 15. Automatic Hyphenation and Spelling Verification" on page 157: A description of the SCRIPT/VS dictionary and how SCRIPT/VS uses it for automatic hyphenation and spelling verification. This chapter also describes how to build an addenda dictionary, used to supplement the SCRIPT/VS dictionary.

- "Chapter 16. Diagnostic Aids" on page 163: A description of how to identify errors in your input file and correct them. This chapter includes a description of the SCRIPT/VS input substitution trace facility, which enables you to observe the results of SCRIPT/VS processing at various points as your input file is being processed.

- "Chapter 17. EasySCRIPT" on page 171: A description of EasySCRIPT functions and usage.

- "Chapter 18. Compatibility with SCRIPT/370" on page 177: A description of the similarities and differences between SCRIPT/VS and SCRIPT/370.

- "Chapter 19. ATMS-II Conversion" on page 187: A description of the ATMS-II Conversion program provided with SCRIPT/VS for use with the Document Library Facility.

- "Chapter 20. Compatibility with TSO/FORMAT" on page 197: A description of the similarities and differences between SCRIPT/VS and TSO/FORMAT.

- "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199: A detailed description of each SCRIPT/VS control word: its format, parameters, usage notes, and examples of use.

- "Appendix A. SCRIPT/VS Summary" on page 297: A summary of SCRIPT/VS: file names, SCRIPT command parameters, control words, system symbols, special characters, character sets, and 3800 Printer fonts.

- "Appendix B. Device and Font Table Maintenance" on page 323: A description of how to define a new logical output device or new font to SCRIPT/VS.

- "Appendix C. Fonts Supplied with SCRIPT/VS" on page 329: An illustrated list of the fonts provided with SCRIPT/VS for use with the 3800 Printer

- "Appendix D. Formatting Considerations for the 3800 Printer" on page 337: A description of the use of SCRIPT/VS with the 3800 Printer.

REQUIRED PUBLICATIONS

- Document Composition Facility: General Information, GH20-9158. This manual provides a general description of the SCRIPT/VS program product and supplies a summary of its functions and capabilities. It contains the requirements for installation as well as the storage estimates.

- Document Composition Facility: Generalized Markup Language (GML) User's Guide, SH20-9160. This manual is for all users of the Document Composition Facility. It describes the use of the starter set of Generalized Markup Language tags provided with SCRIPT/VS, how to analyze documents in preparation for the creation of new GML tags and APFs, and how to use the Document Composition Facility to process them.

RELATED PUBLICATIONS

- Document Library Facility: General Information, GH20-9163. This manual introduces the related program product Document Library Facility.

- Document Library Facility Guide. 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 an ATMS-II data set into a SCRIPT/VS input file.

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

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

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

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

  - IBM Virtual Machine Facility/370: CP Command Reference for General Users, GC20-1820.

  - IBM Virtual Machine Facility/370: CMS User's Guide, GC20-1819.

  - IBM Virtual Machine Facility/370: CMS Command and Macro Reference, GC20-1818.

  - IBM Virtual Machine Facility/370: Terminal User's Guide, GC20-1810.

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

  - OS/VS2 TSO Command Language Reference, GC28-0646.

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

- Introducing the IBM 3800 Printing Subsystem and Its Programming, GC26-3829. This manual provides general information about the 3800 Printer. It describes what the 3800 Printer is and provides information about the standard and optional features available for the 3800 Printer. Another manual that includes detailed information about programming for the 3800 Printer is:

  - IBM 3800 Printing Subsystem Programmer's Guide, GC26-3846.

Chapter 1. An Introduction to SCRIPT/VS 1
Generalized Markup Language 3
How SCRIPT/VS Works 3
Control Words and Their Parameters 3
Defaults and Initial Settings 4
SCRIPT/VS Input File Characteristics 4
Logical and Physical Output Devices 4
Vertical and Horizontal Space Units 5
Fonts (IBM 3800 Printing Subsystem Only) 6
Calling the SCRIPT/VS Processor 6
In a Batch Environment 6
In an Interactive Environment 7
Using SCRIPT/VS as a Subroutine 7
Using SCRIPT/VS as a Preprocessor 7
When To Use SCRIPT/VS Control Words 7
Selecting Control Words 8
SCRIPT/VS Functions 8
Formatting Functions 8
Page Layout 8
Head Levels 8
Table of Contents 9
Highlighted Phrases 9
Footnotes 9
Hyphenation and Spelling Verification 9
Printing Part of the Output Document 9
Tabs 9
Boxes 9
Documents Marked up for SCRIPT/370 9
Keeping Text Together 10
General Document Handling Functions 10
Saving Input Lines for Subsequent Processing 10
Identifying Updated Material 10
Imbedding Separate Files 10
Processing Symbols and Macros 10
Processing Input Conditionally 10
Processing Interactively During Formatting 10
Specifying the Destination of Output 11
Converting ATMS-II Documents 11
Debugging by Tracing Processing Actions 11

Chapter 2. Using the SCRIPT Command 13
Naming the Input File 13
CMS Naming Conventions 13
TSO Naming Conventions 14
SCRIPT Command Options 14
Defaults 16
Mutually Exclusive Options 16
Logical and Physical Output Devices 17
Examples 17
BIND: Shift the Page Image to the Right 17
CHARS: Specify Fonts 17
CONTINUE: Continue Processing After a Nonsevere Error Occurs 18
DEST: Name a Remote Output Station 19
DEVICE: Specify a Logical Output Device 19
DUMP: Enable the .ZZ Control Word 19
FILE: Name a Disk File for Output 19
LIB: Specify Symbol and Macro Libraries 21
MESSAGE: Control Message Printing 22
NOPROF: Suppress the Profile 23
NOSPIE: Prevent Entering SPIE Exit Routines 23
NOWAIT: Prevent Prompting for Paper Adjustment 23
NUMBER: Print the File Name and Line Number 23
OPTIONS: Name a File That Contains Options 23
PAGE: Selectively Print Pages 24
PRINT: Produce Printer Output 25
PROFILE: Specify a Profile 25
QUIET: Suppress the Formatter's Identifier Message 26
SEARCH: Specify a Library 26
SPELLCHK: Enable the .SV Control Word 26
STOP: Print Separate Pages at the Terminal 26
SYSVAR: Set System Variable Symbols 26

TERM: Display the Output at the User's Terminal 27
TWOPASS: Prepare the Document With Two Formatting Passes 27
UNFORMAT: Print All Input Lines Without Formatting 28
UPCASE: Print Lowercase Letters as Uppercase 28

Chapter 3. Basic Text Processing 29
GML Markup and Control Words 29
SCRIPT/VS Text Formatting 29
Format Mode 29
Ragged Right 30
Concatenate Mode 31
SCRIPT/VS Implicit Formatting Conventions 31
Using Tabs In SCRIPT/VS 32
Setting Tabs 32
Some Uses for Tabs 33
Tab Fill Characters 33
Breaks 34
Changing the Margins 35
Simple Indention 36
Indenting a Single Line 37
Offsetting Text 38
Using Indention with Tabs 39
Vertical Space 40
Line Spacing 41
Positioning Lines on the Page 42
Underlining and Capitalizing 44
Forcing a New Page 45
Specifying the Odd or Even Page 46
Specifying Page Eject Mode 46
Guidelines for Entering Text and Control Words In SCRIPT/VS 46
Start All Input Lines In Column One 47
Avoid a Text Period In Column One 47
Remember Which Control Words Cause Breaks 47
Group the SCRIPT/VS Control Words 48
Redefining the Control Word Separator 48
Comments in SCRIPT/VS Documents 49
Two Kinds of Comment 50

Chapter 4. Defining a Page Layout 51
Basic Page Dimensions 51
Changing the Page Length 53
Changing the Line Length 53
Top and Bottom Running Titles 54
Multiline Running Titles 55
Allocating Space For Running Titles 57
Running Title Defaults 59
Running Headings and Footings 59
Page Numbers In Headings and Footings 61
Where To Define Headings, Footings, and Running Titles 62
A Heading on Page One 62
Page Numbers 62
Decimal Page Numbers 62
Roman Numeral Page Numbers 63
Alphabetic Page Numbers 63
Prefixes for Page Numbers 63

Chapter 5. Multicolumn Page Layout 65
Defining Multicolumn Layout 65
Page Sections and Section Breaks 66
Column Positions 67
Column Width 68
Starting a New Column 69
Suspending and Resuming Multicolumn Processing 69

Chapter 6. Head Levels and Table of Contents 71
Characteristics of Head Levels 71
Spacing and Page Ejects 72
Defining Head Levels 73
The Table of Contents 73
Adding Lines to the Table of Contents 74
Like This One 74
Printing the Table of Contents 74
TWOPASS Considerations 75

Chapter 7. Additional Formatting Features of SCRIPT/VS 77
Using Special Characters 77
Input Character Translation 78
Character Translation For Terminal Output 81
Defining Special Characters That Affect SCRIPT/VS Processing 81
The Continuation Character 82
Ensuring That Blocks of Text Stay Together 82
Footnotes 84
Starting Text at the Top of a Page or Column 85
Conditional Column and Page Ejects 85
Marking Updated Material 86
Drawing Boxes 87
Stacking one box on another 90
Drawing a box within a box 91
Drawing the middle portion of a box within another (larger) box 92
Drawing boxes in a horizontal row 92
Drawing the top line (only) of a box 92
Drawing the middle portion of a box (without top or bottom lines) 93
Drawing the bottom line (only) of a box 94
Using Fonts With the IBM 3800 Printing Subsystem 94

Chapter 8. The SCRIPT/VS Formatting Environment 97
Parameters That Define the Formatting Environment 97
The Keep Environment 97
The Footnote Environment 98
Saving and Restoring the Current Formatting Environment 98

Chapter 9. Conditional Processing 99
The .IF Control Word 99
Special Techniques for Conditional Processing 100
Conditional Sections 102
Conditional Processing With Symbols 104

Chapter 10. Combining SCRIPT/VS Files 105
Imbedding and Appending Files 105
Naming the File To Be Imbedded or Appended 106
Master Files 107
Writing To an Output File 108
Several .WF Files 109
Delaying the Imbedding of Input Text 110
Terminating the Formatting of a File 110
Merging Documents From Several Sources 111
Creating a Customized Letter For Mass Mailing 111
Interactive SCRIPT/VS Processing 112
Communicating With VM/370 114
Communicating With TSO 115

Chapter 11. Symbols in Your Document 117
How SCRIPT/VS Substitutes Values For Symbol Names 119
Compound Symbols 119
Unresolved Symbols 120
GML Tags 120
Inhibiting Substitution 121
Cancelling a Symbol 122
Attributes of a Symbol's Value 122
Symbol and Macro Libraries 125
SCRIPT/VS System Symbols 126
Symbols for the System Date and Time 126
Elaborating the System Date 127
Symbols for SCRIPT/VS Control Values 128
The &$RET Special Symbol 128
Passing Parameters To Input Files 129
Setting Symbols With the SCRIPT Command 129
Symbols Set When a File Is Imbedded or Appended 129
Symbols Set When a Macro Is Processed 130
Some Things You Can Do With Symbols 131
Setting the Current Page Number 131
Numbering Figures 131
Prefixes and Suffixes For Figure Numbers 133
Extended Symbol Processing 133
Symbols for Arrays of Values 134
Controlling the Array Elements 134
Accessing the Index Counter 135
Setting the Index Counter 135

Chapter 12. Writing SCRIPT/VS Macro Instructions 137
When Should You Use Macros? 137
How To Define a Macro 137
Macro Naming Conventions 138
Local Symbols for Macros 138
Redefining SCRIPT/VS Control Words 140
Avoiding an Endless Loop 141
How Values Are Substituted For Symbols Within a Macro Definition 141
Redefining SCRIPT/VS Formatting Conventions 142
Processing Input Lines That Are Empty 142
Processing Input Lines That Begin With a Blank or a Tab 142
Specifying a Macro Library 143

Chapter 13. GML Support in SCRIPT/VS 145
The Role of a Document Profile 145
Creating Your Own Profiles 145
GML Tag to APF Mapping 146
APFs for Text Items 146
Symbols Within Starter Set APFs 147
Starter Set Macros for Attribute Processing 149
Use of the .LI [Literal] Control Word 149
Debugging Your APFs 150

Chapter 14. Using SCRIPT/VS with Other Programs 153
Using SCRIPT/VS as a Postprocessor 153
Using SCRIPT/VS as a Preprocessor 153
Developing Preprocessor APFs and Profiles 154
Redefining Symbols 155
Handling Directly Entered Control Words 155
Source Document Management 155
Modifying Unpredictable Processing Results 156
Special Graphic Effects 156
Preparing for Processing 156

Chapter 15. Automatic Hyphenation and Spelling Verification 157
How Automatic Hyphenation Works 157
Altering the Hyphenation Parameters 157
Hyphenating Single Words 157
How Spelling Verification Works 158
The SCRIPT/VS Dictionary 158
The Main Dictionary 159
The Addenda Dictionary 159
Building an Addenda Dictionary 160
Stem Processing 160
Prefixes Removed from Words 161
Suffixes Removed from Words 161
Fallibility 161

Chapter 16. Diagnostic Aids 163
Debugging With the SCRIPT Command 163
CONTINUE: Continue Processing After an Error Occurs 163
DUMP: Enable the .ZZ [Diagnostic] Control Word 163
MESSAGE: Control Information in Error Messages 163
NOSPIE: Prevent Entering SPIE Exit Routines 164
NUMBER: Print the File Name and Line Number 164
PAGE: Selectively Print Pages 164
SPELLCHK: Verify Spelling 164
TWOPASS: Provide Two Formatting Passes 164
UNFORMAT: Print All Input Lines Without Formatting 165
Control Words to Assist Debugging 165
Spelling Verification 165
Displaying the Sequence of SCRIPT/VS Processing 165
The Output Line Generated by Input Tracing 165
Capabilities of the .IT Control Word 166
Stepping through an Input Trace 167
Using Terminal Entry to Test a Control Word Sequence 169
Putting Messages In Macros 170
Displaying Control Blocks 170

Chapter 17. EasySCRIPT 171
EasySCRIPT Tags 171
EasySCRIPT Formats 172
Headings 172
Setting the Heading Counter 173
EasySCRIPT Heading Defaults 173

Cross-References to EasySCRIPT Headings 173
Examples of EasySCRIPT Formatting 174
Paragraphs 174
Automatic Item Numbering 174
Unnumbered Lists 175
Bullets 175
Tables of Contents 175

Chapter 18. Compatibility with SCRIPT/370 . . . . . **177**
Changes to the SCRIPT Command 177
Changes to SCRIPT/370 Control Words 178
The SCRIPT/370 Dictionary 181

Chapter 19. ATMS-II Conversion **187**
Converting ATMS-II Documents To SCRIPT/VS Format 187
  Conversion Technique 187
  Non-Format Command Conversion 188
  Formatting Control Conversion 189
ATMS Control - SCRIPT/VS Macro Relationship 194

Chapter 20. Compatibility with TSO/FORMAT **197**
  Creating a TSO/FORMAT Compatible Environment 197
  The SCRIPT Command in TSO 197

Chapter 21. SCRIPT/VS Control Word Descriptions **199**
Control word syntax 199
  Macros 199
  The control word modifier 199
  Type 1 control words 199
  Space units 200
Notational conventions 200
... [Set Label] 202
.AP [Append] 203
.BC [Balance Columns] 204
.BF [Begin Font] 204
.BM [Bottom Margin] 205
.BR [Break] 206
.BT [Bottom Title] 206
.BX [Box] 207
.CB [Column Begin] 209
.CC [Conditional Column Begin] 209
.CD [Column Definition] 210
.CE [Center] 211
.CL [Column Width] 212
.CM [Comment] 212
.CO [Concatenate Mode] 213
.CP [Conditional Page Eject] 214
.CS [Conditional Section] 214
.CW [Control Word Separator] 215
.DC [Define Character] 217
.DD [Define Data File-id] 219
.DH [Define Head Level] 221
.DI [Delay Imbed] 222
.DM [Define Macro] 223
.DS [Double Space Mode] 225
.DU [Dictionary Update] 226
.EB [Even Page Bottom Title] 227
.EC [Execute Control] 227
.EF [End of File] 228
.EM [Execute Macro] 229
.EP [Even Page Eject] 230
.ET [Even Page Top Title] 230
.EZ [EasySCRIPT] 231
.FM [Footing Margin] 232
.FN [Footnote] 233
.FO [Format Mode] 234
.FS [Footing Space] 235
.GO [Goto] 236
.HM [Heading Margin] 237
.HN [Headnote] 237
.HS [Heading Space] 238
.HW [Hyphenate Word] 239
.HY [Hyphenate] 239
.H0 - .H6 [Head Level 0 - 6] 240
.IF [If] 241

.IL [Indent Line] 243
.IM [Imbed] 244
.IN [Indent] 245
.IR [Indent Right] 246
.IT [Input Trace] 246
.JU [Justify Mode] 248
.KP [Keep] 248
.LB [Leading Blank] 250
.LI [Literal] 251
.LL [Line Length] 252
.LS [Line Spacing] 252
.LT [Leading Tab] 253
.LY [Library] 253
.MC [Multicolumn Mode] 254
.MG [Message] 255
.MS [Macro Substitution] 256
.NL [Null Line] 256
.OB [Odd Page Bottom Title] 257
.OC [Output Comment] 257
.OF [Offset] 258
.OP [Odd Page Eject] 259
.OT [Odd Page Top Title] 259
.PA [Page Eject] 260
.PF [Previous Font] 261
.PL [Page Length] 261
.PN [Page Numbering Mode] 262
.PP [Paragraph Start] 264
.PS [Page Number Symbol] 264
.PT [Put Table of Contents] 265
.QQ [Quick Quit] 266
.QU [Quit] 266
.RC [Revision Code] 267
.RD [Read Terminal] 268
.RE [Restore Status] 268
.RF [Running Footing] 269
.RH [Running Heading] 270
.RI [Right Adjust] 271
.RT [Running Title] 272
.RV [Read Variable] 273
.SA [Save Status] 274
.SC [Single Column Mode] 274
.SE [Set Symbol] 275
.SF [Save Font] 278
.SK [Skip] 278
.SL [Set Line Space] 279
.SP [Space] 279
.SS [Single Space Mode] 280
.SU [Substitute Symbol] 280
.SV [Spelling Verification] 281
.SX [Split Text] 282
.SY [System Command] 283
.TB [Tab Setting] 284
.TC [Table of Contents] 285
.TE [Terminal Input] 286
.TI [Translate Input] 287
.TM [Top Margin] 287
.TR [Translate Character] 288
.TT [Top Title] 289
.TY [Type on Terminal] 290
.UC [Underscore and Capitalize] 290
.UD [Underscore Definition] 291
.UN [Undent] 292
.UP [Uppercase] 293
.US [Underscore] 294
.WF [Write To File] 295
.ZZ [Diagnostic] 296

Appendix A. SCRIPT/VS Summary 297

Appendix B. Device and Font Table Maintenance 323
Updating a Logical Device Table (LDT) 323
LDT Field Descriptions 323
Default Values for Logical Output Devices 325
Font Table Maintenance 325
Updating the Font Table (FTB) 325

FTB Field Descriptions 326
Fonts Provided With SCRIPT/VS 327
3800 Printer Fonts Supported By SCRIPT/VS 327

Appendix C. Fonts Supplied with SCRIPT/VS 329

Appendix D. Formatting Considerations for the 3800 Printer 337
Font Management 337
Tab, Backspace, Underscore Resolution 338
Interword Space 338
Revision Code Characters 339
Inline Space Management 339
Box Processing 340
Formatter Escape Character 341

Glossary 343

Index 351

Figure 1. SCRIPT Option Scan Order for non-TSO Systems 14
Figure 2. Summary of SCRIPT Options (Part 1 of 2) 15
Figure 3. Logical Output Device vs. Output Destination 18
Figure 4. SCRIPT/VS Logical Device Characteristics 20
Figure 5. Data Set Name Qualification in TSO 21
Figure 6. How the Current Margins Are Established 35
Figure 7. SCRIPT/VS Terms for Parts of the Page 52
Figure 8. Summary of Default Head Level Characteristics 72
Figure 9. Imbedding and Appending SCRIPT/VS Files 106
Figure 10. Master File Structure 108
Figure 11. Result of the Customized Form Letter 113
Figure 12. APF for List Items 154
Figure 13. SCRIPT/370 Command Option Compatibility (Part 1 of 2) 177
Figure 14. Changes to SCRIPT/370 Control Words (Part 1 of 4) 182
Figure 15. New Control Words 186
Figure 16. Obsolete Control Words 186
Figure 17. Character Codes Regcognized by ATMS-II Conversion 194
Figure 18. ATMS-II Controls to SCRIPT/VS Symbols Conversion 195
Figure 19. Unsupported TSO/FORMAT Control Words 197
Figure 20. Index to SCRIPT/VS Summary 297
Figure 21. SCRIPT/VS Terms for Parts of the Page 298
Figure 22. File-id's of SCRIPT/VS Utility files 299
Figure 23. Summary of SCRIPT Options (Part 1 of 2) 299
Figure 24. SCRIPT/VS Control Word Summary (Part 1 of 10) 301
Figure 25. Control Words That Cause a Break 311
Figure 26. Control Words That Take Effect On the Next Page 311
Figure 27. Control Words That End a Keep, Running Heading or Footing, or Footnote 312
Figure 28. Control Words Within a Running Heading or Footing 312
Figure 29. Control Word Values Based On the Logical Device 313
Figure 30. SCRIPT/VS Logical Device Characteristics 313
Figure 31. Summary of Head Level Characteristics 314
Figure 32. The SCRIPT/VS Formatting Environment 315
Figure 33. SCRIPT/VS System Symbol Names (Part 1 of 2) 316
Figure 34. Attributes of a Symbol's Value 318
Figure 35. Characters that Delimit Words for Spelling Verification 319
Figure 36. Characters Not Underscored By Default 319
Figure 37. TN Translate Table For the 1403 Printer 320
Figure 38. Fonts Provided With SCRIPT/VS 320
Figure 39. Fonts Supplied With the 3800 Printer 321
Figure 40. Example of a Font Width Table 326
Figure 41. Fonts Provided With SCRIPT/VS 329
Figure 42. SCRIPT/VS Fonts: Gothic Text 330
Figure 43. SCRIPT/VS Fonts: Serif Text 331
Figure 44. SCRIPT/VS Fonts: Gothic Highlight 332
Figure 45. SCRIPT/VS Fonts: Serif Highlight 333
Figure 46. SCRIPT/VS Fonts: Gothic Special Purpose 334
Figure 47. SCRIPT/VS Fonts: Serif Special Purpose 335
Figure 48. Justification Alignment Error for 3800 Printer Output 341
Figure 49. How the Current Margins Are Established 343

CHAPTER 1. AN INTRODUCTION TO SCRIPT/VS

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

- An interactive environment under:
  - The Conversational Monitor System (CMS), of the IBM Virtual Machine Facility/370 (VM/370).
  - The Time Sharing Option (TSO) of OS/VS2 MVS.

  Use of SCRIPT/VS in the interactive environments requires the Foreground Environment Feature of the Document Composition Facility.
- A batch processing environment under:
  - OS/VS1
  - OS/VS2 MVS
  - DOS/VS

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

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

SCRIPT/VS can also be used as a "preprocessor" to prepare documents for processing by other programs, such as formatters that support photocomposers.

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

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

For more information about VM/370, see:

VM/370 Terminal User's Guide

VM/370 CMS User's Guide

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

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

For more information about TSO, see:

OS/VS2 TSO Terminal User's Guide

OS/VS2 TSO Command Language Reference

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

- Files created by the TSO, CMS, or VSPC editors
- A word processing system attached to the host system via a telecommunications network

- A user-written program that calls SCRIPT/VS as a subroutine

A text processing program reads non-structured input data containing text and control information, formats the data into pages, and produces formatted output on a system printer or other suitable output device.

Information that may appear in the SCRIPT/VS input file includes:

- "text," which is the content of the document.
- "symbols," which are character strings starting with an ampersand (&) that are resolved to a different character string when the line is processed. The new string may be text, another symbol, or control information. For example, in this document the symbol "&3800" resolves to "3800 Printer."
- "control words," which are two-letter codes that are recognized when the first character in the input line is a period (.). For example, to cause a page eject ".PA" is specified in column one of an input line.
- "macros," which are groups of control words having unique names that are themselves treated as control words. A macro is defined using the Define Macro (.DM) control word. For example, the ".XX" macro could be defined to contain a ".NL" control word followed by a ".PA" control word. Anytime the ".XX" macro is processed, the ".NL" and ".PA" control words are substituted and processed as though they were the next two input lines.
- "GML markup," which uses "tags" to identify the associated text as a particular type, such as paragraph or heading. GML stands for "Generalized Markup Language." The Generalized Markup Language provides a syntax and usage rules for marking up a document, and allows you to develop a vocabulary of tags for describing your own documents. A tag is identified by the GML delimiter, which is by default the colon (:), anywhere in an input line. For example, in the GML starter set provided with SCRIPT/VS, ":p" identifies a paragraph.

To "mark up" a source document is to add information to it that tells the SCRIPT/VS system to process it in some specific way. The added information or "markup," is typically GML tags or control words.

Normally, a SCRIPT/VS input file is a sequential file on direct access storage that can be modified using an editing program.

SCRIPT/VS can process the file and produce formatted output that reflects changes to the text or markup.

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

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

- SCRIPT/VS data files are independently maintained. Any editor that can produce files in the form required by SCRIPT/VS may be used to create or modify these files.
- SCRIPT/VS can combine many input files to produce a single, integrated output document. The imbedded files can be arranged in any sequence. While they are being processed, SCRIPT/VS treats each input file as though it were part of a single continuous input file.
- SCRIPT/VS has high-level macro and symbol capabilities. With SCRIPT/VS you can define your own control words, or GML tags, conditionally process text, perform variable symbol substitutions, and do integer arithmetic.

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

GENERALIZED MARKUP LANGUAGE

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

The following example of GML markup describes an element whose type is FIG (figure), and which has a DEPTH attribute of 2.5 inches:

```
:fig depth='2.5i'
```

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

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

HOW SCRIPT/VS WORKS

CONTROL WORDS AND THEIR PARAMETERS

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

Each input line identified as a control word is scanned from left to right for a control word separator, usually a semicolon (;). If found, the control word to the left of the semicolon is processed; the character string to the right of the semicolon (which might be another control word) is saved. This process is repeated until the input line is completely scanned. For example,

```
.pa;.sp .5i
```

will cause a page eject (.PA) followed by a space for one-half inch (.SP).

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" or "OFF" and is specified as follows:

```
.ce on
```

The .SP (Space) control word accepts numeric parameters and is specified as follows:

```
.sp 2i
```

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

```
.ce 10
```

Each control word description lists the parameters that it accepts. See "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199 for details.

Defaults and Initial Settings

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

Each control word description includes initial setting and default values.

SCRIPT/VS INPUT FILE CHARACTERISTICS

SCRIPT/VS input files have the following default characteristics:

- In a CMS environment,
  - filetype of SCRIPT
  - variable-length records, with a maximum of 132 bytes
  - uppercase and lowercase letters, numbers, and special characters

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

  Normally, any TSO editor will create files of appropriate format.
- For input file characteristics in a batch environment, see Document Library Facility Guide.

LOGICAL AND PHYSICAL OUTPUT DEVICES

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

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

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

VERTICAL AND HORIZONTAL SPACE UNITS

Some control words accept parameters that specify vertical or horizontal dimensions or distances. These dimensions may be expressed in any of several different space units. The space unit type is identified with a single letter. Numbers without space unit identifiers are in character spaces horizontally and line spaces vertically.

The space unit types are:

- em: The type size of the blank character (hexadecimal 40) in the current font. The "em" unit is useful when specifying indention, horizontal displacement, and tab settings. The type size of the blank character may be one of the following:

  1/10 inch (2.540 millimeters)
  1/12 inch (2.117 millimeters)
  1/15 inch (1.693 millimeters)

  The EM unit is specified as

  aaM or aam

  For example, to produce an indention of three characters in any font:

  .in 3m

- Cicero: A Cicero in the Didot Point System is 0.1776 inch (4.512 millimeters) and is a standard printer's measurement in most countries except Great Britain and the United States. There are 12 Didot points in one Cicero.

  The Cicero is specified as:

  aaCbb or aacbb

  aa = number of Ciceros
  bb = number of Didot points

  For example,

  C12 = 12 Didot points

  2C3 = 2 Ciceros and 3 Didot points

- Pica: A Pica is 0.1663 inch (4.224 millimeters) and is a standard printer's measurement in Great Britain and the United States. There are 12 points in one Pica.

  The Pica is specified as:

  aaPbb or aapbb

  aa = number of Picas
  bb = number of points

  For example,

  P12 = 12 points

  2P3 = 2 Picas and 3 points

- Inch:

  The Inch is specified as:

  aaI or aai

  aa = number of inches, and can be fractional. Up to two decimal positions may be specified.

  For example,

  3.5i = 3-1/2 inches

- Millimeter:

  The Millimeter is specified as:

  aaW or aaw

  aa = number of millimeters, and can be fractional. Up to two decimal positions may be specified.

  For example,

  12.7W = 12.7 millimeters

FONTS (IBM 3800 PRINTING SUBSYSTEM ONLY)

A "font" is a set of characters having the same size and type style.

In a "monopitch" font, all characters have the same width.

In a "proportional" font, characters may have different widths. For example, the "I" may be narrower than the "H", and the "M" may be wider than the "N".

The width attribute of a font is called "pitch" and is the number of "characters per inch" in a line of printed text.

The 3800 Printer has three pitch values:

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

When using the 3800 Printer, you may select multiple fonts via the CHARS option of the SCRIPT command. Each font corresponds to a Character Arrangement Table (CAT) that will be loaded into the 3800 Printer when the document is printed using the appropriate Job Control Language (JCL).

The CHARS option is described in "Chapter 2. Using the SCRIPT Command" on page 13.

Using different fonts within the input file is described in "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77.

The fonts provided with SCRIPT/VS are illustrated in "Appendix C. Fonts Supplied with SCRIPT/VS" on page 329.

CALLING THE SCRIPT/VS PROCESSOR

IN A BATCH ENVIRONMENT

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

IN AN INTERACTIVE ENVIRONMENT

You call the SCRIPT/VS processor by issuing the SCRIPT command specifying the input file name. The default file type in CMS is SCRIPT.

- In CMS: SCRIPT filename (options
- In TSO: SCRIPT dsname options

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

USING SCRIPT/VS AS A SUBROUTINE

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

USING SCRIPT/VS AS A PREPROCESSOR

SCRIPT/VS can be used to prepare an input file for use as input to another text processing program. For details on translating GML markup to non-SCRIPT/VS formatting controls, see Document Composition Facility: Generalized Markup Language (GML) User's Guide.

WHEN TO USE SCRIPT/VS CONTROL WORDS

When you create an input file, or when you create Application Processing Functions (APFs) for GML processing, you should consider:

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

SELECTING CONTROL WORDS

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

SCRIPT/VS FUNCTIONS

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

FORMATTING FUNCTIONS

Page Layout

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

Page layout includes:

- Line Formatting. You can control line concatenation, line justification, line centering, or line alignment to left or right. For details, see "Chapter 3. Basic Text Processing" on page 29 .
- Line Spacing. You can control the amount of space left between output lines, including the reservation of space for drop-in art. For details, see "Chapter 3. Basic Text Processing" on page 29 .
- Paragraphing. You can control the style of paragraphing, spacing between paragraphs, and indention. For details, see "Chapter 3. Basic Text Processing" on page 29.
- Fonts. You can control which font is used for different portions of text, both in the body and in running headings and footings. For details, see "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77.
- Columns. You can define the number of columns as well as the size of each one and its placement on a page. For details, see "Chapter 5. Multicolumn Page Layout" on page 65.
- Margins. You can control the size of the top and bottom margins as well as the left and right margins. Title lines can be defined that will be put into the top or bottom margins. For details, see "Chapter 4. Defining a Page Layout" on page 51.
- Indention. You can control indention in a number of ways. For example, you can have hanging indents, left or right margin indention, and indent one line only. For details, see "Chapter 3. Basic Text Processing" on page 29.
- Headings and Footings. You can have running headings and footings with page numbers and separate treatment for odd and even pages. For details, see "Chapter 4. Defining a Page Layout" on page 51.

Head Levels

You can specify up to seven head levels for distinctive formatting of headings for different levels of topics. Distinctive formatting includes before and after spacing, font selection, capitalization, underscoring, and right or left alignment. For details, see "Chapter 6. Head Levels and Table of Contents" on page 71.

Table of Contents

You can control whether or not a table of contents is automatically generated and where it is placed. SCRIPT/VS collects entries for a table of contents from the text of head levels and supplies the page number. You can also specify phrases other than the text of head levels to appear in the table of contents. The table of contents of this manual was automatically generated by SCRIPT/VS. For details, see "Chapter 6. Head Levels and Table of Contents" on page 71.

Highlighted Phrases

You can highlight phrases for emphasis. For devices that don't have multiple fonts, highlighting is done with underscore, uppercase, or uppercase underscored. For devices that support multiple fonts, you can change font for emphasis. For details, see "Chapter 3. Basic Text Processing" on page 29.

Footnotes

SCRIPT/VS saves text indicated as a footnote and places it at the bottom of the page.[1] Subsequent footnotes are placed below it.[2] For details, see "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77.

Hyphenation and Spelling Verification

You can control whether or not words are to be hyphenated at the end of output lines, and whether words are to be checked for correct spelling. SCRIPT/VS provides a dictionary of many common English root words. An algorithm for prefix and suffix variations significantly extends the basic root word set. SCRIPT/VS determines hyphenation points and spelling validity based on the prefix and suffix algorithms and the root word set. You can add words to a supplementary dictionary as required for a particular document. For details, see "Chapter 15. Automatic Hyphenation and Spelling Verification" on page 157.

Printing Part of the Output Document

You can control whether every page of formatted text is put in the output document or only the range or ranges of pages specified. For details, see the PAGE option in "Chapter 2. Using the SCRIPT Command" on page 13.

Tabs

You can specify the values of tabs. When formatting output lines, SCRIPT/VS tabs to the right to the prescribed tab stop. For details, see "Chapter 3. Basic Text Processing" on page 29.

Boxes

You can construct boxes around formatted text. You can also draw boxes within boxes, vertical lines to separate columns of text, and horizontal lines to separate rows. For details, see "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77.

---

[1] Like this.
[2] Up to 10 lines per footnote.

Documents Marked up for SCRIPT/370

If you have documents prepared for SCRIPT/370 Version 3, you can use SCRIPT/VS to format them, with very few changes, if any, required. For details, see "Chapter 18. Compatibility with SCRIPT/370" on page 177.

Keeping Text Together

SCRIPT/VS processing includes functions which keep text together to improve the appearance of output. For example, SCRIPT/VS keeps the text of a head level and the following three lines of output text together, so that they appear in the same column. For details, see "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77.

GENERAL DOCUMENT HANDLING FUNCTIONS

Saving Input Lines for Subsequent Processing

You can control whether certain input lines will be written to a data set or file. For details, see "Chapter 10. Combining SCRIPT/VS Files" on page 105.

Identifying Updated Material

You can control the placement of up to nine distinct revision codes in the left margin to indicate revised lines. For details, see "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77 .

Imbedding Separate Files

You can control how separate source files are brought together for processing as a single document. Any number of source files can be imbedded in the primary source file. A source file that has been imbedded can itself imbed another source file. For details, see "Chapter 10. Combining SCRIPT/VS Files" on page 105.

Processing Symbols and Macros

You can define symbols and macros for substitution during processing. Symbols have many uses: for example, in tests for conditional processing, for cross-references to pages or figure numbers, for entering characters unavailable on the entry keyboard, and as abbreviations for repetitive phrases. You can define what a particular macro will do. For example, you might redefine a particular head level to alter the SCRIPT/VS formatting style. Symbol and macro instruction facilities are used to support the Generalized Markup Language. For details, see "Chapter 11. Symbols in Your Document" on page 117 and "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137.

Processing Input Conditionally

You can cause SCRIPT/VS to alter input processing. For example, by setting symbol values you can control whether a block of input text is included in the output document. SCRIPT/VS uses condition testing as part of its normal processing. It checks the amount of space left in a column before processing certain blocks of text. Conditional processing can be controlled by defining macro instructions to supplement SCRIPT/VS control words. For details, see "Chapter 11. Symbols in Your Document" on page 117.

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. For details, see "Chapter 10. Combining SCRIPT/VS Files" on page 105.

Specifying the Destination of Output

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

Converting ATMS-II Documents

With the IBM Document Library Facility program product installed with SCRIPT/VS, you can convert most ATMS-II markup to equivalent SCRIPT/VS markup. For details, see Document Library Facility Guide and "Chapter 19. ATMS-II Conversion" on page 187.

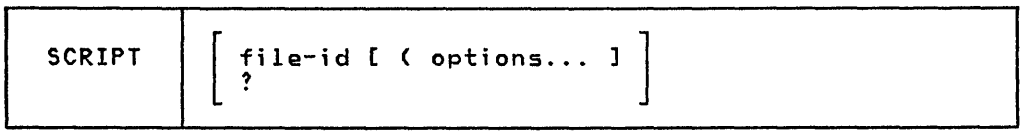
Debugging by Tracing Processing Actions

You can trace all control words and each step of symbol and macro substitution in input lines. In cases where unexpected results are observed, trace information can be an invaluable aid in pinpointing the problem area. For details, see "Chapter 16. Diagnostic Aids" on page 163.

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

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

In CMS,



In TSO,



where:

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

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

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

NAMING THE INPUT FILE

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

CMS NAMING CONVENTIONS

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

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

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

TSO NAMING CONVENTIONS

In TSO, you can use a fully or partially qualified data set name to refer to the primary input file, identified by file-id in the SCRIPT command, and other input files associated with the document, such as the profile and imbedded files.

If the file-id given is not fully qualified (placed within quotes), the userid is prefixed to the file-id as the leftmost qualifier, and "TEXT" is added (unless it already appears) as the rightmost 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)

Option	Minimum Abbreviation
BIND	B
CHARS	C
CONTINUE	CO
DEVICE	D
DUMP	DU
FILE	F
LIB	L
MESSAGE	M
NOPROF	N
NOWAIT	NOW
NUMBER	NU
OPTIONS	O
PAGE	P
PRINT	PR
PROFILE	PRO
QUIET	Q
SEARCH	S
SPELLCHK	SP
STOP	ST
SYSVAR	SY
TERM	T
TWOPASS	TW
UNFORMAT	U
UPCASE	UP
NOSPIE	NOS

Figure 1. SCRIPT Option Scan Order for non-TSO Systems

SCRIPT COMMAND OPTIONS

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

The options you can specify with the SCRIPT command are shown in Figure 2 on page 15.

Option	Parameters	Description
BIND	(bind) (obind ebind)	Shift the page image to the right.
CHARS	(font1 ... font4)	Specify up to four fonts.
CONTINUE		Continue processing after a nonsevere error occurs.
DEST	(station id)	Specify a remote output station. (Valid only for TSO.)
DEVICE	(devtype)	Specify a logical output device.
DUMP		Enables the .ZZ [Diagnostic] control word.
FILE	[(fileid)]	Specify a disk file for output.
LIB	(libname ...)	Specify symbol and macro libraries. (Only one for TSO; up to eight for CMS.)
MESSAGE	([DELAY] [ID] [TRACE])	Control message printing.
NOPROF		Suppress the profile.
NOSPIE		Prevent entering SPIE exit routines. (Valid only for CMS and TSO.)
NOWAIT		Prevent prompting for paper adjustment. (Valid only for typewriter terminal output.)
NUMBER		Print file name and line number.
OPTIONS	[(fileid)]	Specify a file that contains SCRIPT options (Valid only for CMS.)
PAGE	[FROM] p [TO] q [FROM] p FOR n [FROM] p ONLY PROMPT	Selectively print pages.
PRINT	[(copies,class, fcb,ucs)]	Produce printer output. (Sub-options valid only for TSO.)
PROFILE	[(fileid)]	Specify a profile. (A file to be imbedded before the primary input file is processed.)
QUIET		Suppress the formatter's identifier message.
SEARCH	(libname)	Specify a library. (Not valid in a CMS environment.)
SPELLCHK		Enable the .SV [Spelling Verification] control word.
STOP		Print separate pages at the terminal. (Valid only for typewriter-terminal output.)
SYSVAR	(n value ...)	Set symbol values for &SYSVARn.
TERM		Display the output at a user's terminal. (Valid only in CMS and TSO.)
TWOPASS		Prepare with two formatting passes, and produce output on the second pass.

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

Option	Parameters	Description
UNFORMAT		Print all input lines without formatting.
UPCASE		Fold lowercase letters to uppercase before printing.

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

The name of each option can be shortened to its minimum unambiguous length. In TSO, ambiguous truncations are not accepted: you will be prompted to reenter the option. In other systems, ambiguous truncations are accepted and resolved by matching the first option encountered in the scan order shown in Figure 1 on page 14. Note that this scan order is not alphabetical.

DEFAULTS

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

For CMS,

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

For TSO,

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

For Batch,

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

All other options must be explicitly specified when desired.

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

MUTUALLY EXCLUSIVE OPTIONS

Some of the SCRIPT command options are mutually exclusive from a logical standpoint. However, when two are specified, no error results. Instead, an option can cancel the effect of a previously specified option. In TSO, 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. Within the following groups of options, the last one processed by SCRIPT/VS takes effect:

- PROFILE and NOPROF.
- TERM and PRINT. TERM or PRINT serves to identify the type of formatting, for a typewriter or a printer, unless DEVICE is also specified. If FILE is not also specified, TERM or PRINT also identifies the output's destination. (When FILE is specified, the document's destination is a direct-access file.) When both DEVICE and FILE are specified, neither TERM nor PRINT has any effect.

LOGICAL AND PHYSICAL OUTPUT DEVICES

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

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

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

Figure 3 on page 18 shows the logical output device and output destination for a document when various combinations of options are specified.

Examples

- In CMS, format and print the document named TEST for an IBM 1403 printer. Print the last part of the document, starting with page 10, and allow for a binding margin on the left side of each page of 4 character-spaces:

  ```
  SCRIPT TEST ( PRINT PAGE (10) BIND (4)
  ```

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

  ```
  SCRIPT RESUME NOWAIT QUIET
  ```

BIND: SHIFT THE PAGE IMAGE TO THE RIGHT

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

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

You can specify a binding for odd-numbered pages (obind) and a different binding for even-numbered pages (ebind). If ebind is not specified, the value of bind applies to both odd- and even-numbered pages. The actual (or potential) page number of the output page is controlled by the .PA [Page Eject] and .PN [Page Numbering Mode] control words, which are used to specify even and odd page numbers. Consequently, you can have two or more even-numbered (or odd-numbered) pages in a row.

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

If the BIND option is not specified, it defaults to two character spaces. This allows room for potential revision codes for the first column. (Revision codes for subsequent columns are placed in the gutter between columns.) If sufficent room is not provided for revision codes, they are discarded.

CHARS: SPECIFY FONTS

The CHARS option identifies the fonts to be used when formatting for the 3800 Printer. You may specify up to four uppercase only fonts, or two upper- and lowercase fonts. Fonts provided with SCRIPT/VS are illustrated in "Appendix C. Fonts Supplied with SCRIPT/VS" on page 329.

Options Specified	Logical Output Device	Output Destination
none [CMS, TSO]	TERM [Terminal]	Terminal
none [Background]	1403W6 [Printer]	Printer
TERM	TERM [Terminal]	Terminal
PRINT	1403W6 [Printer]	Printer
DEVICE(devtype)	devtype	devtype
FILE [CMS, TSO]	TERM [Terminal]	Disk file
FILE [Background]	1403W6 [Printer]	Disk file
FILE TERM	TERM [Terminal]	Disk file
FILE PRINT	1403W6 [Printer]	Disk file
FILE DEVICE(devtype)	devtype	Disk file
TERM DEVICE(devtype)	devtype	Terminal
PRINT DEVICE(devtype)	devtype	Printer

Figure 3. Logical Output Device vs. Output Destination: It is the user's responsibility to ensure that the characteristics of the physical device to which the output is directed match the characteristics of the specified or implied logical device. Your installation's conventions for output class, FCB, and forms must be included in these considerations.

The CHARS option is specified as:

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

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

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

After formatting, the document must be printed by a job or subsystem on an operating system that supports the 3800 Printer, in order to use the appropriate fonts. The CHARS JCL parameter must specify the corresponding character arrangement tables in the same sequence as the fonts specified in the CHARS option of the SCRIPT command.

CONTINUE: CONTINUE PROCESSING AFTER A NONSEVERE ERROR OCCURS

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

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

DEST: NAME A REMOTE OUTPUT STATION

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

The DEST option is specified as:

```
DEST (station-id)
```

station-id is a one- to eight-character name.

DEVICE: SPECIFY A LOGICAL OUTPUT DEVICE

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

The DEVICE option is specified as:

```
DEVICE (devtype)
```

devtype is the name of a logical output device that takes into account the physical characteristics of the device as well as the characteristics that can be changed by the operator or by program control: font, lines per inch, form size, and page image size. SCRIPT/VS logical device support allows a single physical device type to be defined as many different logical device types, each having different characteristics. The logical devices defined in SCRIPT/VS are summarized in Figure 4 on page 20.

You can add a new logical device to the SCRIPT/VS logical device table. For details about this procedure, see "Appendix B. Device and Font Table Maintenance" on page 323.

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

The formatted output for all 3800 logical devices contains table reference characters (TRCs). Consequently, the parameter DCB=OPTCD=J must be included in the output JCL.

DUMP: ENABLE THE .ZZ CONTROL WORD

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

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

For details about the output produced when the .ZZ [Diagnostic] control word is processed, see ".ZZ [Diagnostic]" on page 296.

FILE: NAME A DISK FILE FOR OUTPUT

The FILE option directs the formatted output document to a direct-access file.

The FILE option is specified as:

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

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

[1] Line lengths are given as "default/maximum" in 10-pitch characters. For the 3800 Printer, 12-pitch and 15-pitch fonts have values 20% and 50% greater, respectively.

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

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

Figure 4. SCRIPT/VS Logical Device Characteristics

Output is formatted for either a printer or a terminal. When you specify the PRINT option or the DEVICE option with a printer device type, the output document is in the specified printer format. Otherwise, the output document is sent to the direct-access file in terminal format.

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

- In CMS, the file-id is of the form:

  filename [filetype [filemode]]

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

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

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

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

2. If a partially qualified name is provided, it is fully qualified by prefixing it with "userid." and suffixing it with ".LIST" (unless ".LIST" is already the rightmost qualifier) or replacing a rightmost qualifier of ".TEXT" with ".LIST".

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

Figure 5 illustrates the manner in which file-ids are qualified in the TSO environment.

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

```
Organization: PS or PO
Record format: VB or VBM
Record length: 210
```

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

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

If a document is formatted for a printer and is sent to a direct-access file, the output document has printer controls imbedded in it appropriate for the specified or implied logical output device type. You must ensure you print the document on the same device for which it was formatted.

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

File Specification	Input DSNAME	Generated Output DSNAME	Rule Number
FILE('DOC.OUT')	N/A	DOC.OUT	1
FILE(DOC.OUT)	N/A	userid.DOC.OUT.LIST	2
FILE(DOC.LIST)	N/A	userid.DOC.LIST	2
FILE(DOC.TEXT)	N/A	userid.DOC.LIST	2
FILE((CHAP2))	N/A	userid.LIST(CHAP2)	2
FILE	'DOC.TEXT'	DOC.LIST	3
FILE	'DOC.OTHER'	*** error ***	

Figure 5. Data Set Name Qualification in TSO

LIB: SPECIFY SYMBOL AND MACRO LIBRARIES

The LIB option is only valid in the CMS and TSO 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, you can use the SEARCH option for a similar function.

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 quotes), the userid is prefixed to the libname as the leftmost qualifier, and "MACLIB" is added (unless it already appears) as the rightmost qualifier.

libname names a symbol and macro library. SCRIPT/VS uses the library if your input file includes the .LY [Library] control word with the ON, SYM, or MAC parameters specified, or when the .DM [Define Macro] and .SE [Set Symbol] control words include the LIB parameter.

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

```
.se symbolname LIB

.dm macroname LIB
```

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

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

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

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

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

MESSAGE: CONTROL MESSAGE PRINTING

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

The MESSAGE option is specified as:

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

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

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.

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.

NOPROF: SUPPRESS THE PROFILE

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

NOSPIE: PREVENT ENTERING SPIE EXIT ROUTINES

The NOSPIE option requests that SCRIPT/VS not establish a program interrupt exit. The NOSPIE option is intended for use by the system programmer who is maintaining SCRIPT/VS. It is not allowed when calling SCRIPT/VS from a batch processing environment.

NOWAIT: PREVENT PROMPTING FOR PAPER ADJUSTMENT

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

NUMBER: PRINT THE FILE NAME AND LINE NUMBER

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

OPTIONS: NAME A FILE THAT CONTAINS OPTIONS

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

The OPTIONS option is specified as

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

If the file-id is not specified, the default file-id is SCRIPT OPTIONS, and if only a filename is given, the default filetype is OPTIONS.

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

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

PAGE: SELECTIVELY PRINT PAGES

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

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

```
[FROM] frompage [TO] topage

[FROM] frompage FOR n

[FROM] page ONLY

PROMPT
```

If no parameter is given with the PAGE option, PAGE (PROMPT) is assumed.

The following are examples of valid explicit page range specifications:

```
FROM 10 TO 15

7 FOR 2

viii ONLY 93 TO *

FROM 95.1 FOR 3 99 ONLY
```

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

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

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

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

```
PAGE (6 1)
```

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

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

PRINT: PRODUCE PRINTER OUTPUT

The PRINT option causes SCRIPT/VS to send the output document to a printer. If the DEVICE option is not specified, SCRIPT/VS assumes DEVICE (1403W6).

If the FILE option is also specified, SCRIPT/VS formats your document for a printer (implicitly, DEVICE (1403W6)) unless the DEVICE option is also specified. SCRIPT/VS then sends the output to a disk file.

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

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

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

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

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

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

PROFILE: SPECIFY A PROFILE

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

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

For details about creating a profile, see "Chapter 13. GML Support in SCRIPT/VS" on page 145.

The PROFILE option is specified as:

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

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

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

In CMS, the default is:

```
PROFILE SCRIPT
```

In TSO, the default is:

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

QUIET: SUPPRESS THE FORMATTER'S IDENTIFIER MESSAGE

The QUIET option causes SCRIPT/VS to not display the version identification message that is otherwise typed or displayed as a response to the SCRIPT command.

SEARCH: SPECIFY A LIBRARY

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

The SEARCH option is specified as:

```
SEARCH (libname)
```

The SEARCH option is invalid in a CMS environment.

For TSO, if the SEARCH option is not specified:

- And the user allocates a DDname of TEXTLIB, SCRIPT/VS uses whatever data sets are allocated to this ddname to attempt to find files to be imbedded when the .DD [Define Data File-id] control word has not been used to specify the actual data set name. Any number of data sets may be concatenated in this manner.
- And no ddname of TEXTLIB exists, SCRIPT/VS checks the primary input data set. If it is a partitioned data set, it is used as the SEARCH data set; otherwise, SCRIPT/VS assumes that a data set named 'userid.TEXT' contains imbedded files.

SPELLCHK: ENABLE THE .SV CONTROL WORD

The SPELLCHK option causes SCRIPT/VS to verify spelling if the .SV [Spelling Verification] control word has been specified with the ON option. SCRIPT/VS verifies each word in the input file using the spelling and hyphenation dictionary. Spelling errors are listed with other errors found during formatting.

If the UNFORMAT option is specified, the .SV [Spelling Verification] word has no effect, even when the SPELLCHK option is given.

STOP: PRINT SEPARATE PAGES AT THE TERMINAL

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

The STOP option is valid only for typewriter-terminal output.

When SCRIPT/VS stops, no message is issued. SCRIPT/VS unlocks your keyboard, and is ready to type the new output page at the proper position for the first output line. The first output line might be either a top title, a running heading, a blank line, or a line of text.

When SCRIPT/VS stops, position the paper one line above where you want the first line to be typed and press RETURN to resume output typing.

SYSVAR: SET SYSTEM VARIABLE SYMBOLS

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

The SYSVAR option is specified as:

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

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

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

For example, your input file might include the lines

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

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

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

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

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

TERM: DISPLAY THE OUTPUT AT THE USER'S TERMINAL

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

If the FILE option is also specified, SCRIPT/VS formats your document for the terminal and sends the output to a disk file rather than the terminal.

TERM is valid only in interactive environments and is the default when neither PRINT or FILE are specified.

TWOPASS: PREPARE THE DOCUMENT WITH TWO FORMATTING PASSES

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

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

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

You can also use the TWOPASS option to detect errors in an input file. If you process a document with TWOPASS and without CONTINUE, the second pass will not begin unless the first pass is completed with no errors.

If TWOPASS is used while processing a file that uses .TE [Terminal Input], text entered as a result of .TE on the first pass will be excluded from the formatted output. Text entered during the second pass, however, will be formatted. You can use the TWOPASS symbol, &$TWO, and .IF [If] to skip the .TE on the first pass.

Note: The SCRIPT/VS symbol and conditional processing functions might cause the input file to look entirely different on the second pass than it did on the first pass. As a result, page numbers might not be accurate in the table of contents or in other cross-references. They reflect the page numbers set during the first pass.

UNFORMAT: PRINT ALL INPUT LINES WITHOUT FORMATTING

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

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

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

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

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

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

UPCASE: PRINT LOWERCASE LETTERS AS UPPERCASE

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

GML MARKUP AND CONTROL WORDS

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

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

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

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

The Document Composition Facility: Generalized Markup Language (GML) User's Guide describes marking up with GML tags. This manual discusses SCRIPT/VS control words.

SCRIPT/VS TEXT FORMATTING

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

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

FORMAT MODE

Most writing that you do requires some kind of formatting. With format mode on, lines that are entered in a SCRIPT/VS file as:[3]

```
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.
```

---

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

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

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

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

```
Boston
Chicago
New York
Providence
```

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

```
Boston Chicago New York Providence
```

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

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

To restore normal formatting, use the control word:

```
.fo on
```

Since ON is the default, you can also use:

```
.fo
```

If you use the .FO OFF control word when you create tables or charts, remember to turn formatting back on when you resume entering text. For more about the .BR [Break] control word, see "Breaks" in this chapter.

RAGGED RIGHT

The .FO [Format Mode] OFF control word suspends both concatenation and justification. When you want to produce SCRIPT/VS output that resembles normal typewriter output (that is, "ragged right" output), you do not want to suspend concatenation. You still 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
  or
.fo
```

CONCATENATE MODE

You can suspend concatenation by using the .CO [Concatenate Mode] control word with the OFF parameter:

```
.co off
```

Each subsequent line of output contains the text of its corresponding input line. If the input line is shorter than the output line width, it is padded with blank space when justification is in effect. If the input line is longer than the output line width, the placement of excess words depends on the .FO [Format Mode] control word parameters:

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

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

With concatenation off, each input line results in a new output line. It is not joined to the previous input line.

To restore concatenation, use the control word

```
.co on
  or
.co
```

Since ON is the default for the .CO [Concatenate Mode] control word, you do not need to specify ON.

The .FO ON (Format Mode On) control word restores both concatenation and justification even if they have been turned off separately with the .CO [Concatenate Mode] OFF and the .FO [Format Mode] LEFT control words.

SCRIPT/VS IMPLICIT FORMATTING CONVENTIONS

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

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

If you follow the publishing convention that requires sentences to be separated by two blanks, you can take advantage of another SCRIPT/VS function. When a SCRIPT/VS input line ends in a "full stop" and the output document is being printed on an IBM 1403 Printer or at a typewriter terminal, a second blank is automatically added when the line is concatenated to the next input line.[4] A "full stop" is a period (.), a question mark (?), or an exclamation point (!). A line is also considered to end in a full stop if it ends with a double quote (") or a right parenthesis ()), and the next-to-last character is a "full stop" character. You can change the characters that are treated as "full stop" characters with the .DC [Define Character] STOP control word (see "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77 for details).

---

[4] See "Appendix D. Formatting Considerations for the 3800 Printer" on page 337 for special considerations relating to the insertion of blanks after "full stop" characters when the document is to be printed on the 3800 Printer.

A null input line in SCRIPT/VS causes a break. Input lines that start with a leading blank or leading tab also cause breaks. SCRIPT/VS generates a control word and executes it when it detects one of these situations. For null lines, SCRIPT/VS generates and executes the .NL [Null Line] control word. For leading blanks, the .LB [Leading Blank] control word, and for leading tabs, the .LT [Leading Tab] control word. All of these control words do exactly 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 .NL, .LB, or .LT macro to provide whatever processing you require. Note that input lines processed in literal mode, under the .LI [Literal] control word, do not invoke the .NL, .LB, or .LT functions.

USING TABS IN SCRIPT/VS

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

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

  ```
  .ti ¬ 05
  ```

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

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

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

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

SETTING TABS

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

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

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

If you enter a word or number (for example, a list item), then press the tab key, SCRIPT/VS fills the remaining character positions (through the next tab setting position) with blanks, then continues formatting text. For example, the lines[5]:

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

result in:

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

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

Neither the physical tab key settings nor the appearance of input lines on the terminal has any affect on the SCRIPT/VS tab positions.

Once a .TB control word has been processed, the tab settings remain in effect until explicitly reset by another .TB control word. You can restore the SCRIPT/VS default values using the .TB control word without parameters:

```
.tb
```

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

SOME USES FOR TABS

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

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

are formatted as:

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

TAB FILL CHARACTERS

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

---

[5] Space units here are specified in "ems." For details about other ways to specify an amount of space, see "Specifying Vertical and Horizontal Space Units" in "Chapter 1. An Introduction to SCRIPT/VS" on page 1.

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

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

is formatted as:

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

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

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

results in:

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

BREAKS

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

If you begin a line with a blank or a tab, the formatting process is interrupted, the text that has accumulated for the current output line is printed, 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 letter J
and S. That's why the quick
brown fox usually jumps.
   But the poodle was fright-
ened and ran away.
```

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

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

results in:

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

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

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

Some SCRIPT/VS control words cause a break in addition to their explicit function. For a complete list of the control words that cause a break, see Figure 25 on page 311.

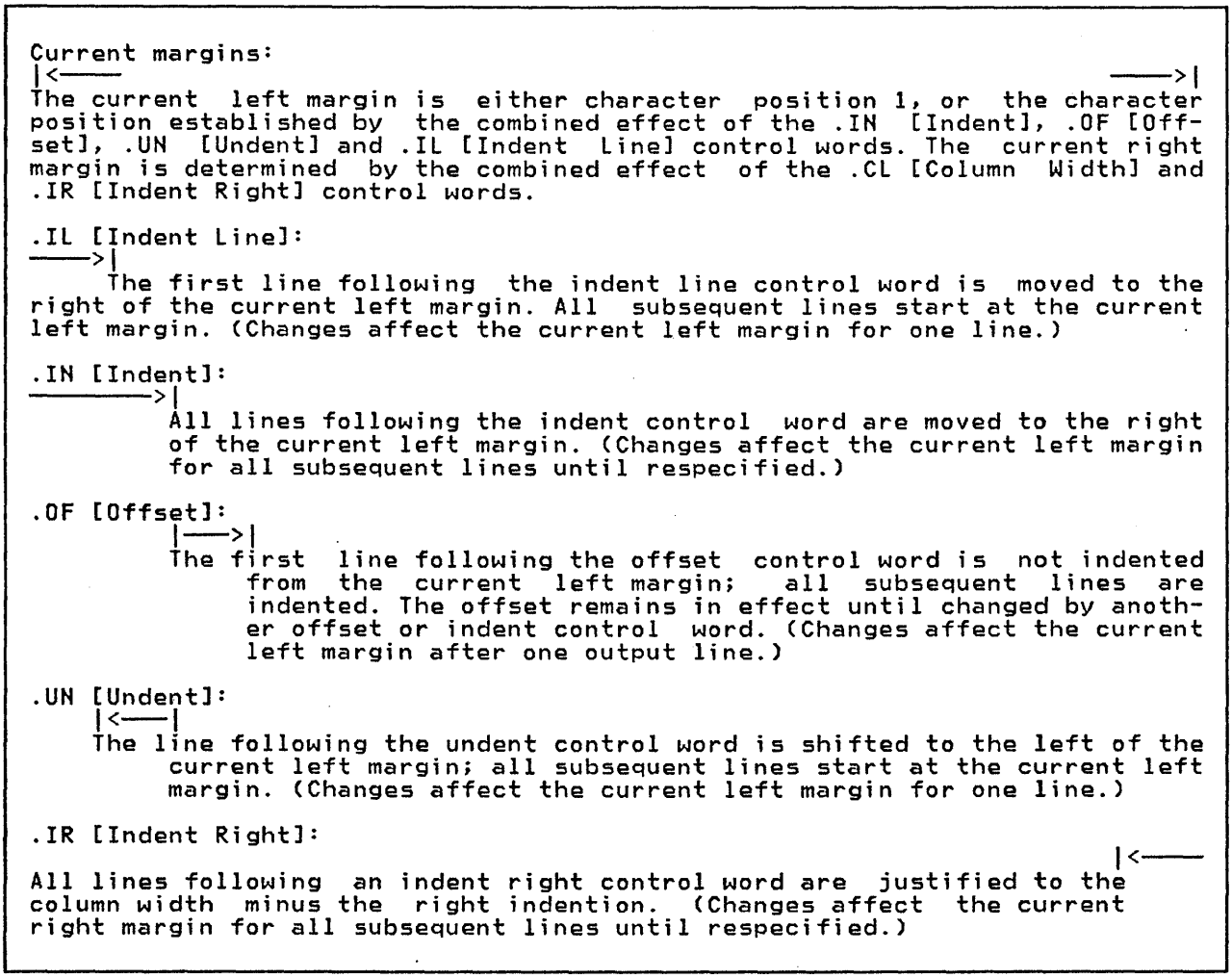


Figure 6. How the Current Margins Are Established

CHANGING THE MARGINS

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

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

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

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

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

- .IN [Indent] - change the left margin for all subsequent output lines.
- .IR [Indent Right] - change the right margin for all subsequent output lines.
- .IL [Indent Line] - apply the specified indent to only the next output line.
- .UN [Undent] - reduce the indent of only the next output line.
- .OF [Offset] - apply this indent to all lines after the next output line.

All margin-modifying control words cause the current output line to be placed on the page and a new output line started. This is called a "break."

Figure 6 on page 35 illustrates the effects of the various margin-modifying control words.

SIMPLE INDENTION

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

```
.in 6m
```

```
——6m->
```

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

```
.in +5m
```

```
———11m———>
```

the value 5M is added to the current left margin. In this example, 6M + 5M is 11M, so the current left 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
```

```
——8m——>
```

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.

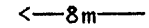
You can return the left margin to the column origin at any time by specifying

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

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

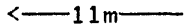
For example, by specifying

```
.ir 8m
```



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 at any time by specifying

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

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

INDENTING A SINGLE LINE

If you would like to indent a single line without modifying the current left margin, you can use the .IL [Indent Line] or .UN [Undent] control word. These control words may be used interchangeably. The indent value specified is summed with the current left margin value to produce a net indent which is applied to the next output line only. If specified, the sign indicates the direction of the indent. For example, to indent the next output line 6M, you would specify

```
.il 6m
  or
.il +6m
  or
.un -6m
```


As you can see, this line is indented 6M and all subsequent lines originate at the current left margin. This technique is often used to indent the first line of a paragraph.

If the body of the text is indented, the opposite effect can be achieved by "undenting" a single line. For example, specify

```
.in 8m
```

to indent the body text.



Undenting the first line of text in a block makes it stand out, providing a natural break for the eye between blocks of text such as list items. For example, by specifying

```
.il -6m
  or
.un 6m
  or
.un +6m
```

<-6m—
Look, this output line originates 6M to the left of the current left margin and catches your eye. All subsequent lines originate at the current left margin. This block of text is easily distinguished from those around it.

In the following examples, the SCRIPT/VS system symbol "&$RB" is used to generate a "required blank" in the input line. Required blanks are not recognized as wordspaces for the purpose of justification. Consequently, they may be used to create blank space of a fixed length in a line that may be justified.

The required blank is by default hexadecimal 41, but may be changed with the .DC [Define Character] RB control word.

A typical application of the .UN [Undent] control word is the formatting of an ordered list. Each list entry starts with an undented output line. For example, to create an ordered list you would specify

```
.in +3m
.un 3m
1)&$RB.The first line of this ordered list
entry is undented 3M.
Subsequent output lines originate
at the current left margin.
.sk 1
.in +3m
.un 3m
A)&$RB.The current left margin has been
moved to the right
by 3M to create this sublist entry.
The first output line is undented 3M.
.in -3m
.sk 1
.un 3m
2)&$RB.The first line of this ordered list
entry is undented 3M.
Subsequent output lines originate
at the current left margin.
```

With justification on, the result will be

1) The first line of this ordered list entry is undented 3M. Subsequent output lines originate at the current left margin.

   A) The current left margin has been moved to the right by 3M to create this sublist entry. The first output line is undented 3M.

2) The first line of this ordered list entry is undented 3M. Subsequent output lines originate at the current left margin.

In the above example, the indent value was set equal to the undent value so that only the number would be undented while the text portion of each output line would originate at the same point.

OFFSETTING TEXT

As an alternative to using the .IN [Indent] and .UN [Undent] control words you can use the .OF [Offset] control word to create an ordered list. For example, you would specify

```
.of 3m
1)&$RB.The first line of this ordered list
entry originates at the current left margin.
Subsequent output lines are offset
3M to the right.
.sk 1
.in +3m
.of 3m
A)&$RB.The current left margin has been
moved to the right
by 3M to create this sublist entry.
Specifying .IN cancels the previous
offset.
The first output line originates at the new
left margin. Subsequent output lines
are offset 3M to the right.
.in -3m
.sk 1
.of 3m
2)&$RB.The first line of this ordered list
entry originates at the current left margin.
Subsequent output lines are offset
3M to the right.
```

With justification on, the result will be

1) The first line of this ordered list entry originates at the current left margin. Subsequent output lines are offset 3M to the right.

   A) The current left margin has been moved to the right by 3M to create this sublist entry. Specifying .IN cancels the previous offset. The first output line originates at the new left margin. Subsequent output lines are offset 3M to the right.

2) The first line of this ordered list entry originates at the current left margin. Subsequent output lines are offset 3M to the right.

You can reset the offset to zero at any time by specifying

```
.of 0
  or
.of
```

USING INDENTION WITH TABS

In the simple case of an ordered list, all the undented lines were the same. That is, they all began with the number, and right parenthesis, followed by a required blank. When the undented lines do not all begin with the same convention, a different technique should be used.

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

```
.in 12m
.tb 12m
.ti ¬ 05
.un 12m
.uc term¬definition
.sk 1
.un 12m
BEE¬any of a number of related four-winged, hairy
insects which feed on the nectar of flowers.
.sk 1
.un 12m
BEEKEEPER¬person who keeps bees for producing
honey; apiarist.
.sk 1
.un 12m
BEESWAX¬a tallow like substance secreted by
honeybees and used by them in making their
honeycomb.
```

With justification on, 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.

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

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

For details on the margin-modifying control words, see "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199.

VERTICAL SPACE

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

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

For example,

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

are formatted as:

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

But the poodle was frightened
and ran away.

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

If the space generated by the .SK [Skip] control word occurs at the top of a column (or page), no blank lines are printed. However, a .SP [Space] control word always results in blank lines. For this reason, you may prefer to use the .SK [Skip] control word instead of the .SP [Space] control word whenever you need blank output lines.

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

```
.sp 2i
```

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

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

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

results in:

```
A Love Story



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

LINE SPACING

When you want to produce output that is double-spaced or multiple-spaced, you can indicate to SCRIPT/VS that, while formatting is to continue for each input line, extra line-spaces are to be inserted between each output line. For double-spacing, use the .DS [Double Space Mode] control word:

```
.ds
```

After the .DS [Double Space Mode] control word is processed, all output text lines have an additional line space between them.

Some of the spaces or skips that you have placed in the file are doubled as well: if you have a .SP 2 control word, then your output page has four line spaces. However, if you have specified blank vertical space in terms of inches, picas, ciceros, or millimeters, that space is not doubled: if you have a .SP 2i control word, then your output page has two inches of blank vertical space.

Double-spacing can be cancelled by the .SS [Single Space Mode] control word, which returns SCRIPT/VS output to normal single-line spacing.

You can also obtain additional space between output lines using the .LS [Line Spacing] control word. For example, you can specify

```
.ls 2
```

which results in two additional blank lines between each output line, or "triple-spaced" output. The .LS control word is a generalization of .DS and .SS; .LS 1 is equivalent to .DS, and .LS 0 is equivalent to .SS. Any of the three control words .SS, .DS, and .LS, cancels the other two.

In contrast, the .SL [Set Line Space] control word defines the vertical size of an output line. For example,

```
.sl .5i
```

results in each output line occupying a vertical space of one-half inch. Each output line is subject to line spacing as defined by the .SS, .DS, and .LS control words. For example,

```
.sl .5i
.ls 1
```

defines double-spacing mode where each output line is one-half inch deep. With these control words in effect,

```
.sp 2
```

results in two inches of vertical space. It is a request for a space of two lines, and each line is one-half inch deep. The resulting one inch of space is doubled because double-spacing is in effect. However, no matter what line spacing controls are in effect, an absolute space request always has the same effect. For example,

```
.sp .75i
```

causes SCRIPT/VS to space vertically as close to 3/4 inch as the resolution of the logical device allows.

POSITIONING LINES ON THE PAGE

Most line positioning is based on a displacement from the left margin -- a cumbersome way to format when you want text centered between the margins or aligned with the right margin (leaving a "ragged left edge"). SCRIPT/VS allows you to center text using the .CE [Center] control word, and to align text with the right margin using the .RI [Right Adjust] control word.

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

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

```
.ce Chapter 1
```

results in:

```
                    Chapter 1
```

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

```
.ri Chapter 1
```

results in:

```
                                                    Chapter 1
```

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

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

results in:

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

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

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

results in:

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

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

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

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

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

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

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

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

which results in:

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

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

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

is formatted as:

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

The space between the parts of split text can be left blank. However, you can specify a "fill string" that is repeated as often as necessary to fill the space between two parts of the split text. The fill string can be up to eight characters long. For example,

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

results in:

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

UNDERLINING AND CAPITALIZING

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

For example,

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

prints as:

Do not destroy this letter.

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

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

which results in:

This sentence contains a very important word for contemplation.

When lines are underscored automatically, SCRIPT/VS does not usually underscore blanks and punctuation marks. It does underscore letters, numbers, and some special characters. You can specify the characters to be underscored (or specify characters you do not want SCRIPT/VS to underscore) with the .UD [Underscore Definition] control word.

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

```
.up Chapter 10
```

results in:

```
CHAPTER 10
```

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

```
.uc preface
```

results in:

PREFACE

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

```
.us 3
Do not
destroy this letter
until
its expiration date,
which is January 31, 1978.
```

results in:

Do not destroy this letter
until its expiration date,
which is January 31, 1978.

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

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

results in:

This is capitalized for EMPHA-
SIS and EMOTIONAL impact.

FORCING A NEW PAGE

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

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

```
.pa
```

The .PA [Page Eject] control word causes a break. SCRIPT/VS prints the output line being constructed, then leaves the remainder of the current page blank. The .CP [Conditional Page Eject] control word is described in "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77.

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.

For a method of suppressing the numbering of introductory pages, see the discussion of the .PN [Page Numbering Mode] control word in "Chapter 4. Defining a Page Layout" on page 51.

SPECIFYING THE ODD OR EVEN PAGE

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

```
.pa odd
```

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

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

SPECIFYING PAGE EJECT MODE

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

```
.pa even on
```

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

```
.pa odd on
```

You can end page eject mode by issuing:

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

  ```
  .pa even on
  ```

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

  ```
  .pa odd off
  ```

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

  ```
  .pa 12
  ```

GUIDELINES FOR ENTERING TEXT AND CONTROL WORDS IN SCRIPT/VS

If you have not used a text processor before, or if you have used a text processor other than SCRIPT/VS, you may find some of the following tips useful when entering input for SCRIPT/VS files.

START ALL INPUT LINES IN COLUMN ONE

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

AVOID A TEXT PERIOD IN COLUMN ONE

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

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

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

prints as:

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

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

```
Study the following control words:
.li .li on
.DS,
.LI,
.PA, and
.IM.
.li .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 Mon-
day.
```

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

REMEMBER WHICH CONTROL WORDS CAUSE BREAKS

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

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

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

Many control words that provide format functions do not cause breaks. (A list of those that cause an implicit break is provided in Figure 25 on page 311.) The underscoring and capitalizing control words are good examples:

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

results in:

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

GROUP THE SCRIPT/VS CONTROL WORDS

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. Its effect is to allow SCRIPT/VS to separate an input line into two or more processable input lines. For example,

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

is the same as the two lines:

```
.sk
.ce on
```

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

Redefining the Control Word Separator

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

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

is processed as follows:

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

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

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

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

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

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

results, on output, in:

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

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

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

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

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

The .DC [Define Character] CW control word is also useful when you define symbols. For details on symbol definition, see "Chapter 11. Symbols in Your Document" on page 117. The example above shows how to use it to solve a text input problem.

Another way to avoid this problem is to use the control word separator (that is, the semicolon) to separate the control word from the text it operates on. For example,

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

is the same as

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

which results in:

```
Be careful; semicolons end input lines ...
```

In this case, the semicolon after "careful" was not a control word separator because it was not in a control word line. The control word separator ends an input line when the input line begins with a period (that is, it is a control word). Otherwise, the control word separator character is regarded as a text character.

COMMENTS IN SCRIPT/VS DOCUMENTS

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

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

Two Kinds of Comment

You can place comments in a SCRIPT/VS file with the

```
.cm Created:  12/21/75
.cm Updated:  3/3/76
```

The comments are control word lines. They can appear on lines with other control words (grouped and separated with control word separators):

```
.cm change format;.in +5;.ll -5
```

If you want comment lines that are not processed by SCRIPT/VS, you should enter them using .*:

```
.* SCRIPT/VS ignores this line; all of it.
```

The .* function, even though it begins with a period, is not considered a control word. Other control words on the same line are ignored, as is the control word separator character.

CHAPTER 4. DEFINING A PAGE LAYOUT

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

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

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

Figure 7 on page 52 shows the layout of a SCRIPT/VS output page. Control words used to specify the size or contents of each area are shown in parentheses.

BASIC PAGE DIMENSIONS

The output pages that SCRIPT/VS formats are designed to fit the form size of the logical output device (for more details, see the DEVICE option in "Chapter 2. Using the SCRIPT Command" on page 13). The default logical devices are defined for a form size of 8-1/2 by 11 inches. When SCRIPT/VS formats output for logical devices that specify a form size of 8-1/2 by 11 inches, each SCRIPT/VS page has the default dimensions of:

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

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

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

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

- The physical size of the paper on which you are printing SCRIPT/VS output.

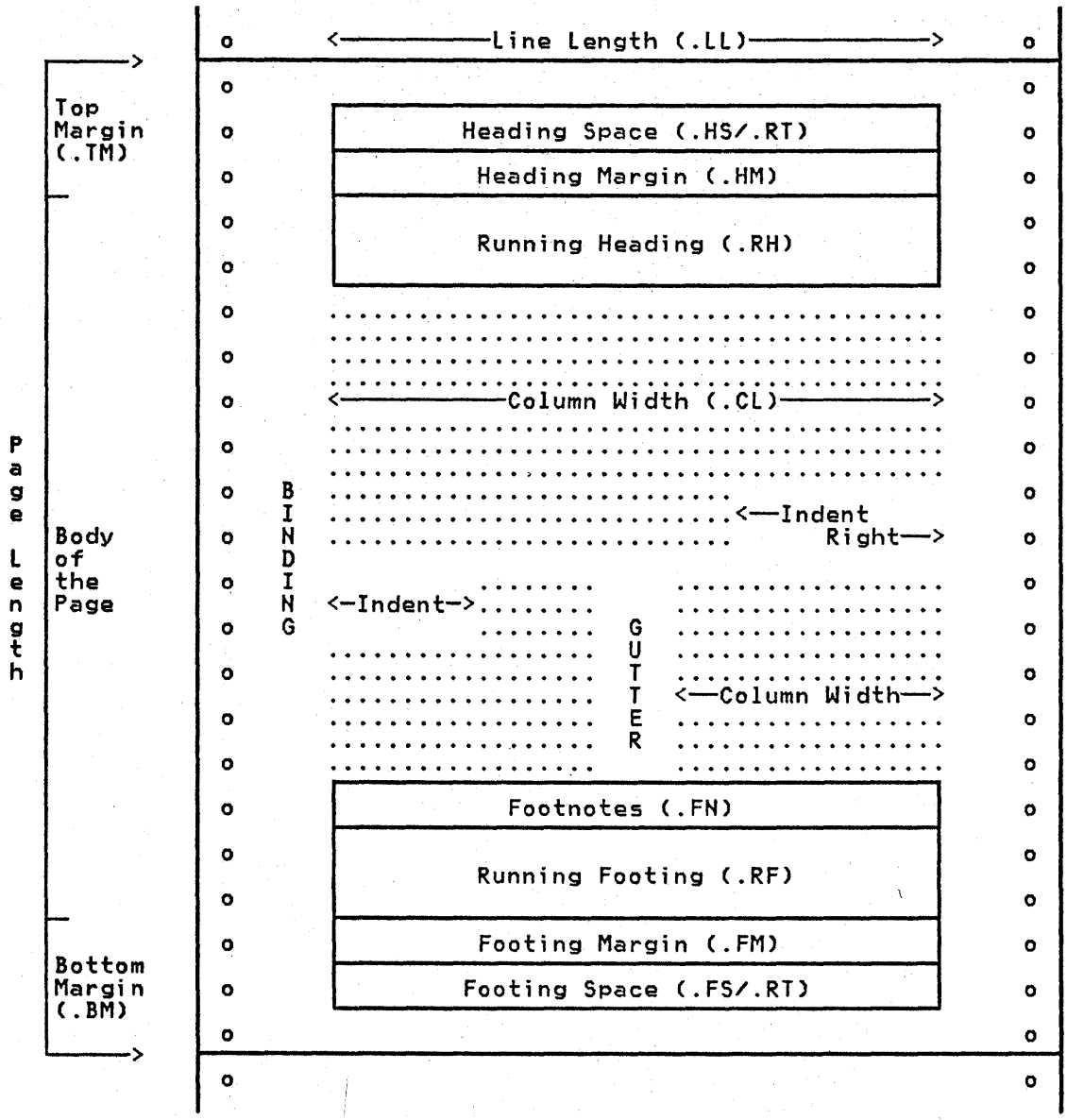


Figure 7. SCRIPT/VS Terms for Parts of the Page: Note that Top Margin and Bottom Margin include all the space on the paper that is accessible to SCRIPT/VS. For terminals and 1403-type printers, this includes the entire page. For 3800-type devices, Top and Bottom Margin do not include 1/2 inch on each side of the interpage perforation. This space is reserved by the 3800 Printer for accelerating and decelerating the paper when it is necessary to halt the paper path.

- The number of lines printed or typed per page on the output device.
- The 3800 Printer reserves one-half inch at the top and bottom of the page that is not included as part of the page length.

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

CHANGING THE PAGE LENGTH

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

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

You may need to adjust a page dimension to handle a special situation in your document. Instead of recalculating the margin values, you can specify an increment or decrement 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 margin, use the control word

```
.tm -3
```

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

CHANGING THE LINE LENGTH

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

```
.ll 8i
```

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

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

```
.ll -2i
```

decrements the line length by 2 inches.

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

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

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

TOP AND BOTTOM RUNNING TITLES

In addition to running headings and footings, described later in this chapter, SCRIPT/VS provides running titles. Running top titles appear in the heading space (part of the top margin); running bottom titles appear in the footing space (part of the bottom margin). When you do not specify otherwise, SCRIPT/VS provides a default top title that prints the page number in the upper right corner of the output page.

Use the .RT [Running Title] control word to specify titles. Running titles have many uses: they can indicate page numbers, chapter or section headings, document titles, form numbers, or almost anything you want.

Each title consists of three parts, separated by arbitrary delimiters, printed in the following positions according to the current line length:

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

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

To center the words "First Draft" at the top of every output page, specify

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

If you want this title to appear on the left side of every output page, specify

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

To cancel a title, make all three parts null:

```
.rt top ////
```

The above control word cancels top title line 1. To prevent all top titles from being printed, use .HS [Heading Space] 0.

The control words which define titles are:

```
.rt top even /X/Y/Z/          Even-page Top Title
.rt top odd /X/Y/Z/           Odd-page Top Title
.rt bottom even /X/Y/Z/       Even-page Bottom Title
.rt bottom odd /X/Y/Z/        Odd-page Bottom Title
```

If you want the same titles on both even- and odd-numbered pages, you can use the control words:

```
.rt top /X/Y/Z/           Top title
.rt bottom /X/Y/Z/        Bottom title
```

You can use any character that does not appear in the title as the delimiter. SCRIPT/VS assumes the first nonblank, nonnumeric character that follows "bottom," "top," "even," or "odd" is the delimiter. SCRIPT/VS processes the rest of the .RT control word line using that character as the delimiter. For example,

```
.rt bottom ¢¢OS/VS1 Operators Manual¢¢
```

If you use the EVEN or ODD parameter with the .RT [Running Title] control word, the titles you specify appear only on the even- or odd-numbered pages. For example, the titles:

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

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

The heading space and heading margin are allocated from the bottom of the top margin, contiguous with the body of the page. Similarly, footing space and footing margin are allocated from the top of the bottom margin.

The default page margin values, listed in Figure 29 on page 313, are defined to allow one line each for running top and bottom titles (heading and footing space), and two blank lines between the running titles and the body of the page (heading and footing margin).

The default top and bottom margin values allow 1/2 inch of blank space between the running titles and the edge of the form. This space is accessible to SCRIPT/VS for non-3800 devices, and is included in the top and bottom margin values. It is not included in the top and bottom margin values for 3800 logical devices.

If you want one blank line instead of two between the title and the running heading or text, specify

```
.hm 1
```

SCRIPT/VS will then provide one blank line between the lowermost top title (if any) and the first line of text on the page. The top margin value, however, does not change; the position of the top title changes.

Increasing a top or bottom margin does not automatically change the heading or footing margins. If you try to decrease a top or bottom margin to a value too small to accommodate the titles (that is, the values of the heading and footing space) plus the heading or footing margin, an error results.

SCRIPT/VS automatically numbers the pages of your output for you. When you want the page number to appear in a title, you use the page number symbol (an ampersand (&) unless otherwise defined) to represent the page number. SCRIPT/VS replaces the page number symbol with the current page number when the title is formatted. The default top title is:

```
.rt top ///PAGE &/
```

If you want to replace the default top title with one that has the page number centered, you can specify

```
.rt top //- & -//
```

If you need to use an ampersand as text in a running title, you can redefine the page number symbol to some other character with the .DC [Define Character] PS control word. For example, the sequence

```
.dc ps ¢
.rt odd bottom ///Page ¢/
.rt even bottom /Page ¢///
```

results in the page number appearing in the lower right of all odd-numbered pages and on the left of all even-numbered pages.

MULTILINE RUNNING TITLES

You can specify up to twelve running title line positions at one time in a SCRIPT/VS file. The twelve possible title line positions are:

- Six lines of titles for even-numbered pages
- Six lines of titles for odd-numbered pages

When you want to use multiline titles, there are two things you should consider:

- Specifying the text of the titles
- Allocating space for the titles to print

The .RT [Running Title] control words allow you to specify the order in which you want the lines printed. For top titles, title lines 1 through 6 are numbered from the top of the page down toward the first line of text. When you specify

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

the top titles are printed in the top margin as:

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

For bottom titles, title lines 1 through 6 are numbered from the bottom of the page up toward the last line of text. When you specify

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

the bottom titles are printed in the bottom margin as:

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

Since there are only six buffers available for storing title lines, top title line 6 uses the same buffer as bottom title line 1. In other words, each title line can be referred to as either a top title line or as a bottom title line. That is,

```
Top title line 1 = bottom title line 6
Top title line 2 = bottom title line 5
Top title line 3 = bottom title line 4
Top title line 4 = bottom title line 3
Top title line 5 = bottom title line 2
Top title line 6 = bottom title line 1
```

Therefore, you should take care when you specify multiline top and bottom titles, and when you cancel a title. If you would like to specify four top title lines and four bottom title lines, enter

```
.rt top 1 /Top title line 1///
.rt top 2 /Top title line 2///
.rt top 3 /Top title line 3///
.rt top 4 /Top title line 4///
.rt bottom 1 /Bottom title line 1///
.rt bottom 2 /Bottom title line 2///
.rt bottom 3 /Bottom title line 3///
.rt bottom 4 /Bottom title line 4///
```

However, you do not get eight unique title lines. Instead, the titles that are printed on your output pages are:

```
Top title line 1
Top title line 2
Bottom title line 4
Bottom title line 3
  ...
      body of the output page
  ...
Bottom title line 4
Bottom title line 3
Bottom title line 2
Bottom title line 1
```

This is because top title lines 3 and 4 were redefined as bottom titles 3 and 4 before they were used.

Allocating Space For Running Titles

No matter how many different title lines you specify, SCRIPT/VS prints only one top title line and one bottom title line on each page unless you allocate space for more titles. To print multiline titles on either the top or bottom of a page, you must tell SCRIPT/VS that you want more space in which to print the title lines. The .HS [Heading Space] and .FS [Footing Space] control words specify how many lines you want SCRIPT/VS to reserve for titles.

The top margin area is established by specifying an amount of space with the .TM [Top Margin] control word. For example,

```
.tm 8
```

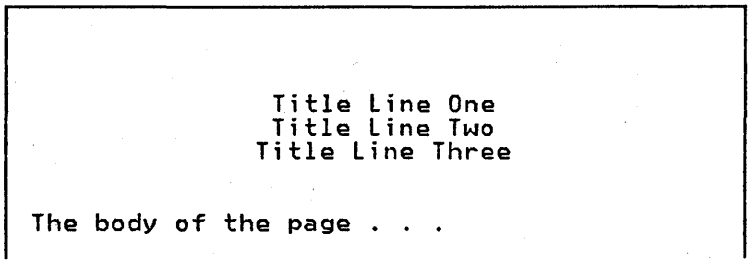
specifies a top margin of eight lines. The .HM [Heading Margin] control word tells SCRIPT/VS how many lines of the top margin to leave blank above the body of the page. To continue the example,

```
.hm 2
```

specifies two blank lines between the title lines and the body. Therefore, there are six lines available to contain title lines. However, you have to tell SCRIPT/VS how many of those lines are to actually contain titles. The remaining lines, between the title lines and the top of the form, are left blank. To conclude our example,

```
.hs 3
```

specifies that SCRIPT/VS can print the first three title lines (if they are specified), leaving three blank lines at the top of the form. The top margin, when formatted, looks like this:

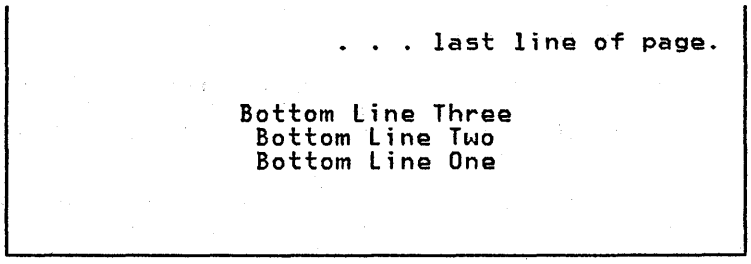


You specify the format of the bottom margin in the same way, using the .BM [Bottom Margin], .FM [Footing Margin], and .FS [Footing Space] control words.

For example, the control word sequence

```
.bm 8
.fm 2
.fs 3
```

results in a bottom margin that looks like this:



You can specify top titles such that one title line appears on all pages, centered, and another title line is designated specifically for an even- or an odd-numbered page. For example,

```
.hs 2
.rt top 1 //Preliminary Draft//
.rt top even 2 /Page &///
.rt top odd 2 ///Page &/
```

After these control words are processed, SCRIPT/VS prints the top title lines on the even-numbered pages as:

```
                        Preliminary Draft
Page 12
```

and the top title lines for odd-numbered pages as:

```
                        Preliminary Draft
                                                        Page 13
```

As a result, the page number is aligned with the outside margin when the pages are printed on both sides.

You can specify space for bottom titles the same way.

When you specify space for multiline titles, you should note the following restrictions:

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

As with other SCRIPT/VS control words, the most recent specification of a title line replaces any previous specifications for that line. Therefore,

```
.rt bottom 3 ////
```

cancels the previous specification of bottom title line 3 and top title line 4, including

```
.rt bottom 3 ...,
.rt bottom even 3 ...,
.rt bottom odd 3 ...,
.rt top 4 ...,
.rt top even 4 ..., and
.rt top odd 4 ....
```

You can specify the number of title lines within the top or bottom margin by adjusting the .HS [Heading Space] and .FS [Footing Space] control words. For example, if top title lines 1 through 6 have been specified and you want to print only top title line 1,

specify:

```
.hs 1
.fs 0
```

This causes SCRIPT/VS to provide one line for top titles and no space for bottom titles. SCRIPT/VS will print top title line 1 in the first line available for titles in the top margin (preceding the heading margin).

Note: If you specify six top title lines, and you specify heading and footing spaces of six, all six title lines print at the top and the bottom of the page. Any time the heading space plus the footing space is greater than six, some titles are used in both places.

Running Title Defaults

The default value for heading space and footing space is 1 line. Therefore, if you specify .RT [Running Title] with no numeric parameter, the title is printed (even if no heading or footing space control words have been specified).

To prevent titles (including the default top title) from appearing on your output pages, specify a zero amount of heading space and footing space:

```
.hs 0
.fs 0
```

RUNNING HEADINGS AND FOOTINGS

Part of the format of each of your output pages can be information at the top and bottom of the body, below the top margin and above the bottom margin. The information at the top of the body is called a running heading; information at the bottom of the body is called the running footing. Running headings and footings can be repeated for each page, and can be specified differently for even and odd pages. You can use running headings and footings to indicate page numbers, chapter or section titles, document titles, form numbers, security warnings, and anything else you want repeated on each page. Running headings and footings are processed exactly alike.

When you want a running heading or footing on each page, use the .RH [Running Heading] or .RF [Running Footing] control word. For example,

```
.rh on
text and control words for the running heading
.rh off
   or
.rf on
text and control words for the running footing
.rf off
```

When you want different running headings or footings for even and odd pages, you can specify

```
.rh odd
text and control words for the odd-page running heading
.rh even
text and control words for the even-page running heading
.rh off
```

When you want to cancel a previously-set running heading or footing, you can either respecify it with new text and control words:

```
.rf odd
new text and control words
for the odd-page running footing
.rf off
```

or you can eliminate it completely:

```
.rf odd cancel
```

The running heading is placed beginning in the line (or lines) following the top margin. Similarly, the running footing is placed beginning in the line (or lines) that immediately precedes the bottom margin.

Within the running heading and footing definition (that is, the input lines bounded by .RH ON and .RH OFF), you can specify many of the SCRIPT/VS control words. For example, if you want to have two lines of space after a running heading, you can issue

```
.rh on
text of running heading
.sk 2
.rh off
```

SCRIPT/VS formats your running heading and footing (in single-column mode) as it would any other group of input lines. That is, the text is concatenated and justified unless you specify otherwise. Some of the SCRIPT/VS control words you specify are processed once (the .IM [Imbed] control word, for example). Most of the control words, however, are saved with the running heading or footing definition. Each time SCRIPT/VS has to build a running heading or footing, it processes the "saved" definition. If you want to modify a running heading or footing, you must redefine it in its entirety.

Control words you specify for the running heading or footing definition are effective only for the running heading or footing, and not also for the body of the page. Most control words specified outside the running heading or footing definition have no effect on the running heading or footing. For example, you can specify an unformatted running heading in singlespace, though doublespace is specified for the body of the text, by issuing:

```
.ds
.rh on
.fo off[6]
text and control words for the running heading
.rh off
```

You can use indention control words to position your running heading or footing. If you do not specify indention, the running heading and footing are printed flush left (aligned with the left margin). You can also position the running heading or footing with the .CE [Center] control word to center it, with the .RI [Right Adjust] control word to align it with the right margin, and with the .SX [Split Text] control word to align parts with both the left and right margins.

You can put parts of your running heading or footing in uppercase, and have words underscored, by using the .UP [Uppercase], .US [Underscore], and .UC [Underscore and Capitalize] control words. For example,

```
.rh on
.us This part is underscored
and this part isn't.
.rh off
```

results in a heading that looks like:

This part is underscored and this part isn't.

---

[6] ".FO ON" is not necessary at the end of the definition, because the page environment before the definition is restored during formatting. See "Chapter 8. The SCRIPT/VS Formatting Environment" on page 97 for details about the formatting environment.

PAGE NUMBERS IN HEADINGS AND FOOTINGS

SCRIPT/VS can automatically number the pages of your output for you. When you want the page number to appear in a running heading or footing, you must use the page number symbol (an ampersand (&) unless otherwise defined with the .DC [Define Character] PS control word) within the text of the running heading or footing to represent the page number. SCRIPT/VS replaces the ampersand with the current page number when the page is printed. For example,

```
.rf on
.ri PAGE &
.rf off
```

results in a running footing aligned with the right margin that looks like

PAGE 61

You can specify a running heading that has the page number centered on the page. For example,

```
.rh on
.ce -- Page & --
.rh off
```

results in a heading that looks like:

-- Page 61 --

If you need to use an ampersand as text in the running heading or footing, you can redefine the page number symbol to some other character using the .DC [Define Character] PS control word. For example, the heading "Topsy & Turvy" can be specified with this sequence:

```
.dc ps ¢
.rh on
.ce Topsy & Turvy -- Page ¢
.rh off
```

resulting in a running heading that looks like:

Topsy & Turvy -- Page 61

The .DC [Define Character] PS word, when processed, affects all headings, footings, and titles, including those that have been defined previously. For example, if a title has been set with

```
.rt top ///Page &/
```

and, later, SCRIPT/VS processes

```
.dc ps ?
```

the top title must be reset to

```
.rt top ///Page ?/
```

Otherwise, the current page number will not be substituted in the top title for all pages following the .DC PS control word.

Note: The page number symbol is used only for substituting page numbers in headings, footings, and titles. SCRIPT/VS always recognizes the ampersand as "the current page number" when it occurs on the right-hand side of a symbol definition, regardless of the current page number symbol:

```
.se here = &
```

However, in normal body text an ampersand by itself is always considered to be part of the text.

WHERE TO DEFINE HEADINGS, FOOTINGS, AND RUNNING TITLES

If you are working on a document that has different sections or chapters, you may need to change running headings and footings frequently. In these instances, put .RF [Running Footing], .RH [Running Heading], and .RT [Running Title] control words before a .PA [Page Eject] control word (or a control word that causes a page eject).

Keep in mind that SCRIPT/VS will always apply the new heading or footing to the page following the current page. SCRIPT/VS does not use the running heading, footing, and title you specify until the next page eject occurs.

A Heading on Page One

Normally, SCRIPT/VS processing begins with the first text line on the first page of output. Running headings and footings usually don't appear on page 1. If you want the running heading or footing to print on the first page of output, you must enter the .RH [Running Heading] or .RF [Running Footing] control word before any text line or any control word that would cause the page to be started. For example, if you want the running heading

```
.rh on
Company Confidential Document
.rh off
```

to appear on the title page as well as every other page, the definition should be entered first.

PAGE NUMBERS

You can use the .PA [Page Eject] control word to both eject to a new page and to reset the page number, and you can use the .DC [Define Character] PS control word to respecify the current "page number symbol."

With the .PN [Page Numbering Mode] control word, you establish whether or not you want page numbering, and what kind of page numbering you want.

- If you do not want SCRIPT/VS to print the page number on subsequent output pages, and you want SCRIPT/VS to continue incrementing the page number internally, you can specify:

  ```
  .pn off
  ```

- If you do not want SCRIPT/VS to print the page number on subsequent output pages and also not to increment the page number internally, you can specify

  ```
  .pn offno
  ```

- To reset the OFF and OFFNO parameters, you can specify

  ```
  .pn on
  ```

DECIMAL PAGE NUMBERS

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

```
.pn frac
```

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

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

```
.pn norm
```

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

ROMAN NUMERAL PAGE NUMBERS

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

```
.pn roman
```

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

```
.pn arabic
```

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

Sometimes, the page numbering scheme for your document requires that each page number begin with a prefix. For example, you want page numbers for chapter one to begin with the prefix "1-", page numbers for chapter two to begin with "2-", and so on. You can use the PREF parameter of the .PN [Page Numbering Mode] control word to attach a prefix to each page number. For example,

```
.*** Beginning of your document ***
.pn pref 1-
.*** Text of Chapter 1 ***
.pn pref 2-
.pa 1
.*** Text of Chapter 2 ***
   ...
```

When this sequence is processed, all pages in Chapter 1 are numbered 1-1 through 1-n. A running footing for chapter 1 on page 8 that includes the "current page number" symbol, &, is printed as

```
Chapter 1 footing                                        Page 1-8
```

That is, the page number symbol & is replaced by the entire page number, and is composed of both the prefix (if any) and the page number. If the initial processing of chapter 2 respecifies the running footing and prefix, the footing for the first page of chapter two might be printed as

```
Chapter 2 footing                                        Page 2-1
```

In the above coding sequence, the .PA [Page Eject] control word is used to eject to a new page for the next chapter and to reset the page number counter to 1, so the first page for chapter 2 is numbered 2-1 on the output page. If you do not want to reset the page number counter each time a new chapter is processed, omit the .PA [Page Eject] control word's numeric parameter. The PREF operand is useful for texts that begin a new numbering scheme with each section.

All of the above operands affect running titles, running headings, and running footings that have the page number symbol (usually an ampersand (&)) in them.

Each table of contents entry generated by the head level (.Hn) control words contains the page number in the format specified by the .PN [Page Numbering Mode] control word. For information on creating tables of contents in SCRIPT/VS, see "Chapter 6. Head Levels and Table of Contents" on page 71.

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

DEFINING MULTICOLUMN LAYOUT

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

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

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

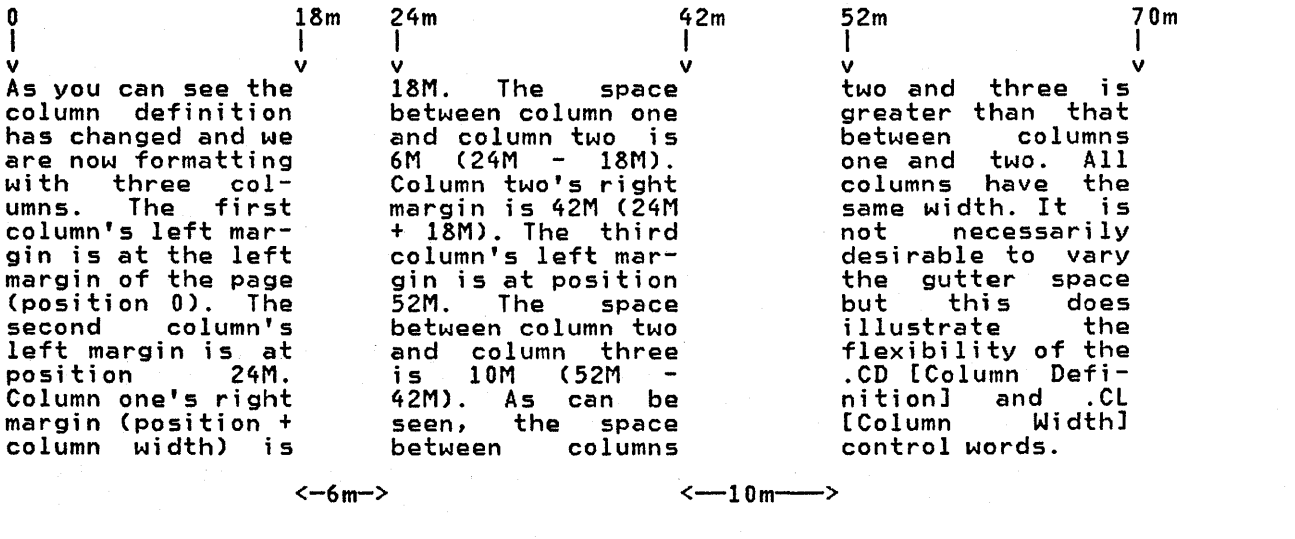
- .CD [Column Definition], which provides for
  - Specifying the number of columns
  - Specifying the left margin position for each column
- .CL [Column Width] which provides for
  - Specifying the column width

Note: All columns must be the same width but the space between columns may vary.

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

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

and would produce this effect:



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

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

```
.ll 70m
```

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

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

Notes:

- The .CD [Column Definition] and .CL [Column Width] control words take effect immediately on the next output line.
- "Appendix D. Formatting Considerations for the 3800 Printer" on page 337 contains additional considerations regarding the definition of multicolumn layout for the 3800 Printer.

PAGE SECTIONS AND SECTION BREAKS

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

Top Title

Running Heading

Body Text

Footnote

Running Footing

Bottom Title

When a page section is completely formatted, the appropriate output device module is called to process the accumulated lines and send them to the output destination. This is called a "section break."

See Figure 21 on page 298 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 footnote if any. See "Chapter 4. Defining a Page Layout" on page 51 for details on these component space values.

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

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

A section break occurs when:

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

  .PA [Page Eject]

  .CP [Conditional Page Eject]

  .CB [Column Begin]

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

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

  .MC [Multicolumn Mode]

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

  .SK [Skip] with the "P" parameter

  .SP [Space] with the "P" parameter

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

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

COLUMN POSITIONS

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

This first section was produced by specifying

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

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

This second section was produced by specifying

```
.cd 2
```

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

This third section was produced by specifying

```
.cd 3
```

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

COLUMN WIDTH

Column width remains in effect until explicitly changed by a .CL [Column Width] control word. The formatter attempts to build each output line to fill the column width by concatenating short input lines or folding long input lines. A partially full output line is padded to full width by justification. Line concatenation and justification are controlled by:

.CE, .CO, .JU, .FO, and .RI control words

For details see "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199.

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

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

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

With a line length of 68M, and a gutter of 4m, two columns would be defined as:

```
.cd 2 0 36m
.cl 32m
```

This two column data is formatted with a column width of 32M to maximize the use of the space on the page. As can be seen, there is little wasted. This example is meant to show typical usage. Normally columns will be laid out to be as dense as possible for economic page use. Readability is also a factor in column definition.

With the same line length and gutter size three columns would be defined as:

```
.cd 3 0 24m 48m
.cl 20m
```

This three column data is formatted with a column width of 20M to maximize the use of the space on the page. As can be seen, there is little wasted. This example is meant to show typical usage. Normally columns will be laid out to be as dense as possible for economic page use. Readability is also a factor in column definition. In the three column example the columns are a little narrow.

STARTING A NEW COLUMN

If the current column is forced to end, and a new page is not started, that column and all columns to the left of it are ineligible for column balancing. The following SCRIPT/VS control words may be used to end a column before it is full.

- .CB [Column Begin] ends the column unconditionally.
- .CC [Conditional Column Begin] ends the column based on the space remaining in the column.
- .CP [Conditional Page Eject] ends the column and causes a page eject based on the space remaining in the column.

Use the .BC [Balance Columns] control word to enable or disable column balancing. If column balancing is OFF, no columns are balanced. If column balancing is ON, all eligible columns are balanced whenever a section break occurs.

Blocks of text, such as figures or tables, can be kept together and balanced as a unit. Text lines in a block will not be split across columns. See "Chapter 7. Additional Formatting Features of SCRIPT/VS" on page 77 for details on use of the .KP [Keep] control word.

SUSPENDING AND RESUMING MULTICOLUMN PROCESSING

The .SC [Single Column Mode] control word

- Saves the current column definition
  - Column width
  - Number of columns
  - Column positions

- Defines a single column with a column width equal to line length.

The .MC [Multicolumn Mode] control word

- Restores the last saved column definition

The .SC [Single Column Mode] and .MC [Multicolumn Mode] control words are always paired; you must specify the .SC control word before you specify .MC control word.

SCRIPT/VS provides an automatic table of contents facility which is based on the concept of "head levels." When you create a SCRIPT/VS file, you can enter topic headings[7] 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[8]. When SCRIPT/VS processes a .H0 - .H6 [Head Level 0 - 6] control word:

- The text portion of the heading is formatted according to characteristics associated with the head level. The formatting may include such things as spacing above and below the heading, capitalization, underscoring, and font.
- If the heading requires a table of contents entry, the heading's text and current page number are saved in a temporary file called DSMUTTOC.

For example, if you enter a topic heading as

```
.h3 Symptoms
```

SCRIPT/VS uses the characteristics for a level three heading to format the heading's text on the page. SCRIPT/VS also creates an entry in the table of contents file for the topic "Symptoms" and the page number on which it appears. All the headings entered with the .H3 control word are formatted in the same way.

If you use SCRIPT/VS head level control words exclusively, you need not create a table of contents manually. When you revise or reorganize your document, the table of contents is automatically updated.

CHARACTERISTICS OF HEAD LEVELS

Head levels are commonly associated with the following sections of a document:

.H0 Table of Contents entry only

.H1 Chapter

.H2 Major section

.H3 Minor section

.H4 Topic

.H5 Inline heading

.H6 Inline heading

The .DH [Define Head Level] control word allows you to redefine the characteristics of any head level to suit your needs. The characteristics are:

- Whether the heading in the text should begin on a new page or cause a break.

---

[7] The word "heading" is used in this section to mean a topic heading that is printed as part of the text.

[8] GML and EasySCRIPT provide tags with similar names and functions. This discussion is concerned only with the SCRIPT/VS control words.

	.H0	.H1	.H2	.H3	.H4	.H5	.H6
Page eject before heading		yes					
Heading Out-justified [1]		yes					
Line skips before heading	0	0	3	3	3	1	1
Line spaces after heading	0	5	2	2	2	0	0
Heading underscored		yes	yes		yes	yes	yes
Heading capitalized		yes	yes	yes		yes	
Heading will cause a break		yes	yes	yes	yes		
Table of Contents entry	yes	yes	yes	yes			
Table of Contents entry only	yes						
Skip before T.O.C. entry		yes					
Table of Contents indention	0	0	0	2	4	6	8

[1] The heading will be right-justified on the page if the page number is odd.

Figure 8. Summary of Default Head Level Characteristics: This table lists the default characteristics of the .Hn [Head Level n] control words. The .DH [Define Head Level] control word allows you to redefine any of these characteristics to suit your needs. Note that by default all headings are printed in the current font.

- Whether the heading should be right-aligned if it occurs on an odd page.
- Whether the heading should be capitalized or underscored, and the font it is to be printed in.
- The amount of vertical space which precedes and follows the heading.
- Whether or not a table of contents entry is to be created, and if so, what its indention should be.
- Whether only a table of contents entry should be created, placing no heading at all in the text.

Figure 8 lists the default characteristics of the .H0 - .H6 [Head Level 0 - 6] control words.

SPACING AND PAGE EJECTS

Headings are printed in the current column when there is enough room for the heading and at least two lines of text that follow it in the body of the document. If there is not enough room, the heading is placed at the top of the next column.

The line spaces that follow topic headings are conditional. If the heading is followed by more vertical space (whether caused by the .SP [Space] or .SK [Skip] control words, or another head level), only the larger of the two spaces is used, not the sum.

Head levels that are defined to begin new pages cause page ejects only if SCRIPT/VS is not already at the top of a page. This can be useful:

- To assign a page number to the output page with a .PA [Page Eject] control word.
- To eject to a new even- or odd-numbered page with the .PA EVEN or .PA ODD control words.

DEFINING HEAD LEVELS

The .DH [Define Head Level] control word allows you to redefine the characteristics of any head level. The .DH control word accepts parameters that describe head level characteristics, such as SPAF (SPace AFter) to set the amount of vertical space to follow the heading, and TC to indicate that a table of contents entry is to be generated. For example,

```
.dh 3 skbf 1 us
```

will redefine the .H3 head level to provide only one line of space before the heading, and to underscore the heading. The .DH control word is described in "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199.

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.
- If the head level function provided by the .H0 - .H6 [Head Level 0 - 6] control words are basically suitable, but you want to augment them, you can take advantage of SCRIPT/VS internal head level macro definitions.

  The names of the head level macros are DSMSTDH0 through DSMSTDH6 for the .H0 - .H6 [Head Level 0 - 6] control words, and DSMEZSH0 through DSMEZSH6 for EasySCRIPT head level tags.

  When a head level control word is first processed, SCRIPT/VS builds a corresponding macro to provide the requested function. SCRIPT/VS then processes that macro whenever the head level control word is encountered. This internal macro can be modified using the .DM [Define Macro] control word.

For details, see "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137.

THE TABLE OF CONTENTS

When SCRIPT/VS processes a head level control word that requires a table of contents entry, it writes an entry in the DSMUTTOC file. The entry contains the following information:

- The text of the heading.
- The page number of the page on which the heading appears.
- The revision code character in effect when the heading was processed.

All entries in the table of contents file are inserted by .PT [Put Table of Contents] control words in the head-level macros. Each entry consists of one or more text lines and standard control words such as .SX [Split Text], .IN [Indent], .OF [Offset], and .RC [Revision Code].

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.

ADDING LINES TO THE TABLE OF CONTENTS

You can place lines directly into the table of contents with the .PT [Put Table of Contents] control word.

The .PT [Put Table of Contents] control word causes the text line to be written into the file DSMUTTOC along with the current page number as a .SX [Split Text] control word. For example, the input line:

```
.pt Sail and Rudder
```

will cause the following control word to be written into DSMUTTOC:

```
.'SX /Sail and Rudder/ ./74/
```

When the DSMUTTOC file's input lines are processed, the line appears in the table of contents as:

```
Sail and Rudder . . . . . 74
```

You can insert any SCRIPT/VS control word into the table of contents with the .PT control word. If the "text line" part of the .PT control word begins with a period (with only one blank between .PT and the text line), SCRIPT/VS inserts it directly into the DSMUTTOC as a control word, rather than as the text of a .SX [Split Text] control word. For example,

```
.pt .h3
```

inserts a .H3 control word into the table of contents.

If the line of text you want to enter into the table of contents begins with a period, begin the line with a leading blank so that SCRIPT/VS will not interpret the line as a control word, but will include the page number with the line in the table of contents. For example,

```
.pt    .h3
```

inserts the control word

```
.'SX F /.h3/ ./74/
```

into the table of contents.

PRINTING THE TABLE OF CONTENTS

Use the .TC [Table of Contents] control word to imbed the DSMUTTOC file. When .TC is encountered, SCRIPT/VS:

- Ejects to a new page, if it is not already at the top of a page.
- Prints the word "CONTENTS," unless otherwize specified with the .PT control word.

  If you want a different title for the table of contents page, you can specify it as

  ```
  .tc Table of Contents
  ```

  If you don't want a title at all, specify

  ```
  .tc /
  ```

- Formats the DSMUTTOC file according to the SCRIPT/VS environment in effect when the .TC control word is processed, as modified by formatting controls inserted in the DSMUTTOC file.

  The DSMUTTOC file is not deleted until the next time a new table of contents file is started.

TWOPASS CONSIDERATIONS

If you place the .TC [Table of Contents] control word at the beginning of your input file, you must use the TWOPASS option of the SCRIPT command to produce a complete table of contents. Otherwise, the DSMUTTOC file will be empty when the .TC control word is encountered. For details about the TWOPASS option, see "Chapter 2. Using the SCRIPT Command" on page 13.

In order to have correct page numbers in the table of contents, pages must be numbered the same way on both passes. On the first pass, the table of contents is empty. On the second pass, it can contain several pages of information. Because SCRIPT/VS doesn't know how many pages will be required for the table of contents, it numbers the pages following the table of contents the same way on both passes.

You can tell SCRIPT/VS the number of page numbers to reserve for the table of contents. For example, you can reserve six pages if the table of contents is to occupy pages 3 through 8. The page number range you reserve has nothing to do with how many actual pages the table of contents will occupy: it only establishes the page number of the page that follows the table of contents page.

For example, if the table of contents will require three pages, you can reserve the current page number and the next two page numbers by specifying:

```
.tc 3 Table of Contents
```

If the document is formatted with the TWOPASS option, SCRIPT/VS will allow page numbering to continue sequentially following the table of contents if the page number is explicitly reset with a .PA [Page Eject] or .PN [Page Numbering Mode] control word before any head level or .PT [Put Table of Contents] control word is encountered that requires knowledge of the page number.

You can precede the .TC [Table of Contents] control word with other SCRIPT/VS control words:

- To number table of contents pages with Roman numerals, use the .PN [Page Numbering Mode] control word:

  ```
  .pn roman
  ```

- To put bottom titles on each table of contents page, use the .RT [Running Title] control word:

  ```
  .rt bottom even /Contents &///
  .rt bottom odd  ///Contents &/
  ```

- To ensure that the first page of the table of contents starts on an odd-numbered page, use the .PA [Page Eject] control word:

  ```
  .pa odd
  ```

CHAPTER 7. ADDITIONAL FORMATTING FEATURES OF SCRIPT/VS

In addition to formatting your document as described in previous chapters, SCRIPT/VS allows you to:

- Use special characters for output, even if not available as keyboard characters.
- Establish characters with special meaning for SCRIPT/VS.
- Keep blocks of text together so that, on output, the kept-together material appears entirely in one column.
- Define footnotes to be placed at the bottom of the page.
- Mark all updated material in a review draft, so readers can identify the modified material.
- Draw rectangular boxes with horizontal and vertical interior lines.
- Intermix different fonts when printing your document on the IBM 3800 Printing Subsystem.

USING SPECIAL CHARACTERS

If you are using a terminal with a standard keyboard, you do 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 might want to be able to print a bullet and other special characters as well.

One way to enter special characters into a file is to use CMS or TSO commands while editing.

SCRIPT/VS provides another method for printing special characters. You can specify one of your keyboard characters to be translated to the special character, using the .TR [Translate Character] control word.[9]

For example,

```
.tr * af
```

Each occurrence of an asterisk in your file is translated, on output, to the bullet character (•) which has the hexadecimal code AF. For example, the input line

```
* How about this for a paragraph?
```

results in:

• How about this for a paragraph?

"Appendix C. Fonts Supplied with SCRIPT/VS" on page 329 and Figure 37 on page 320 illustrate the various character sets available and their hexadecimal codes. You can use these charts when you want to translate characters such as:

- Brackets: [ ]
- Braces: { }
- Algebraic and logical symbols: < ≤ ≠ = ≥ > | ¬ ±

---

[9] See "Appendix D. Formatting Considerations for the 3800 Printer" on page 337 for special considerations regarding the use of .TR within documents that will be printed on the IBM 3800 Printing Subsystem.

- Superscript numerals: ⁰ ¹ ² ³ ⁴ ⁵ ⁶ ⁷ ⁸ ⁹
- Bullets for square-shooters: ▪
- Box characters: ┌ ┐ └ ┘ ─ │ ┬ ┴ ┼ ├ ┤

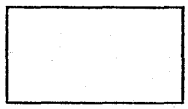
You can specify as many translation pairs with one .TR [Translate Character] control word as your input line allows. For example,

```
.tr a ┌ b ┐ c └ d ┘ - ─ | │
```

specifies the corners, vertical bar, and dash used for drawing a box. Each special character is specified as its character code: hexadecimal AC is the code for "┌", hexadecimal BC is the code for "┐", and so on. When the characters "a", "b", "c", "d", "|", and "-" appear in a subsequent output line, they are replaced with the special character's hexadecimal code. For example, the input lines

```
a----------b
|          |
|          |
|          |
d----------c
```

result in



When you use hyphens to draw boxes, you can translate them to the special dash (hexadecimal BF), which aligns with the corner symbols and extends further than a hyphen to create an uninterrupted line.

To cancel translation of all previously specified input characters, use the .TR [Translate Character] word with no parameters:

```
.tr
```

When you have many special characters specified, you can reassign or cancel some of the special characters 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 translation for "(": the character is to be translated to itself.)

Note: During the time a translation is in effect, every occurrence of the translated 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.

The actual translation is done at various times in the formatting process, depending on the requirements of the logical output device. The latest time translation can be done is when a line is finished and is placed in a column. You should assume, therefore, that a translation will be needed until the next break is done, whether that happens naturally because a line is full or is forced by a control word that causes a break.

INPUT CHARACTER TRANSLATION

SCRIPT/VS also performs character translation on input lines. The .TI [Translate Input] control word allows you to make characters that are unavailable on your terminal effectively part of your input file. For example, the IBM 3270 terminal does not have a tab

key. However, an available character (such as the "¬") 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 to:

- Use hexadecimal codes for the special character rather than the character itself. For example,

  ```
  .ti % $
  ```

  translates all occurrences of % to $. However, you cannot restore the percent-sign character by subsequently issuing

  ```
  .ti % %
  ```

  because that input line is translated to ".ti $ $" before being processed. However, you can restore % to itself with

  ```
  .ti 6C 6C
  ```

  Be careful, though. Remember that 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 "F0" in the above control word would be translated to "F@" before the control word is processed.

- 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
  ```

There are several advantages to using the .TI [Translate Input] control word instead of the .TR [Translate Character] control word:

- You can create a symbol that will be replaced with a special character. Each time you need the special character, you can insert the symbol. You don't have to first translate a keyboard character to the special character, insert the keyboard character (now with its special meaning), and then reset the keyboard character to its usual meaning.

  For example, to set the symbol "&bul" to be the bullet special character (hexadecimal AF), specify

  ```
  .ti % AF
  .se bul = '%'
  .ti 6C 6C
  ```

The percent sign (hexadecimal 6C) was set to specify the special character's hexadecimal code. Next, the symbol "&bul" was set to the percent sign (translated, on input, to the special character). Finally, the percent sign hexadecimal code is restored to its original character meaning.

- You can use the best character available on your output device for the special character. For example, on the IBM 1403 Printer the only vertical line available for drawing boxes is the "or" sign (|). However, some of the IBM 3800 Printing Subsystem fonts include a longer vertical line (|). You can set the symbol "&vline" to the longest vertical line available on your printer with:

  ```
  .ti 6C 4F
  .if x&$PDEV = x3800 .ti 6C FA
  .se vline = '%'
  .ti 6C 6C
  ```

  SCRIPT/VS will conditionally choose the proper vertical bar character for the printer currently being used. For details on how this works, see the discussion of the .IF control word in "Chapter 9. Conditional Processing" on page 99.

- You can create a symbol that will be replaced with a character that has special meaning to SCRIPT/VS. For example, if you use the "¬" keyboard character (on the IBM 3270 Terminal) as a tab (hexadecimal 05) you cannot also enter it from the keyboard when you mean "not".

  You can, however, define a symbol to use instead of the "¬" keyboard character by specifying:

  ```
  .ti 6C 5F
  .se notsym = '%'
  .ti 6C 6C
  ```

  SCRIPT/VS substitutes the '¬' character for the &notsym symbol in the input line. The '¬' character specified in this way can be used as a text character and as a logical "not" character with the .IF [If] control word.

You can create a SCRIPT/VS file that establishes the meaning of each special-character symbol name you create. When you want to use special characters in a document, you can imbed the special-character file at the front of the document and use the special-character symbol names throughout the document.

For example, to set the symbol name "&degree" to mean "°", "&lbrace" to mean "{", and "&rbrace" to mean "}", you can specify

```
.ti ∂ A1
.se degree = '∂'
.ti 7C 8B
.se lbrace = '∂'
.ti 7C 9B
.se rbrace = '∂'
.ti 7C 7C
```

The first .TI [Translate Input] control word redefined the "∂" sign to hexadecimal A1, or ° (that is, a degree: temperature or navigational).[10] The .SE [Set Symbol] control word sets the value of the symbol &degree to the "degree" character.

The second .TI control word resets the "∂" sign to hexadecimal 8B, or the left brace ({). The symbol &lbrace is set to the left brace character.

---

[10] Some fonts do not include the degree sign. However, the superscript zero (⁰) is very similar in shape and position, and is used in this manual for both the degree sign and the superscript zero.

The third .TI control word resets the "@" sign to hexadecimal 9B, and the symbol &rbrace identifies the right brace (}). The last .TI control word restores the "@" sign to itself.

The input lines

```
The schooner's position is &lbrace.149&degree 30' W by
17&degree 30' S&rbrace..
```

result in

```
The schooner's position is {149° 30' W by 17° 30' S}.
```

To restore all characters to normal, use the .TI [Translate Input] control word with no parameters.

CHARACTER TRANSLATION FOR TERMINAL OUTPUT

If you have used the .TR [Translate Character] control word and direct the SCRIPT/VS output to your terminal, then the special characters might not be displayed in the output. The positions occupied by the translated characters appear to be blanks, because there are no equivalent characters on the terminal. You can circumvent this proofreading problem in some cases by making the translation conditional, using the .IF control word.

```
.if SYSOUT eq PRINT .tr * af
```

This control word line results in output translation of asterisks (*) only if output is going to the printer.

This technique is also useful for hyphens, since the dash (hexadecimal BF) that aligns with corner symbols is not printed by most terminals. If you specify

```
.if SYSOUT eq PRINT .tr - BF
```

the hyphens will be translated to dashes only when the file is printed. (You should remember to translate the hyphen back to its normal state for text, since the dash (hexadecimal BF) does not align properly when used for interword hyphenation.)

DEFINING SPECIAL CHARACTERS THAT AFFECT SCRIPT/VS PROCESSING

You can define characters with special meaning to SCRIPT/VS using the .DC [Define Character] control word. The special characters are:

The array element separators, which are placed between elements of a symbol array. See "Chapter 11. Symbols in Your Document" on page 117 for details.

The continuation character, which allows single words to span input lines. The continuation character is described below.

The control word separator, which allows several SCRIPT/VS control words to be "stacked" on a single input line. See "Chapter 3. Basic Text Processing" on page 29 for details.

The GML delimiter, or alternate symbol delimiter, which is used to identify GML tags. See "Chapter 11. Symbols in Your Document" on page 117 for details.

The page number symbol, which is replaced with the current page number wherever it appears in running titles, running headings, and running footings. See "Chapter 4. Defining a Page Layout" on page 51 for details.

Punctuation characters which are recognized during spelling verification. See "Chapter 15. Automatic Hyphenation and Spelling Verification" on page 157 for details.

The required blank, which is not recognized as an interword blank during justification, and is translated to an ordinary blank on output.

Full stop characters which indicate the end of a sentence. See "Chapter 3. Basic Text Processing" on page 29 for details.

Word delimiters which delimit words for purposes of spelling verification. See "Chapter 15. Automatic Hyphenation and Spelling Verification" on page 157 for details.

The parameters of the .DC [Define Character] control word are described in detail in "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199.

THE CONTINUATION CHARACTER

SCRIPT/VS ordinarily appends an interword blank to the last word on a text input line. However, if the last character on a text input line is the continuation character, it is removed and the interword blank 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+
.sf
.bf GB12
light+
.pf
ed characters.
```

will produce this output:

A few high**light**ed characters.

If the formatter control or text which follows the continued word causes a break, continuation is cancelled for that line. The control words that cause breaks are listed in Figure 25 on page 311.

There is no default continuation character; it must be explicitly set before it can be used.

ENSURING THAT BLOCKS OF TEXT STAY TOGETHER

When you place a figure, diagram, or chart in your text, you want to ensure that the figure is printed on one page. You may also want to ensure that all of a paragraph or group of lines prints in the same column. Use the .KP [Keep] control word to tell SCRIPT/VS which lines you want kept together. For example,

```
.kp on
These text lines will all
appear in the same column,
regardless of page ejects
or column balancing.
.kp off
```

Keeps can be separated into two broad categories:

- Inline keeps, started with .KP INLINE, .KP v, or .KP v + v, ensure that a designated vertical amount of formatted text remains together. These keeps do not disturb the formatting of text in any way. Inline keeps that specify an amount of space are automatically ended when the requested amount of space has been formatted. They may also be ended with .KP OFF, before the requested amount is exhausted.

- Those started with .KP ON, .KP FLOAT or .KP DELAY. Each of these keeps must be explicitly ended.

  - Regular Keeps, started with .KP ON, place the kept text in the current column if it will fit. Otherwise, a column eject is performed and the text is placed at the top of the new column.

  - Floating Keeps, started with .KP FLOAT, save the kept text for the top of the next column if it does not fit in the current column, and format text following the keep in the input file into the current column.

  - Delayed Keeps, started with .KP DELAY, are always placed at the top of the next column, regardless of whether they fit in the current column. As with floating keeps, text following the keep in the input file may be moved ahead of it in the output to fill the current column.

  Each of these keeps saves the current formatting environment, including any partially processed output line. The formatting environment is restored when the keep ends. See Figure 32 on page 315 for a list of formatting parameters saved and restored around keeps.

  Some control words are not allowed within keeps, and will force termination of the keep before being processed. This is true regardless of whether the control word is found in the input file, in a tag, or within a macro. In general, these control words alter the page or column definitions; they are listed in Figure 27 on page 312.

There is an order of precedence among keeps, with regular, floating, and delayed keeps taking precedence over inline keeps. If an inline keep is encountered within a floating keep, it is ignored. But if a regular keep is encountered within an inline keep, the inline keep is ended and the regular keep begun. Keeps of the same level of precedence end each other. For example,

```
.kp on
These lines will be
kept together in
one column.
.kp on
So will these lines,
but not necessarily in
the same column with the
previous few lines.
.kp off
```

Examples

In this document, each paragraph is started with the GML starter set ":p" tag. One of the control words invoked by this tag is:

```
.kp 3
```

This ensures that the first three lines of the paragraph will be kept together, preventing widows at the bottom of a column.

If you place a large figure in a regular keep, and it does not fit in the current column, it will be placed at the top of the next column. This will 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 may appear above or
below the figure, depending upon whether
the figure is moved to the next column.
```

Inline keeps are preferable to conditional column ejects, especially when your page layout contains more than one column, because columns which are explicitly ended with .CB [Column Begin] or .CC [Conditional Column Begin] are ineligible for balancing. Inline keeps ensure that text is moved to the next column if necessary to keep the text together, yet allow preceding text to be moved into the next column as needed to balance the columns if the page is not filled. See "Chapter 5. Multicolumn Page Layout" on page 65 for more information on column balancing.

FOOTNOTES

The .FN [Footnote] control word provides an automatic way to format text so it appears at the bottom of a page as a footnote. SCRIPT/VS determines how many lines currently remain on the page and reserves the space needed for the footnote. The .FN [Footnote] control word is specified as:

```
.fn on
** This line is going to
appear as a footnote
on this page.
.fn off
```

SCRIPT/VS prints a 16-dash line, called a "leader," to separate the body of the page from the footnote. However, you can specify up to ten lines of leader material. Define the leader before you define the first footnote for the page. For example,

```
.fn leader
.sp
.tr - BF
.us ----------------
.sk
.fn off
```

You can mark up the footnote with GML tags, control words, macros, and text just as you can the material within a keep. The footnote definition can include up to ten output lines. The control words that are disallowed within a keep are also disallowed within a footnote. For example, to provide special formatting within a footnote:

```
.fn on
.tr 2 B2
.in 5m
2  This is the next footnote
in this section.
.fn off
```

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 before the word or phrase that refers to the footnote (called the "callout").

** This line is going to appear as a footnote on this page.
[2] This is the next footnote in this section.

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 not used outside the footnote. Therefore, it is not necessary to reset the indention. See "Chapter 8. The SCRIPT/VS Formatting Environment" on page 97 for details about saving and restoring the formatting environment.

STARTING TEXT AT THE TOP OF A PAGE OR COLUMN

When you want to start text at the top of a page, you can:

- Precede it with a .H1 [Head Level 1] control word.
- Precede it with a .PA [Page Eject] control word to force a new page.
- Precede it with a .CP [Conditional Page Eject] control word to force a new page if not enough space remains on the current page.
- Use the .DI [Delay Imbed] control word to save the input text until the next page eject occurs, then to process it. (The .DI [Delay Imbed] control word is described in "Chapter 10. Combining SCRIPT/VS Files" on page 105.)

When you need to start text at the top of a column:

- Precede it with a .CB [Column Begin] control word to force a new column.
- Precede it with a .CC [Conditional Column Begin] control word to force a new column if not enough space remains in the current column.
- Use a .KP [Keep] DELAY control word to keep a block of text together and print it in the next column.
- Use a .KP [Keep] control word to keep a block of text together and print it in the next column if it won't fit in the current column.

Many of these control words are discussed in other parts of the book. This section describes the .CP [Conditional Page Eject] and .CC [Conditional Column Begin] words.

CONDITIONAL COLUMN AND PAGE EJECTS

The .CP [Conditional Page Eject] and the .CC [Conditional Column Begin] control words allow you to specify how much space must remain in the column for SCRIPT/VS to continue formatting lines in that column. If there is not enough space remaining, SCRIPT/VS performs the page (or column) eject. For example:

```
This list includes
.sk
.cp 3
GML Tags
Symbols
Macros
```

When the .CP [Conditional Page Eject] control word is encountered, SCRIPT/VS determines the number of lines left in the column. If there are at least three lines, processing continues and the lines are printed on the current page. If there are fewer than three lines, however, SCRIPT/VS performs a page eject; the lines following the .CP control word are printed on the next page.

When you use the .CP [Conditional Page Eject] control word, SCRIPT/VS always ejects to the next page when less than the required amount of space remains in the column.

The .CC [Conditional Column Begin] control word works in a similar fashion. A column eject (which might result in a page eject if it occurs when the page's last column is processed) is performed when there are fewer than the required number of lines left in the column.

MARKING UPDATED MATERIAL

If you process documents that are frequently revised, you can identify revised text with a "change bar" (or other symbol) in the left margin.[11] You use the .RC [Revision Code] control word to identify changed material. You can establish up to nine different revision code characters, which are printed to the left of your text output.

For example, the lines

```
.rc 1 |
.rc 2 *
```

initialize two different revision codes. Within the body of your document, you can identify revised material with a pair of .RC [Revision Code] control words:

```
.rc n on
.rc n off
```

"n" specifies which revision code identifier to use. If the identifier has not been defined it is, by default, a blank, which is tantamount to no character at all. The lines:

```
.rc 1 |
.rc 2 *
  ...
.rc 1 on
These lines were revised at
one time.
.rc 1 off
  ...
.rc 2 on
These lines were
revised later.
.rc 2 off
```

result in

```
| These lines were revised at
| one time.

  ...

* These lines were revised
* later.
```

For a one-line change, you can use the ON/OFF parameter of the .RC [Revision Code] control word. For example,

```
.rc 1 on/off
This is the revised line.
```

results in

```
| This is the revised line.
```

---

[11] See "Appendix D. Formatting Considerations for the 3800 Printer" on page 337 for special considerations regarding the use of .RC within documents that are printed on the IBM 3800 Printing Subsystem.

You can mark the next output line only by specifying the .RC [Revision Code] control word with an asterisk (*) and any character. For example,

```
.rc * $
This line has new information.
```

which results in:

```
$ This line has new information.
```

The revision code is placed to the left of the column of text to which it applies. For the leftmost column, the revision code is placed in the binding area provided with the BIND option of the SCRIPT command. For other columns, it is placed in the intercolumn gutter. If the space for the revision code is insufficient, the revision code is omitted.

When you do not want a revision code character to be printed, you can respecify the character to a blank character with the .RC control word. For example,

```
.rc 1
```

Revision code 1 now prints as a blank.

DRAWING BOXES

SCRIPT/VS can draw boxes around illustrations or text, and can format charts with horizontal and vertical lines.[12] The control word that draws boxes and lines within boxes is the .BX [Box] control word. You use the .BX control word in three different ways to draw a box:

1. Define the box left and right edge, and the character positions you want to contain vertical lines. For example, you might want a box thirty spaces wide, starting in character position 1, with vertical lines at character positions 10 and 20. Specify the .BX [Box] control word as:

   ```
   .bx 1m 10m 20m 30m
   ```

   which 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. For example,

   ```
   .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
   ```

---

[12] See "Appendix D. Formatting Considerations for the 3800 Printer" on page 337 for special considerations regarding the use of the .BX [Box] control word within documents that are printed on the IBM 3800 Printing Subsystem.

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. However, SCRIPT/VS will not overlay a printed character on the output line with a vertical rule unless the output device is an IBM 3800 Printing Subsystem. For 3800 printers, the vertical rule has precedence and replaces the printed character.

You can use the .BX [Box] control word to build a three-column table. Comment lines (identified with ".*") are included to explain the control words used:

```
.* Define the box:
.bx 1m 10m 20m 55m
.*
.* Set the width of the text line slightly
.* shorter than the box's right edge:
.ir 2m
.*
.* Set the indention for the third column:
.in 21m
.*
.* Set tabs for the second and third
.* columns:
.ti ¬ 05
.tb 11m 21m
.*
.* Begin the table's entries.  The first
.* line of each entry has text in each
.* column.  Subsequent lines are concatenated
.* and formatted in the third character position.
.* The first line is shifted left (with
.* Undent) to the first column position.
.un 19m
.*
.* Text for the first item follows:
Item 1¬Part 1¬The first part
of item 1 is described here.
.*
.* Separate the parts of an item with
.* a blank line:
.sk
.*
.* The first line of an item's part
.* is shifted left to the second column
.* position:
.un 10m
Part 2¬The second part of item 1 is
described here.
It is a very
long description.
.*
.* Separate the items with a horizontal line:
.bx
.*
.* Begin the next item:
.un 19m
Item 2¬Part 1¬The second
and subsequent items are entered in a
similar fashion . . .
.*
.* Complete the box:
.bx off
.*
.* Reset the page layout parameters so
.* subsequent markup is not disrupted:
.tb
.in
.cl
```

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 very long description.
Item 2	Part 1	The second and subsequent items are entered in a similar fashion. ...

Note: The character positions defined with the .BX control word are the positions at which the vertical lines are drawn. Contrast this with the displacement setting of the .TB [Tab Setting] control word (.TB 12m results in spaces through character position 12; the text begins in character position 13). Therefore, you can use the same numbers for the .BX control word and for the .TB control word, and use the tab to position to the character position immediately after the vertical bar.

The special corners and intersections that SCRIPT/VS uses to format boxes are based on the character set available with the logical output device. For example, the input lines:

```
.bx 1m 5m 25m 29m
.cl 30m
.ce on
These lines
are centered within
this
lovely box.
.ce off
.bx off
.cl
```

are displayed for the TERM logical output device as:

```
+---+-------------------+---+
|   |    These lines    |   |
|   |are centered within|   |
|   |       this        |   |
|   |    lovely box.    |   |
+---+-------------------+---+
```

However, when the same input lines are formatted for the IBM 3800 logical output device used to produce this manual, they appear as:

```
┌───┬───────────────────┬───┐
│   │    These lines    │   │
│   │are centered within│   │
│   │       this        │   │
│   │    lovely box.    │   │
└───┴───────────────────┴───┘
```

SCRIPT/VS chooses the appropriate box character set for the logical output device. However, you can force SCRIPT/VS to use any of the box character sets (see "Using Fonts With the IBM 3800 Printing Subsystem" later in this chapter for an example). The ability to force SCRIPT/VS to use a specific box character set is important, because some box character sets, such as 3270 text and APL, are never automatically selected.

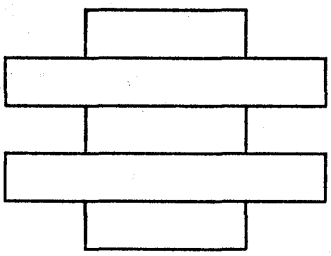
You can use SCRIPT/VS to produce many different box configurations, horizontal lines, and graphic structures. Some of the ways you can use the .BX [Box] control word are described below.

Stacking one box on another

You can define a box and then define a larger or smaller box, without first ending the first box's definition. The top of the second box is printed on the same line as the bottom of the first box. For example, the lines:

```
.bx 10m 20m
.sp
.bx 5m 25m
.sp
.bx 10m 20m
.sp
.bx 5m 25m
.sp
.bx 10m 20m
.sp
.bx off
```

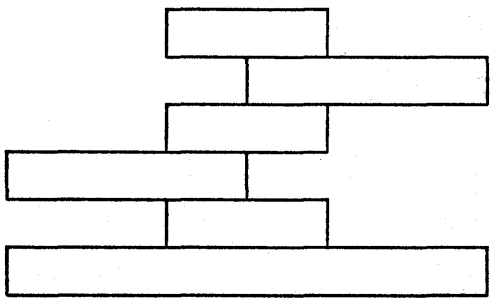
result in:



By using this form of the .BX [Box] control word, you can create a complex structure of boxes. For example, the lines

```
.bx 10m 20m
.sp
.bx 15m 30m
.sp
.bx 10m 20m
.sp
.bx 1m 15m
.sp
.bx 10m 20m
.sp
.bx 1m 30m
.sp
.bx off
.pa
```

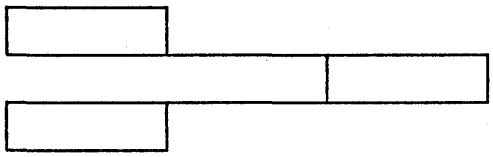
result in



When the upper box bottom line does not touch the lower box top line, SCRIPT/VS joins the two lines together. For example, the input lines:

```
.bx 10m 20m
.sp
.bx 30m 40m
.sp
.bx 10m 20m
.sp
.bx off
```

result in



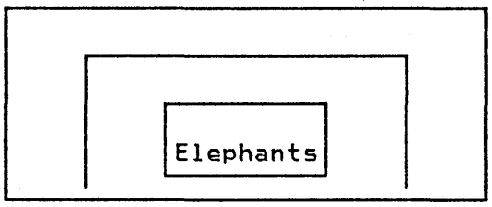
Drawing a box within a box

You can draw a box within a box, using the NEW parameter of the .BX [Box] control word.

Each box is ended with a .BX CAN or .BX OFF control word. Note the different results of each type of ending. For example,

```
.bx 1m 30m
.sp
.bx new 5m 25m
.sp
.bx new 10m 20m
.sp
.in 11m
Elephants
.bx off
.bx can
.bx off
```

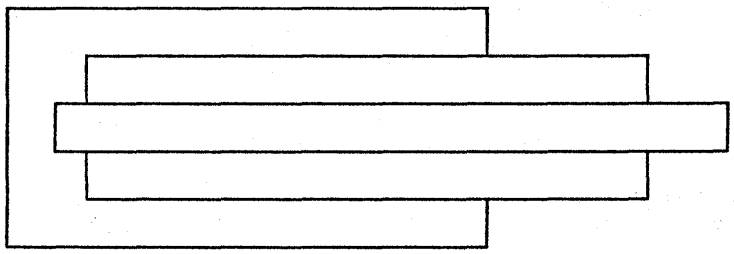
results in



When nesting boxes, the new box does not have to be completely within the previous box. For example,

```
.bx 1m 30m
.sp
.bx new 5m 40m
.sp
.bx new 3m 45m
.sp
.bx off
.sp
.bx off
.sp
.bx off
```

results in

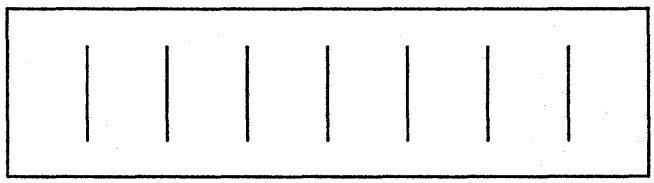


Drawing the middle portion of a box within another (larger) box

You can draw a box that is open at the top and bottom by using slashes (/) between the character position displacements (as shown previously). You can also nest that type of box within a larger box. For example

```
.bx 1m 40m
.sp
.bx new 5m / 10m / 15m / 20m / 25m / 30m / 35m
.sp 2
.bx off
.sp
.bx off
```

results in

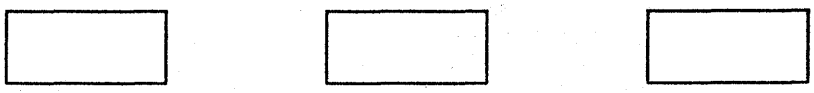


Drawing boxes in a horizontal row

You can draw a row of boxes by specifying a box definition with slashes. For example,

```
.bx 1m 10m / 20m 30m / 40m 50m
.sp 2
.bx off
```

The slash indicates a discontinuity with no horizontal connection. These lines result in:

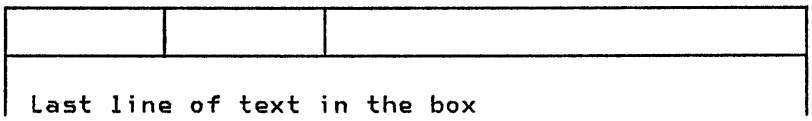


Drawing the top line (only) of a box

When you want SCRIPT/VS to draw the top portion of a box, but not the bottom line, you use the CAN parameter of the .BX [Box] control word to cancel the box definition. For example,

```
.bx 1m 10m 20m 50m
.sp
.bx new 1m 50m
.sp
Last line of text in the box
.bx can
.bx can
```

results in



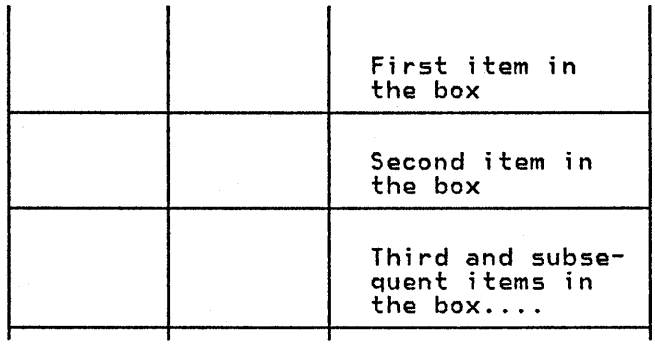
Drawing the middle portion of a box (without top or bottom lines)

When you want SCRIPT/VS to draw a box without horizontal top and bottom lines, you insert slashes (/) between the character position displacements in the .BX [Box] control word. This causes SCRIPT/VS to draw the "top and bottom lines" of the box with only the vertical bars (that is, without a horizontal line).

When you want a horizontal line across the box, redefine the box using the NEW parameter of the .BX [Box] control word. For example

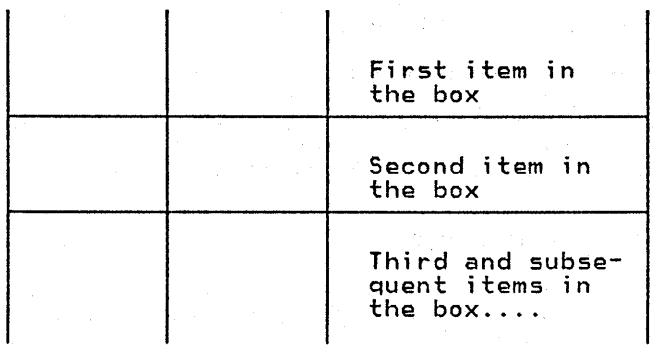
```
.in 22m
.fo left
.cl 38m
.bx 1m / 10m / 20m / 40m
.sp
First item in the box
.bx new 1m 10m 20m 40m
.sp
Second item in the box
.bx
.sp
Third and subsequent items
in the box
.bx off
.bx can
```

results in



Note: the .BX OFF control word ends the box defined with the .BX NEW control word, and results in a box bottom line. The .BX CAN control word cancels the next previous .BX. control word, to end the "open-topped" box.

If the .BX CAN control word preceded the .BX OFF control word (instead of the way it is shown in the example), the result would be:



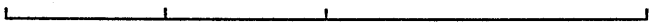
Therefore, the .BX CAN control word can be used repeatedly to cancel all previous box definitions.

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



USING FONTS WITH THE IBM 3800 PRINTING SUBSYSTEM

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

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

SCRIPT/VS provides sixteen full uppercase and lowercase fonts, which are listed and illustrated in "Appendix C. Fonts Supplied with SCRIPT/VS" on page 329. In addition, any locally-created font can be used by SCRIPT/VS when its characteristics are listed in a font table. (See "Appendix B. Device and Font Table Maintenance" on page 323 for details about adding a new font's characteristics.)

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

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

```
This is a
.bf GB12
bold
.bf GT12
word.
```

would produce the line:

This is a **bold** word.

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

```
This is a
.bf &$CHAR(2)
bold
.bf &$CHAR(1)
word printed in the &$CHAR(2) font.
```

which prints as:

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

In a longer input file, you may not want to keep track of the current font for restoration after using the .BF [Begin Font] control word. Use the .SF [Save Font] and .PF [Previous Font] control words to save and restore font names. For example,

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

Note: The font-save stack is 16 entries deep.

When you use the .BF, .PF [Previous Font], and .SF [Save Font] control words when formatting for a printer that doesn't dynamically change fonts, the control words are ignored.

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

Special considerations apply to font changes within a box. Because SCRIPT/VS does not provide three widths of each box character in each font, SCRIPT/VS performs monospace justification inside a box. The following restrictions apply within a box:

- All nested boxes are in the font of the outermost box, regardless of the font changes within the box.
- All fonts used within the box must be of the same pitch as the box itself (that is, the pitch of the current font when the outermost box was begun).
- Proportional fonts (for example, GP12) cannot be used within a box.

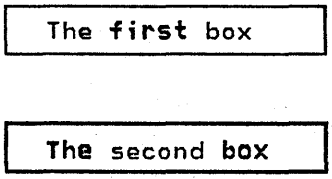
You can produce boxes of different line thicknesses containing text in several fonts. For example,

---

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

```
.fo center
.bf GT12
.bx 1m 20m
The
.sf
.bf GB12
first
.pf
box
.bx off
.sp 2
.bf GB12
.bx 1m 20m
The
.sf
.bf GT12
second
.pf
box
.bx off
```

results in:



The formatting environment is a set of values and parameters that specify exactly how SCRIPT/VS is to format each line on an output page. The formatting environment consists of three parts:

- The active environment, which contains parameters for formatting text
- The page control area, which contains parameters that define the entire page
- The translate tables associated with the .TI [Translate Input] and .TR [Translate Character] control words

PARAMETERS THAT DEFINE THE FORMATTING ENVIRONMENT

The parameters that make up the active environment area and the page control area are listed in Figure 32 on page 315. Each parameter, its corresponding control word, its initialized value, and its special SCRIPT/VS symbol (if any) is listed.

When SCRIPT/VS ejects to a new page (or begins the first page), it prepares the output page in the following manner:

1. It saves the active environment values for body text and initializes the active environment for formatting:
   - Top titles
   - Running heading
   - Running footing
   - Bottom titles

   The active environment is reinitialized before each of these is formatted.

   The output page's running titles, heading, and footing are now in place on the output page. All "page control" dimensions are now fixed for the page; any changes to these values will take effect on the next page.
2. SCRIPT/VS restores the active environment for body text that it had saved.

   Input lines are processed to produce output lines, which are inserted into the body of the page. When the page is full, or when a page eject occurs, the formatted page is sent to its destination.

THE KEEP ENVIRONMENT

When a keep with certain parameters (ON, FLOAT, and DELAY) is started, SCRIPT/VS saves a copy of the active environment and then modifies the active environment:

- The values of the .OF [Offset] and .UN [Undent] control words are cleared, and indention is restored to the basic .IN [Indent] value currently in effect.
- Maximum allowed column width is set to column width. The width of the material within a keep cannot exceed the column width, although it can be narrower.

When the keep ends, the saved copy of the active environment is restored.

THE FOOTNOTE ENVIRONMENT

When a footnote is started, SCRIPT/VS saves a copy of the active environment and then modifies it:

- The values of the .OF [Offset] and .UN [Undent] control words are cleared, and indention is restored to the basic .IN [Indent] value currently in effect.
- Column width is set to line length. The footnote text goes across the page in single-column format.

When the footnote ends, the saved copy of the active environment is restored.

SAVING AND RESTORING THE CURRENT FORMATTING ENVIRONMENT

The .SA [Save Status] and .RE [Restore Status] control words are used to save and restore the current formatting environment. All three parts of the environment are saved and restored by .SA and .RE:

- The active environment
- The page control area
- The .TI and .TR translate tables

For example, part of an input file that contains a distribution list requires indention and tab settings to format properly. However, the main document indention and tab settings are different. To avoid having to reset the main document's values, use the .SA [Save Status] and .RE [Restore Status] control words:

```
.sa
.in
Distribution list for special publications:
.sk
.in 3m
.tb 20m 30m
.us Name Dept Address
 .
 .
 .
*** End of distribution list ***
.re
```

CHAPTER 9. CONDITIONAL PROCESSING

SCRIPT/VS provides several methods for processing input conditionally. You can write input files and macros that are capable of making simple decisions, and taking action based on the result. With conditional processing techniques, you can:

- Select the alternative input lines to be processed in a particular run.
- Construct loops that process the same material several times to provide several copies of the formatted output. (Each copy can, of course, contain different specific information, as in the form letter example in "Chapter 10. Combining SCRIPT/VS Files" on page 105.)
- Construct loops that process input in several small steps.
- Write macros that define other macros based on the current logical device and the formatting required.
- Provide processing based on the content of an input line.

These capabilities use the basic conditional processing techniques in conjunction with other techniques that are not discussed here. "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137 contains information about the mechanics of writing macros, and "Chapter 11. Symbols in Your Document" on page 117 discusses symbol substitution. Individual control words are described in "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199 .

There are three basic conditional processing techniques you can use:

- The .IF control word
- Conditional sections
- Conditional processing with symbols

THE .IF CONTROL WORD

SCRIPT/VS allows you to test a symbol's value to determine whether to process an input line, or to ignore it. Use the .IF [If] control word to specify a conditional statement in the form:

```
.if argument1 test-condition argument2 input-line
```

The input-line part of the .IF [If] control word can be any valid SCRIPT/VS input line: a GML tag, a control word, a symbol, a macro, or text. The first nonblank character after "argument2" 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.

When you want to bypass a part of your input file, you can use the .GO [Goto] and ... [Set Label] control words. For example

```
.if &type = 1 .go bypass
   .
   .
   .
...bypass
```

In the above example, if the symbol "&type" has a value of 1, all the control words and text between the .IF and the ... [Set Label] control words (which sets the label "bypass") are skipped.

The conditions that 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 input line of a .IF [If] control word does not have to be a .GO control word. For example, you can use

```
.if &SYSHOUR >= 12 .im pm
.if &SYSHOUR < 12 .im am
```

In this example, a different file is imbedded (either the file named PM or the file named AM), based on whether the time is before or after 12 noon.

Conditional processing with the .IF [If] control word can be especially convenient when one file is imbedded in several different masterfiles. You can provide for slight differences among the files by setting the same symbol to a different value in each masterfile, and using that symbol to determine how processing is done in the imbedded file.

There are two special arguments that you can use on an .IF [If] control word line. They are keywords, not symbols; do not delimit them with an &.

- The keyword SYSPAGE tests whether the page currently being formatted is EVEN or ODD. You can use SYSPAGE to place text on an output page, based on whether the output page is even-numbered or odd-numbered:

  ```
  .if SYSPAGE = EVEN .sx /Evenpage Top Line///
  .if SYSPAGE = ODD .sx ///Oddpage Top Line/
  ```

- The keyword SYSOUT tests whether the destination of the output is the printer (PRINT) or the terminal (TERM). This keyword is provided for compatibility with SCRIPT/370. The SCRIPT/VS system symbols &$LDEV and &$PDEV provide a better way to test which of the many logical and physical output devices possible with SCRIPT/VS is currently in use.

The input line of an .IF [If] control word can be another .IF [If] control word. This allows you to satisfy two or more tests as criteria for processing the final input line. For example, if the document is processed in the afternoon, you want to imbed the file named PMPRINT. If the document is processed in the morning or if it is sent (as output) to a terminal, the file PMPRINT is not necessary:

```
.if &SYSHOUR ge 12 .if &$LDEV ne TERM .im pmprint
```

SPECIAL TECHNIQUES FOR CONDITIONAL PROCESSING

There are two techniques you should be aware of when using the .IF [If] control word.

- More Than One Control Word On a .IF [If] Input Line

  The input line of a .IF [If] control word can be only one actual input line. However, you can set a control word separator character (other than the semicolon) and cause the .IF [If] input line to contain two or more control words.

  SCRIPT/VS detects control word separators on an input line before processing the line. For example, the input line

  ```
  .if &sval gt 32 .sp 5;.im fig7
  ```

is processed as the two lines:

```
.if &sval gt 32 .sp 5
.im fig7
```

so that only ".sp 5" is subject to conditional processing.

To overcome this, make the subject of the condition a .DC CW (Define Character) control word, followed by a .CM [Comment] control word, as shown in the following example:

```
.if &sval gt 32 .dc cw ?;.cm ?.sp 5?.im fig7?.dc cw
```

SCRIPT/VS processes this line one of two ways, based on the condition "&sval gt 32":

- Condition is not satisfied: SCRIPT/VS does not process the .DC CW. The next line is merely a comment (.CM), and nothing happens:

  ```
  .if &sval gt 32 .dc cw ?
  .cm ?.sp 5?.im fig7?.dc cw
  ```

- Condition is satisfied: SCRIPT/VS processes the conditional input line (that is, .DC CW ?). First, the control word separator is changed to "?". Next, SCRIPT/VS recognizes the following (now properly separated) control words and processes them:

  ```
  .cm
  .sk 5
  .im fig7
  .dc cw
  ```

  This sequence ends by restoring the control word separator to its default value (;).

- Comparing Null-Value Symbols

  When you specify the name of a symbol value that might be null, you should precede the symbol name with a character-prefix to avoid a possible error. For example, the input line

  ```
  .se a = ''
  .if &a = ON .go next
  ```

  results in a SCRIPT/VS error because the symbol &a was set to a null value. The conditional statement resolves to:

  ```
  .if = ON .go next
  ```

  The "=" character is treated as the first comparand, and "ON" is not a valid comparison. However, the input line

  ```
  .if x&a = xON .go next
  ```

  resolves to

  ```
  .if x = xON .go next
  ```

  When the symbol &a has the value "ON," it resolves to

  ```
  .if xON = xON .go next
  ```

  That is, the prefix "x" is concatenated with the value of &a to result in "xON," which satisfies the test. When the symbol &a hasn't been set to ON, x&a = x and the test fails, but no error results.

CONDITIONAL SECTIONS

When your document is likely to be read by several different audiences, you may want to build it so that you can customize it for each. For example, your company might build three very similar devices:

- Class A Widget, which is a very basic machine.
- Class B Widget, which is really an improved Class A Widget.
- Class C Widget, which includes some, but not all, of the features of Class A and B Widgets, and includes some improvements of its own.

Because the devices are very similar, you can write a section of material that applies to all three. You can follow each general information paragraph or section with more specific information applicable to one or two, but not all three, of the device types.

In this way, you can keep all information about Widgets in one input file. When you format the input file for printing, however, SCRIPT/VS can customize it so that all information (general as well as specific) about one of the classes of Widgets is printed. To do this, you identify those sections of the input file that are to be processed conditionally.

SCRIPT/VS processes a conditional section, or ignores it, based on the setting of a .CS [Conditional Section] control word. Each conditional section number, from 1 to 9, can be used many times in a document. You can associate each type of information to be processed conditionally with its own conditional section number.

For example,

Conditional Section Number	Conditional Section Applies To
1	Only Class A Widgets
2	Only Class B Widgets
3	Only Class C Widgets
4	Class B or Class C Widgets (Not Class A)
5	Class A or Class C Widgets (Not Class B)
6	Class A or Class B Widgets (Not Class C)

At the beginning of your document, specify that SCRIPT/VS is to bypass all conditional sections with the IGNORE parameter of the .CS [Conditional Section] control word. The SCRIPT/VS default is to process all conditional sections not specifically bypassed.

```
.cs 1 ignore
.cs 2 ignore
    .
    .
    .
.cs 6 ignore
```

Before you issue the SCRIPT command to process your document, change some of the .CS [Conditional Section] IGNORE control words to .CS [Conditional Section] INCLUDE control words, to process each desired conditional section. For example, to print all material appropriate for readers interested in Class B Widgets, specify

```
.cs 2 include
.cs 4 include
.cs 6 include
```

In the body of your input file, you identify each conditional section by preceding it and following it with the .CS [Conditional Section] control words, using the ON and OFF parameters. For example,

```
.cs 2 on
This material applies only
to Class B Widgets.
It does not apply to either
of the other types.
.cs 2 off
```

Because you can only specify one conditional section number with the .CS [Conditional Section] control word, you must use a separate number to identify sections that apply to either one of two (but not the third) type of device.

When you "nest" the .CS [Conditional Section] control words, you identify a section that applies only when two (or more) conditions are met. For example,

```
.cs 1 on
.cs 2 on
This material is applicable to
people who use the Class A Widget
with the Class B Widget.
It is not to be printed for
readers interested only in Class A
Widgets or only  in Class B Widgets.
.cs 2 off
.cs 1 off
```

Because the .CS [Conditional Section] control word doesn't cause a break, you can process small units of text conditionally. For example, the input lines

```
.cs 1 ignore
.cs 2 ignore
.cs 3 include
This book is written specifically
for the operator of a
.cs 1 on
Class A
.cs 1 off
.cs 2 on
Class B
.cs 2 off
.cs 3 on
Class C
.cs 3 off
Widget.
```

are printed as:

```
This book is written specifically for the
operator of a Class C Widget.
```

The input lines (GML tags, control words, and text) between the .CS ON and the .CS OFF control words are included unless explicitly bypassed as a result of a preceding .CS IGNORE control word.

When ignoring a conditional section n, SCRIPT/VS recognizes one control word only:

```
.cs n off
```

which ends the ignored conditional section. All other input lines (control words as well as text lines) are ignored without further processing. This means that an ignored conditional section that is started in one file cannot be ended in another file that is imbedded by the first, because the .IM [Imbed] control word will not be processed in the ignored section. However, if a conditional section is started in an imbedded file, it can be ended in the outer file, because SCRIPT/VS returns to the outer file automatically when it reaches the end of the imbedded file. No control word is needed to switch back from the end of an imbedded file to the file that imbedded it.

CONDITIONAL PROCESSING WITH SYMBOLS

With set symbols, you can do conditional processing in several ways. The simplest of these is to have a symbol that resolves into one control word or another depending on the conditions. For example, the symbol "xim" could be set to either ".CM" or ".IM" to cause the input line

```
&xim filename
```

to be treated as an .IM [Imbed] control word or a .CM [Comment] control word. If your file has several places where 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 11. Symbols in Your Document" on page 117 for details on symbol attributes. The existence attribute causes a string to be substituted with 0 if a symbol does not exist, and with 1 if it does. You could write a macro called "X0" to provide the appropriate processing when a given symbol does not exist, and another called "X1" for when it does exist. Now, the expression:

```
.X&E'&name
```

will resolve to ".X0" if the symbol "&name" does not exist, and ".X1" if it does.

You can also use the symbol length attribute (&L') to perform conditional processing. The length attribute and the following string or symbol are replaced with the length of the string or symbol during substitution. See "Chapter 11. Symbols in Your Document" on page 117 for details. If a symbol called "&num" contains a number that is from one to five digits long, you can develop a five-digit number by adding the correct number of leading zeros to &num. First, you need to define symbols that contain the number of zeros needed for each possible length the number might be:

```
.se 5z =
.se 4z = 0
.se 3z = 00
.se 2z = 000
.se 1z = 0000
```

If the number is five digits long, you need to add no zeros. If it is four digits long, you need one zero, and so forth. Now, the expression

```
&&L'&num.z.&num
```

concatenates the correct number of zeros to the number to form a five-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 two zeros, the proper number of zeros for a three-digit number, and concatenated with the number itself when "&num" is substituted.

SCRIPT/VS provides the ability to combine many SCRIPT/VS input files for processing as a single document. The control words that allow you to do this are:

- .IM [Imbed], which causes SCRIPT/VS to process another file immediately, then return to the imbedding file.
- .AP [Append], which causes SCRIPT/VS to process another file immediately and not return to the appending file.

IMBEDDING AND APPENDING FILES

An input file can "call" other input files with the .IM [Imbed] and .AP [Append] control words:

- When a file is imbedded into an outer file, the contents of that file are read and processed as though the file were inserted into the outer file immediately following the .IM [Imbed] control word. When the imbedded file completes, SCRIPT/VS resumes processing at the outer file's input line that follows the .IM [Imbed] control word.
- When a file is appended to another file, the contents of the appended file are processed immediately. The appended file replaces the appending file as the current input file. Consequently, when the appended file completes, SCRIPT/VS does not resume processing input lines in the appending file.

You must specify the filename of the file you want to imbed or append. If the SCRIPT/VS file named OUTER processes the input line

```
.im tester
```

SCRIPT/VS stops reading input lines from the OUTER file and begins reading and processing lines from a file named TESTER. Whatever formatting controls are in effect when the file is imbedded remain in effect until respecified by control words in TESTER. Imbeds do not cause breaks, either. When SCRIPT/VS reaches the end of the TESTER file, it continues processing in OUTER with the input line following the .IM [Imbed] control word.

The file TESTER might also contain .IM [Imbed] control words to imbed additional files. The process is the same as when TESTER was imbedded. The imbedded file is read and processed, then SCRIPT/VS returns to the line in the imbedding file that follows the .IM [Imbed] control word. Although many files can be imbedded in another SCRIPT/VS file, the maximum level of nesting is eight.

For example, consider the following four files:

```
MASTER:          FILEA:          FILEB:          FILEC:

.im filea        The quick       brown fox       over
.im filec        .im fileb                       the lazy
dog.             jumps
```

When you issue the SCRIPT command to format the MASTER input file, the result is:

```
The  quick brown fox
jumps over  the lazy
dog.
```

The .AP [Append] control word is similar to the .IM [Imbed] control word, except that when SCRIPT/VS finishes processing the input lines from a file specified in a .AP control word, it does not return to the calling file. For example, when SCRIPT/VS processes the input line

```
.ap names
```

it closes the current input file and begins processing the NAMES file. All the lines from the NAMES file are treated as though they were in the original file. When the end of the NAMES file is reached, SCRIPT/VS does not return to the file that appended it:

- If the file that appended NAMES was the file named in the SCRIPT command, SCRIPT/VS completes processing.
- Otherwise, if the file that appended NAMES was itself imbedded, SCRIPT/VS returns to the next input line in the file that originally imbedded the file that appended NAMES, as shown in Figure 9.

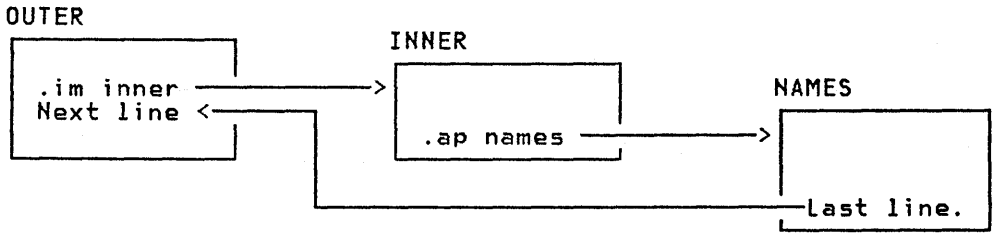


Figure 9. Imbedding and Appending SCRIPT/VS Files

You can pass values to the imbedded or appended file, so the file can be customized each time it is called. See "Chapter 11. Symbols in Your Document" on page 117 for details.

NAMING THE FILE TO BE IMBEDDED OR APPENDED

The name of the file to be imbedded or appended is given as a 1- to 8-character name with the .IM or .AP control word:

```
.im filename
.ap filename
```

"filename" is an internal SCRIPT/VS name for the file to be read. The real name of the file can be established in one of two ways:

- You can use the .DD [Define Data File-id] control word to associate the "filename" to any real file name available in the system under which SCRIPT/VS is executing, as explained in "Naming an Input File" in "Chapter 1. An Introduction to SCRIPT/VS" on page 1.
- If no .DD control word has been processed for "filename," SCRIPT/VS uses "filename" to derive the real name of the file to be read, based on rules appropriate for the system under which it is executing.
  - In CMS, "filename" is used as the filename of a CMS file whose filetype is SCRIPT.
  - In TSO, SCRIPT/VS builds the name "userid.filename.TEXT," using "filename" as the second qualifier.

When the .DD [Define Data File-id] control word defines the file-id, SCRIPT/VS makes assumptions about the file name based on the environment, as explained in "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199.

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

- the imbedded filename is different from the actual CMS filename.
- the filetype is other than SCRIPT.
- a specific filemode that is not the first in the CMS search sequence is to be used.

In TSO, use the .DD [Define Data File-id] control word when the imbedded or appended file is not a member of the partitioned data set (PDS) named in the SCRIPT command, or when the member name is different from the file-id.

In the batch processing environment, use the .DD [Define Data File-id] control word when:

- the library document name is different from the imbedded filename.
- a password is required to access the file.
- the file is stored on a library other than the ones listed with SCRIPT command options.

The format and usage of the .DD [Define Data File-id] control word when defining a file are described in "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199.

MASTER FILES

There are several advantages to using imbeds in SCRIPT/VS files:

- For convenience in updating and tracking SCRIPT/VS files, you can use one file as the master file for a SCRIPT/VS document. The master file can contain the formatting controls (for page size, depth, column definitions, and so on) that are to be in effect for the entire document. The remainder of the master file might contain only the .IM control words that imbed the remaining files.
- You can easily reorganize a large document that is composed of many small files that are imbedded in a single master file. When you want to move or remove information, you need only to change the position of the .IM [Imbed] control word in the master file, or to delete it.
- Small files can be shared by several master files. Each master file can imbed the small file where appropriate. Therefore, you do not need to keep duplicate copies of the same information.
- While there may be a limit to the number of records that can be contained in a single disk file, there is no restriction on the number of files that SCRIPT/VS can process. Also, many different people can work on pieces of the same document simultaneously.
- Some CMS and TSO editors have a limit on the number of records in a file. With the .IM [Imbed] control word, you can structure a document by combining many small files, each of which can be edited. The document as a whole can be formatted and printed using the SCRIPT command.

Figure 10 on page 108 illustrates a typical master file structure.

When you are proofreading SCRIPT/VS output files that contain many imbed files, you can use the NUMBER option of the SCRIPT command. As a result, SCRIPT/VS prints (next to each output line)

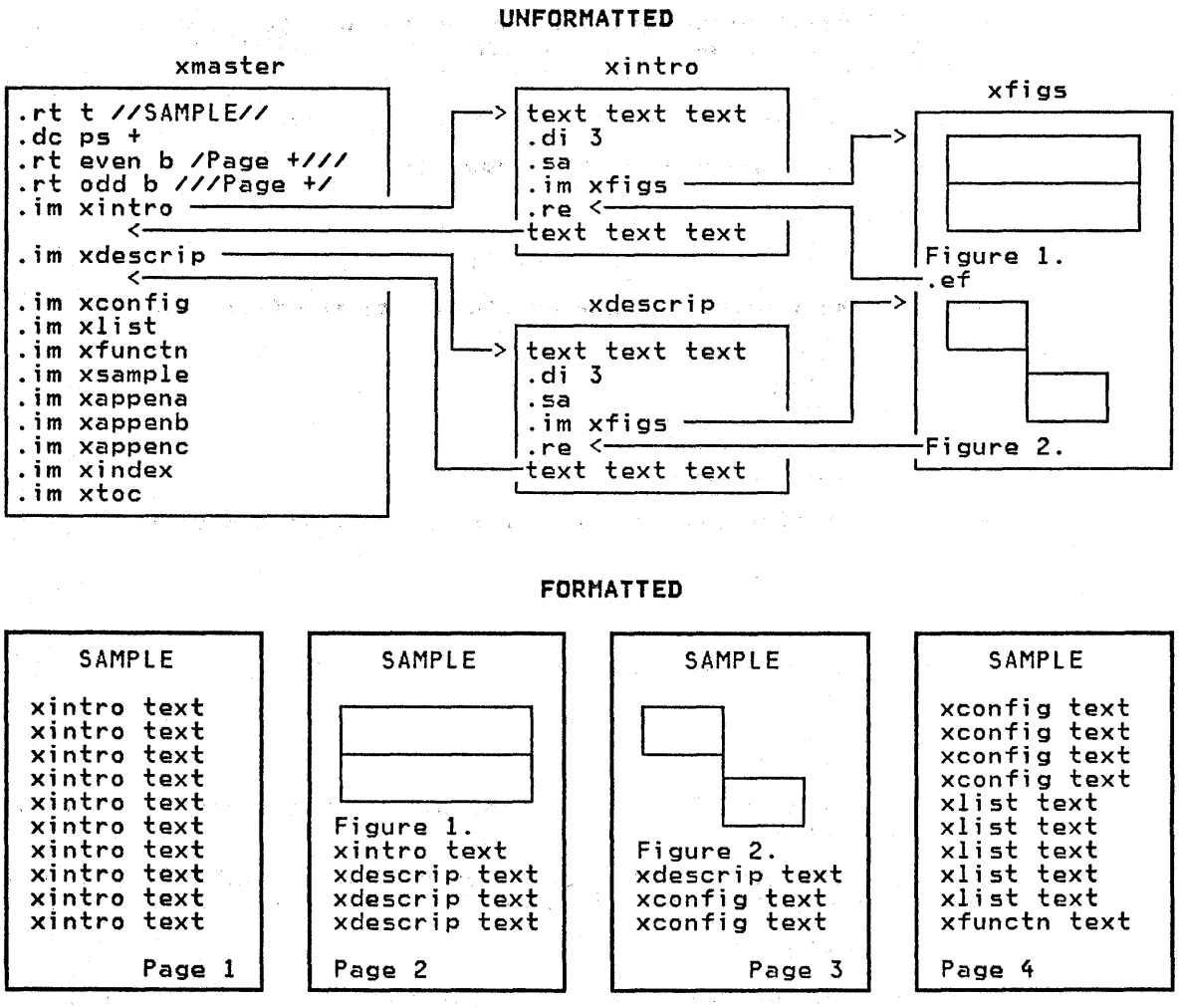


Figure 10. Master File Structure

- the filename of the file that is currently being processed.
- the number of the last input line SCRIPT/VS had read when the output line was formatted.

The line number and file name are useful when you update and correct the SCRIPT/VS files.

The UNFORMAT option of the SCRIPT command causes SCRIPT/VS to print input lines instead of output lines. SCRIPT/VS produces an unformatted document that contains all the input lines from all imbedded and appended files.

WRITING TO AN OUTPUT FILE

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

You can have several .WF files, one for each of several lists you want to build as your document is processed. However, only one .WF file can be open at a time.

When SCRIPT/VS processes the .WF [Write To File] control word, one or more input lines are written to a SCRIPT/VS file named DSMUTWTF.

- You can insert one line into the file with:

```
.wf contents of the input line
  .
  .
  .
.wf .ce Text to be centered.
```

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

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

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

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

  Note: the .WF [Write To File] OFF control word must be on an input line by itself.

You can later process the contents of the DSMUTWTF file with the IMBED parameter of the .WF [Write To File] control word:

```
.wf imbed
```

After imbedding the DSMUTWTF file, you can add to the end of it with more .WF control words. You can imbed the DSMUTWTF file into another file many times.

When the contents of DSMUTWTF are no longer useful to you, you can erase the file with the ERASE parameter of the .WF [Write To File] control word:

```
.wf erase
```

The DSMUTWTF file can be erased and reused many times.

Several .WF Files

By using the .DD [Define Data File-id] control word, you can define the file-id DSMUTWTF so that you can use the .WF [Write To File] control word to write lines to several different files. For example, to add lines to the end of the CMS file PART6 SCRIPT A1:

```
.dd dsmutwtf lib part6
.wf on
Input lines
to be added
to PART6.
.wf off
```

Note: If the file (PART6 in the above example) is currently being imbedded or appended, you cannot add lines to it. That is, you cannot write into a file that is currently being read.

To restore the file-id DSMUTWTF to the default real file, specify

```
.dd dsmutwtf lib dsmutwtf
```

DELAYING THE IMBEDDING OF INPUT TEXT

The .DI [Delay Imbed] control word is used to store input lines in a SCRIPT/VS file named DSMUTDIM. When you are finished storing lines into it with the .DI control word, SCRIPT/VS continues processing your file's input lines. As soon as a page eject occurs (either because of a full page or because of a control word that causes a page eject), SCRIPT/VS imbeds and processes the DSMUTDIM file.

When SCRIPT/VS encounters a .DI [Delay Imbed] ON control word, it puts the following input lines into the DSMUTDIM file. The lines are put in DSMUTDIM as they are; SCRIPT/VS does no other processing except to check each line for the .DI [Delay Imbed] OFF control word which must begin in column 1. When the .DI OFF control word is found, SCRIPT/VS continues to format the input lines in the file.

For example, to delay the inclusion of a few input lines until the next page eject occurs, you can specify:

```
.di on
.ce on
**********************
*  READ CAREFULLY  *
**********************
.ce off
.di off
```

The input lines from .CE ON to .CE OFF are written into the DSMUTDIM file. When one of the lines is a .IM [Imbed] control word, the DSMUTDIM file does not receive the imbedded file's contents. Instead, when the DSMUTDIM file is imbedded at the top of the next page, the file it imbeds is read in and processed.

Rather than begin an open-ended delay imbed with .DI ON, you can specify the number of input lines to be delayed. For example,

```
.di 3
```

causes the next three input lines to be delayed. You don't have to terminate the .DI 3 with .DI OFF. However, .DI OFF ends a delay imbed, whether it was started with .DI ON or with .DI n before n lines have been imbedded.

TERMINATING THE FORMATTING OF A FILE

There are three control words that cause SCRIPT/VS to terminate processing: .EF [End of File], .QU [Quit], and .QQ [Quick Quit].

The .EF [End of File] control word is useful with imbedded files. If a .EF control word occurs in an imbedded file, SCRIPT/VS does not continue imbedding the file but returns to process the outer file. If another .IM [Imbed] control word is encountered that imbeds the same file again, SCRIPT/VS resumes reading and processing with the input line following the .EF control word that was last processed.

You can respecify the start of the file again with the CLOSE parameter of the .EF [End of File] control word:

```
.ef close
```

The next time the file is imbedded, SCRIPT/VS begins reading input lines at the first line.

The other two control words, .QU [Quit] and .QQ [Quick Quit], cause SCRIPT/VS to terminate processing entirely, regardless of whether the current file is an imbed file or not. When you use the .QU [Quit] control word, processing terminates after SCRIPT/VS prints the remainder of the current page (and any bottom titles and running footings in effect) and after SCRIPT/VS closes all open files. In contrast, the .QQ [Quick Quit] control word causes immediate termination of processing with no final page eject. Therefore, some formatted text on the last page might not be printed.

The .QQ [Quick Quit] control word can be useful when checking your file for errors. You can specify the TWOPASS option when formatting the file and terminate processing after the first pass completes.

For example, a very long input file named MASTER10 can have the last input line

```
.qq
```

When you format it at the terminal using the SCRIPT command

```
SCRIPT MASTER10 (TERM TWOPASS
```

the file is completely formatted during the first formatting pass. Errors detected by SCRIPT/VS are displayed at your terminal for you to note and correct later. However, processing terminates before the second pass occurs, when the formatted document would usually be displayed.

MERGING DOCUMENTS FROM SEVERAL SOURCES

You can create a customized document from many different input files by using the .IM [Imbed] and .EF [End of File] control words. An imbedded file can include .EF [End of File] control words to cause a different group of input lines to be processed each time the file is imbedded. This can result in customized documents because each group of lines from the imbedded file can contain the specific information for a particular copy of the basic document.

CREATING A CUSTOMIZED LETTER FOR MASS MAILING

By using the .IM [Imbed] and .EF [End of File] control words, you can create a form letter customized for each of its recipients. For example, the form letter might be built using two files:

- LETTER, which contains the contents of the letter (in this example, a congratulatory notice to a prize winner).
- WINNER, which contains the information about each winner and award needed to customize the letter.

The LETTER file consists of the following input lines (explanatory comments are included):

```
LETTER:
.* The file appends itself to repeat the
.* letter until the last one has been processed.
.*
.* Date and address:
.fo off
October 30, 1978
.* Imbed 1: Winner's name and address
.im winner
.fo left
.sp 2
.*
.* Salutation:
Dear
.* Imbed 2: Winner's name
.im winner
```

```
.sp
.*
.* Letter contents:
Congratulations!
Last July was your lucky month!
You entered our very difficult and
demanding word-finding contest and
.uc you won!!!
You won the cash prize of
.* Imbed 3: Amount won
.im winner
A check for that amount is enclosed
with this letter.
Thank you for
participating in our contest.
May your good luck continue.
.sp
.*
.* Sign off:
Yours truly,
.sp 2
.fo off
X. T. Smith
Manager, Word-finding Contest
.* Controls to finish the letter
.* and start a new letter or to end.
.pa
.if x&last ne xyes .ap letter
.*
End of File
```

The WINNER file can contain many groups of input lines. This example only shows the first group (for the first winner) which is repeated for each recipient, and the last part of the last group (to end the document with the last letter).

```
WINNER:
.* Data for W. Adams
.* Imbed 1:
Ms. Winnifred Adams
1487 Easy St.
San Jalisco, Calif, 95138
.ef
.* Imbed 2:
Ms. Adams,
.ef
.* Imbed 3:
$500 (Five hundred dollars).
.ef
.*
.* Data for next recipient:
  .
  .
  .
.* Imbed 3 (for last recipient):
$5 (Five dollars).
.se last = yes
.* *** No more letters ***
End of File
```

The formatted LETTER file is shown in Figure 11 on page 113.

INTERACTIVE SCRIPT/VS PROCESSING

When you use SCRIPT/VS, you do not have to have all of your input text in final form when you issue the SCRIPT command. There are three control words that allow you to enter one or more input lines from the terminal.

- The .RD [Read Terminal] control word allows you to type text at a typewriter terminal during SCRIPT/VS output. This control word is useful if you are creating form letters and want to enter names, addresses, or other kinds of variable information directly at the terminal. The text you type is not

October 30, 1978
Ms. Winnifred Adams
1487 Easy St.
San Jalisco, Calif, 95138

Dear Ms. Adams,

Congratulations! Last July was your lucky month! You entered our very difficult and demanding word-finding contest and YOU WON!!! You won the cash prize of $500 (Five hundred dollars). A check for that amount is enclosed with this letter. Thank you for using our fine product and for participating in our contest. May your good luck continue.

Yours truly,

X. T. Smith
Manager, Word-finding Contest

Figure 11. Result of the Customized Form Letter

inserted into the input file and is not processed by SCRIPT/VS.

- The .TE [Terminal Input] control word accepts input lines of text or control words as though they were part of an imbedded input file, and processes each line as it is entered.
- The .RV [Read Variable] control word allows you to assign a value to a symbol during SCRIPT/VS processing by entering it at the terminal.

The .TE [Terminal Input] and .RV [Read Variable] control words are enhanced by using the .TY [Type on Terminal] control word to produce a prompting message, which is displayed at the terminal during SCRIPT/VS processing. The prompting message is not formatted as part of the output.

The .TE [Terminal Input] control word accepts several operands. You can specify (in the input file)

```
.te on
```

SCRIPT/VS reads input lines from the terminal until you type in

```
.te off
```

Then, SCRIPT/VS processing continues with the next line in the file. You can enter SCRIPT/VS control words or text.

You can specify a numeric parameter with the .TE control word. For example,

```
.te 4
```

In this example, SCRIPT/VS reads four lines from the terminal.

You can also terminate terminal input with the .EF control word, which indicates the end of the current file. The .TE [Terminal Input] control word is essentially an imbed, where the "file" imbedded is the terminal.

The following example uses these control words to process and format the same file an indefinite number of times.

```
...start
.im heading
.ty Enter NAME and ADDRESS (3 lines)
.te 3
.im letter
.ty Any more?
.rv answer = '
.if x&answer eq xyes .go start
```

The lines between the label ...start and the .IF control word are processed an indefinite number of times. As long as you continue to enter "yes" when prompted with the message "Any more?," SCRIPT/VS loops back to the beginning of the file, prompts you for another name and address, and continues.

Notice how the .RV [Read Variable] control word results in the setting of the symbol &answer, and allows it to be set to a character string value (as entered). When used in the .IF control word, the symbol is preceded with "x." If there is no reply to the .RV request, the symbol &answer has a null value so the comparison is between "x" and "xyes," which is unequal. Without the prefix, if the symbol value of &answer were null, the first argument would "disappear," resulting in a control word error because of an incorrect number of arguments. Once you have set a symbol in this manner, you can use the symbol "&answer" as you would any other set symbol.

To enter variable names from the terminal, you must follow the syntax rules for defining set symbols. See the descriptions of the .SE [Set Symbol] and .RV [Read Variable] control words in "Chapter 21. SCRIPT/VS Control Word Descriptions" on page 199 for more details.

COMMUNICATING WITH VM/370

Another useful feature of SCRIPT/VS is the ability to execute CMS or CP commands from CMS SUBSET during SCRIPT/VS processing. To execute a command (or an EXEC procedure or user program), use the .SY [System Command] control word. For example,

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

The .SY [System Command] control word is convenient if you ordinarily need to issue several commands before you process a SCRIPT/VS file (you may need certain disks accessed, 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.

For example, 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:

```
.sy cp link user2 191 291 rr rpass
.sy access 291 b
.im filea
.im fileb
.sy release 291 (detach
```

When you execute a command during SCRIPT/VS processing, you might not want SCRIPT/VS to continue processing if the command failed. To test the return code from the CMS or CP command, you can check the value of the SCRIPT/VS system symbol, &$RET:

```
.sy exec mysetup
.if &$ret ne 0 .qu
```

If the EXEC procedure MYSETUP completes with a nonzero return code, SCRIPT/VS terminates processing. If the return code is zero, execution continues with the next input line following the .IF control line.

Note: the CMS commands "CP" and "EXEC" are explicitly shown here for clarity. The implied CP (IMPCP) and implied EXEC (IMPEX) functions are not turned off when SCRIPT/VS executes, as they are within an EXEC file.

COMMUNICATING WITH TSO

The .SY [System Command] control word can be used to specify TSO commands and procedures to be executed after SCRIPT/VS completes processing for the input file. The commands specified with .SY are passed to TSO for execution in the order in which they are encountered.

For example, the .SY [System Command] control word might be used to display the output file after it has been formatted. To request the document be sent to an output file:

```
SCRIPT INFILE (FILE('OUTFILE')
```

Within the file, to display the output file:

```
.* TSO command to display the output file
.sy edit 'outfile' old
```

By using symbols, you can refer to page numbers, variable values, character strings, and control words in your input file. A symbol, in SCRIPT/VS, has a name and a value. When SCRIPT/VS encounters a symbol name, it replaces it with the symbol's current value. After all symbol names in an input line have been replaced with their current values, SCRIPT/VS processes the line.

You define a symbol by using the .SE [Set Symbol] control word. For example, to define the symbol &printer, you can issue

```
.se printer = 'IBM 1403 Printer'
```

Later, you can refer to the symbol "printer" in an input line as "&printer". Each SCRIPT/VS symbol is identified with its prefix, an ampersand (&). The symbol is terminated with either a period (.) or a blank. For example, the input line

```
Our publisher uses the &printer for output.
```

is processed by SCRIPT/VS and printed as:

```
Our publisher uses the IBM 1403 Printer for output.
```

but,

```
Our publisher uses the &printer..
```

is processed as:

```
Our publisher uses the IBM 1403 Printer.
```

Your document might contain the symbol "&printer" many times, in different places. In the future, when you want the document to describe a different printing device, you can reset the symbol with

```
.se printer = '3800 Printing Subsystem'
```

At that time, SCRIPT/VS will process your document and substitute the new value for the same symbol:

```
Our publisher uses the 3800 Printing Subsystem for output.
```

The symbol's name can be up to ten characters long. The name cannot contain blanks, but can include alphabetic characters, numerals, and the characters @, #, and $.

The symbol's value can be a character string, as shown above, or a numeric value. It can be an arithmetic expression. It can contain compound data items with imbedded blanks and control words. A symbol's value can even be another symbol. If the symbol's value contains blanks or special characters, enclose the entire value in single quotes (as shown in the example above).

Some examples of valid symbol definitions are:

```
.se corp = 'Scriptographicology, Inc.'
.se add = 1
.se incr = &add + 1
.se mult = &add * 10
.se test = testa
.se TEST = testb
```

You can set a symbol to:

- A numeric value:

  ```
  .se number = 25
  ```

- A character string:

```
.se text1 = 'IBM 1403 Printer'
```

- A SCRIPT/VS control word:

```
.se break = '.br'
```

- The value of another symbol:

```
.se printer = '&text1'
```

You can do arithmetic operations on a symbol's numeric value,

- To increment it:

```
.se nextpage = &pagenum + 1
```

- To decrement it:

```
.se prevpage = &pagenum - 1
```

- To divide it:

```
.se leaf = &pagenum / 2
```

- To multiply it:

```
.se cost = &pagenum * 20
```

- To negate a value:

```
.se negvalue = -&value
```

Symbols can also be set using the .RV [Read Variable] control word. The .RV control word allows you to enter symbol values from a terminal during SCRIPT/VS processing in interactive environments. For details, see the .RV [Read Variable] control word description.

Symbols can be set to a part of the value of another symbol by using the SUBSTR (substring) parameter of the .SE [Set Symbol] control word. The substring is one or more characters of the character string (the symbol's value). For example,

```
.se corp = 'Scriptographicology, Inc.'
.se name = substr &corp 1 6
```

sets the symbol &name to the substring of the value of the symbol "&corp" beginning with character 1 and continuing for 6 characters. Because "&corp" has been previously set to "Scriptographicology, Inc.", this substring results in the symbol &name having the value of the 6-character substring "Script".

In the same manner, the SUBSTR (substring) function can be used to extract characters from a character string that is not another symbol's value. For example,

```
.se name = substr Jonathan 5 4
```

sets the symbol &name to that 4-character substring of "Jonathan" beginning with character 5. That is, 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 allowed within a symbol value (= / + *), it must be enclosed in single quotes.

You can use the INDEX function of the .SE [Set Symbol] control word to find the location of a string of characters within a symbol value or a string of characters. For example,

```
.se name = 'Nicola'
.se location = index &name cola
```

defines the symbol &location to have the value 3, because the string 'cola' starts with the third character of the value of &name (Nicola).

HOW SCRIPT/VS SUBSTITUTES VALUES FOR SYMBOL NAMES

When SCRIPT/VS processes an input line, it first scans for any symbols in the line that require substitution. SCRIPT/VS checks any character string that begins with an ampersand (&) to see if it is a symbol name. When SCRIPT/VS finds a valid symbol, it replaces the symbol's name with its value. A symbol name is terminated either with a blank, a period (.), or the end of the input line. If the symbol name is terminated with a blank, the blank is treated as a normal input character and is left in the input line. If the symbol name is terminated with a period, the symbol value, after substitution, is concatenated with the next input character and the period is removed. Therefore, if a symbol has punctuation immediately after it, you must concatenate the punctuation character to the symbol with a symbol-end period. For example,

```
This list ends with an &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 planned
for shipment on &shipmo &shipday., 19&shipyr..
```

In this example, values for the symbols are substituted with the adjacent text and punctuation with no intervening blanks. The printed sentence appears as:

```
The name of our product is Whizbanger, which is planned
for shipment on June 19th, 1978.
```

If you do not place a concatenating period between &prodname and its punctuation (,), SCRIPT/VS regards "&prodname," simply as a character string, and performs no substitution.

You can redefine a symbol as often as necessary in your input file. Each time you redefine the symbol with the .SE [Set Symbol] control word, the new value replaces the old value.

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 127.

An example of a numeric symbol whose value is concatenated with a character string is in "Numbering Figures" on page 131.

UNRESOLVED SYMBOLS

Sometimes SCRIPT/VS encounters a symbol name that was not previously 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 are resolved during the second pass.

GML TAGS

SCRIPT/VS GML tags are recognized by the symbol processor. A GML tag is actually a symbol set by the .SE [Set Symbol] control word, and is associated with an application processing function (APF) during symbol substitution. Typically, the APF for a tag is a SCRIPT/VS macro. The tag symbol value substituted for the tag name invokes that macro.

SCRIPT/VS recognizes an alternate symbol delimiter for GML tags. The GML delimiter is initially, and by default, the colon (:). You can change it with the .DC [Define Character] GML control word.

Symbols delimited with the GML delimiter are substituted differently than those delimited with an ampersand. To be substituted, a GML-delimited symbol must be:

- Defined with .SE [Set Symbol] with a name all in uppercase characters.
- A simple symbol, not a compound or array symbol.

If no symbol that meets these requirements exists, the GML delimiter is left in the line as a text character. The search for symbol (and GML) delimiters continues from the next character of the input line.

To illustrate the difference between symbol substitution for symbols delimited with the ampersand and those delimited with the GML delimiter, suppose the following symbols have been set:

```
.se charts = 'the symbol'
.se CHARTS = 'the tag'
.se Charts = 'charts'
```

These are three different symbols. The names differ only in the case of the characters used. In symbol substitution:

```
&charts yields: the symbol
&CHARTS yields: the tag
&Charts yields: charts
&&Charts yields: the symbol

:charts yields: the tag
:CHARTS yields: the tag
:Charts yields: the tag
:cHaRtS yields: the tag
::Charts yields: :the tag
```

Notice that every use of the GML delimiter results in substituting the same symbol, &CHARTS. The double ampersand causes two stages of substitution, but the double GML delimiter does not.

Any GML tag can actually be specified with an ampersand and the uppercase form of the symbol name that represents the tag. Conversely, the GML delimiter can be used to refer to any symbol whose name is uppercase. However, you should always use the GML delimiter for markup tags, and the ampersand for symbolic variables to preserve the distinction between the markup and the content of your document.

INHIBITING SUBSTITUTION

Usually, ampersands and GML delimiters that occur in an input file as ordinary text characters are treated as text characters and not as symbol delimiters. The context in which it appears usually prevents the text ampersand from being mistaken for a symbol name. Where a text ampersand precedes a character string that forms a defined symbol name that you want treated as a text character string, there are several ways to inhibit symbol substitution:

- Turn off substitution with the .SU [Substitute Symbol] control word. With the .SU OFF control word, all substitution is turned off. You can turn symbol substitution on again with .SU ON.

- Contrive to make the symbol name unrecognizable by adding punctuation without a delimiting period. For example,

  ```
  I have defined the symbols &AAA, &BBB, &CCC, and others
  for this file.

  The symbol for the day of the month (&SYSDAYOFM) is
  maintained by SCRIPT/VS.

  Use the symbol "&xyz" for this purpose.
  ```

- Translate an unused punctuation mark or special character on your keyboard to the ampersand, and enter the special character in your input whenever you need a text ampersand:

  ```
  .tr ¢ &
  ```

  Because the translation happens after symbol substitution, the text ampersand cannot be mistaken for a symbol-starting ampersand.

- Define a symbol to have the value of of an unused hexadecimal code and translate the code to an ampersand. Enter the symbol name in your input whenever you need a text ampersand. The .TI [Translate Input] control word is used to establish the symbol's value: an unused hexadecimal character code (hexadecimal 07, for example) replaces the symbol &amp in subsequent input lines. The .TR [Translate Character] control word translates the hexadecimal 07 to an ampersand when the input has been formatted for an output line, after all symbol substitution has occurred.

  ```
  .ti 7B 07
  .se amp = '#'
  .ti 7B 7B
  .tr 07 &
  ```

  SCRIPT/VS will substitute the value "&" for the symbol "&amp" whenever it is encountered. Remember to end the symbol "&amp" with a period whenever you want to concatenate the ampersand with characters that follow it. This technique has been used in marking up this book whenever a text ampersand is required.

There are many times when text ampersands are perfectly safe and there is no need to worry about an unexpected substitution. Any time the character string immediately following the ampersand is not a symbol name, no substitution occurs. A character string cannot be a symbol name if:

- It has not been defined as such with a previous .SE [Set Symbol] control word.
- It contains a character that would not be allowed in a symbol name (before the first blank or period that ends a symbol name).
- It contains more than ten characters before the blank or period.

You can use these same techniques to protect text GML delimiters.

CANCELLING A SYMBOL

When you no longer want to use one of the symbols you've previously defined, you can cancel the symbol:

```
.se oldsymbol off
```

The symbol &oldsymbol will be regarded by SCRIPT/VS as an undefined symbol. It is as though it had never been defined; it is not regarded as a null-value symbol. When you specify

```
.se oldsymbol =
   or
.se oldsymbol = ''
```

you redefine the symbol with a "null" value. It exists as a symbol but it has as its value the null string. Note that a null symbol is quite different from an undefined symbol. The null symbol is substituted with a value: the zero-length null string.

ATTRIBUTES OF A SYMBOL'S VALUE

SCRIPT/VS provides you with the ability to determine some of the characteristics of a symbol in your input file, such as:

- Its existence (&E')
- Its length (&L')
- Its type (&T')
- Its current value (&V')

In addition, you can convert

- a numeric symbol value to its Roman numeral (character string) equivalent (&R' or &r')
- 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 lowercase character string to uppercase (&U')

&E' verifies the existence of a symbol. When you use the &E' prefix, the value is substituted with either a 1 or a 0, depending on whether or not the character string following &E' is a defined symbol. For example,

```
.se test = on
The result is &E'&test..
```

results in:

```
The result is 1.
```

If the symbol named &test had not been set, the value of &E'&test would be 0. Any character string that is not a defined symbol name, as in

```
&E'czechoslovakia
```

results in 0.

&L' determines the length of a symbol's value (or the length of any character string, for that matter). For example, after the lines:

```
.se test = 'This is a test.'
.se length = &L'&test
```

the value of &length is 15. If the symbol named &test had not been set, then &length would have a value of 5 (that is, the length of the character string "&test").

&T' analyzes the type of the symbol and replaces the character string with:

- N, if the value can be converted to a numeric value that can be used in an arithmetic expression, or
- C, if the value contains non-numeric 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".

&V' returns the current value of the symbol (as it was last set), without any further substitution. The value attribute of an undefined symbol or a character string is a null value. For example,

```
.se a = '&b.linda'
.se b = 'Be'
```

An occurrence of &a will be substituted with "Belinda" and its length is 7 (that is, &L'&a = 7). However, an occurrence of &V'&a will be substituted with "&b.linda", and &L'&V'&a = 8.

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 symbol has a value; a character string does not have a value (that is, a character string's "value" is null). For example,

```
.se b off
.se a = '&b.linda'
```

- &V'&a yields the character string "&b.linda". The ampersand and the period in "&b." are merely text characters, not symbol delimiters, for this value substitution.
- &V'&b.linda yields the character string "linda". In this case, the ampersand and the period in "&b." act as symbol delimiters. The value of the symbol "&b" is concatenated to the character string "linda". Since "&b" is not a defined symbol, its "value" is null.
- &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 "&b.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 "&b.linda", but in this case substitution stops at this point so that the intermediate result can be traced. After tracing, the string "&V'&b.linda" is evaluated as a separate operation. The ampersand and the period in "&b." now act as symbol delimiters, causing the value of the symbol "&b", which is null, to be concatenated to the string "linda".

Attributes apply only to the symbol (or character string) immediately following them, up to the next delimiter (period or blank). For example,

```
.se a = '&J'
.se b = 'K'
.se JK = 'TIMOTHY'
```

The string &a.&b resolves to "TIMOTHY", since &a.&b resolves to "&J&b", then to "&JK", and finally to "TIMOTHY". However, the string &L'&a.&b results in "2K", because "&L'&a" is evaluated first. SCRIPT/VS provides a length of 2 (for the symbol's value: "&J"), and concatenates the "2" with the character "K". &L'&a&b results in "3", because "&b" is evaluated first and the length SCRIPT/VS provides is the length of the character string "&aK" (because a symbol with that name hasn't been defined in the example).

The symbol attribute names &E', &L', &T', and &V' can be specified, to produce the same result, in either uppercase or lowercase. That is, &L' and &l' will both return the length of a symbol.

However, the symbol attributes &R', which converts a numeric value to Roman numerals, and &A', which converts a numeric value to an alphabetic character string, have different meanings when specified in uppercase and lowercase.

&R' converts a decimal number to a Roman numeral. The decimal integer number is converted to a character string that represents the number's Roman numeral equivalent:

- &R'87 will cause the string LXXXVII to be substituted.
- &R'19&SYSYEAR will cause the string MCMLXXVIII to be substituted (in 1978).
- &r'87 will cause the string lxxxvii to be substituted.

The largest number correctly translated to a Roman numeral is 3999. For numbers between 4000 and 9999, the character "?" is used to represent the number "5000" or "10,000" (for example, &R'6020 = ?MXX and &R'9020 = M?XX). Numbers larger than 9999 are not translated to Roman numerals (zero is returned).

&A' converts a number to a character string. The number is converted to character string that might be thought of as a base-26 number composed of alphabetic letters.

- &A'2 will cause the string B to be substituted.
- &A'26 will cause the string Z to be substituted.
- &A'27 will cause the string AA to be substituted.
- &A'28 will cause the string AB to be substituted.
- &A'705 will cause the string AAC to be substituted.
- &a'28 will cause the string ab to be substituted.

The largest number that can be converted is 65535. Numbers higher than this return a zero.

For both &R' and &A', if the character string to be converted is not a decimal integer number, the result is zero (for example, &R'zorch = 0).

&U' converts lowercase characters to uppercase. For example,

```
&U'hello
```

results in:

```
HELLO
```

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 also reside in the same library, but they occupy as many lines as required. Each line in a library member must have a delimiter to indicate where the line ends.

For example, a library member named "B" might look like:

```
;.BR;J
```

SCRIPT/VS regards the rightmost nonblank character from a symbol library as a delimiter and deletes it. In this case, the letter J serves as the delimiter. You might want a symbol value to have trailing blanks, which must be delimited. The .SE [Set Symbol] control word allows a symbol value to be delimited with single quotes. In a library, the delimiter is the nonblank ending character.

If the member "B" is in the library named "SCRIPT", you can issue the SCRIPT command to process a file that uses the symbol &B:

```
SCRIPT TEST (LIB (SCRIPT
```

If you do not use the LIB option, symbol &B is undefined (in this example).

The default filename for the symbol library is "GML" (because the macro definitions for GML processing functions usually reside in the library). If the symbol library has this name, you can omit the filename that identifies the library but you must specify "LIB" so SCRIPT/VS knows it has to search a library.

Before searching a library for a symbol, SCRIPT/VS translates the symbol name to uppercase characters. Even though SCRIPT/VS recognizes the symbols "&libsym" and "&LIBSYM" as separate and unique symbols, the library doesn't. Member names in the library are always in uppercase. Therefore, the symbol names "libsym" and "LIBSYM" even though they are different, can be set from the same library member. You can use the symbol 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 symbol table.

- To set an unresolved symbol. During substitution, the library can be searched for a definition of an unknown symbol with the same name as the symbol (when converted to uppercase) only when .LY ON or .LY SYM is specified. If found in the library, the symbol is defined in the symbol table.

When you are sure that none of your symbols are defined in a symbol library, you can issue the .LY [Library] control word to prevent library searches for unresolved symbols. (The initial setting is OFF. You have to specify .LY ON or .LY SYM to search a library for undefined symbols.)

The .LY OFF control word prevents all library searches, for unresolved macros as well as for symbols. The .LY MAC control word allows library searches for unresolved macros.

Note: When you specify that a symbol's definition is in the symbol library with

```
.se libsym lib
```

the current .LY [Library] control word specification is ignored. In the above example, the library is searched to find a definition for &libsym. Remember, the symbol name is translated to uppercase before searching the library.

SCRIPT/VS SYSTEM SYMBOLS

There are several groups of system symbol names that are initialized and recognized by SCRIPT/VS:

- Symbols you can use to obtain the current date and time.
- Symbols you can use to obtain current values of SCRIPT/VS formatting parameters: the current line length, left margin indention, and page length, to name a few.
- The symbol set as a return code from the latest CMS command executed using the .SY [System Command] control word.

Some of the system symbols begin with "&$". These symbols cannot be changed with a .SE [Set Symbol] control word, because they are reserved and contain SCRIPT/VS formatting parameters and controls. Most of the special symbols reflect values under your control: you can change them with the appropriate control word or command option, but not with the .SE [Set Symbol] control word.

All other system symbols (those that do not begin with "&$") can be manipulated and modified by .SE [Set Symbol] control words within the input file.

SYMBOLS FOR THE SYSTEM DATE AND TIME

The symbol names for date and time values that are maintained by the system are:

Symbol Name	Description	Value Range
&SYSYEAR	Year	00-99
&SYSMONTH	Month	01-12
&SYSDAYOFM	Day of the month	01-31
&SYSDAYOFW	Day of the week	1-7 (1 = Sunday)
&SYSDAYOFY	Day of the year	001-366
&SYSHOUR	Hour of the day	00-23
&SYSMINUTE	Minute of the hour	00-59
&SYSSECOND	Second of the minute	00-59

The date and time values are set once and stay in effect throughout the processing of the file. You can use these symbol names to set symbol values for the date and time yourself.

No punctuation is provided by SCRIPT/VS for combining these values. You must supply it yourself when combining them. For example, to obtain the current date and time for printing on your output pages, you might enter:

```
DATE: &SYSMONTH./&SYSDAYOFM./&SYSYEAR
TIME: &SYSHOUR.:&SYSMINUTE.:&SYSSECOND
```

Notes:

- Delimit all symbols with a beginning ampersand (&) and an ending period (.). Even though there are situations when an ending period is not required, you can't go wrong by ending your symbol name with a period each time you specify it.
- The date and time symbol names must be specified with all uppercase characters.
- Leading zeros are provided with the symbol value whenever appropriate. That is, the eighth day of the month sets the value of &SYSDAYOFM 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
  ```

  As a result, leading zeros are suppressed. The symbol &SYSDAYOFM will be redefined, for your input file only, with no leading zeros. SCRIPT/VS removes leading zeros from the result of arithmetic expressions on the right-hand side of the equal sign in .SE control words.

Elaborating the System Date

If you want to print the date with the names of the months and days, your output page can include the date in the form

```
Monday, June 19, 1978.
```

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:

```
Monday, June 19, 1978.
```

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 &d2 ends with the second period. In this way, the comma needed for punctuation is concatenated with the name of the weekday.

- 19&SYSYEAR is not a compound symbol. It is resolved with only one stage of substitution. The character string "19" is concatenated with the symbol "&SYSYEAR". The first period ends the symbol &SYSYEAR. The second period is needed (in this example) for punctuation, and is concatenated with the value of the year.

SYMBOLS FOR SCRIPT/VS CONTROL VALUES

SCRIPT/VS allows you to examine the formatting parameter values it uses when processing your input file. Some of the values change dynamically. You can obtain the parameter's current value by using the system symbols.

The symbols that represent SCRIPT/VS internal formatting parameters cannot be set by .SE control words in your input file. The name of each of the following reserved symbols begins with "&$" and can be specified using either lowercase or uppercase characters. The system symbols are listed and described in Figure 33 on page 316.

These symbols allow you to specify control words in a generalized (that is, format independent) way. You can use this technique when devising a macro or APF that must be able to produce desirable results even though some of the formatting parameters can change dynamically. For example, the following sequence produces a box the width of the output page:

```
.se ind = &$IN
.if &ind = 0 .se ind = 1
.bx &ind &$CL
.in +2m;.ir +2m
The .BX control word begins
a box structure ....
.bx off
.in -2m;.ir -2m
```

which results in

```
The .BX control word begins a box structure using the cur-
rent margins. The .IN [Indent] and .IR [Indent Right] con-
trol words shift the margins to position the text within the
box. After the text is processed, the original values are
restored.
```

As another example, you might want to leave a blank page with only a figure caption at the bottom of a single column page. Perhaps the file is to be printed within different master files, each of which requires a different page length. You might code the following sequence:

```
.pa
.cm leave page for a figure
.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.

THE &$RET SPECIAL SYMBOL

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,

```
.* see if optional input file exists
.sy state OPTDATA SCRIPT *
.* if it exists, imbed it.
.* If not, don't try to imbed it.
.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 batch, &$RET is set to "-3" to indicate that the .SY [System Command] control word is not supported.

PASSING PARAMETERS TO INPUT FILES

SCRIPT/VS has three sets of symbols that can be set automatically to contain values that can be used as parameters, to pass values to an input file. These are:

- Symbols you can use to pass parameters to the input file from the SCRIPT command line.
- Symbols set by tokens on the .AP [Append] and .IM [Imbed] control words.
- Symbols set by tokens when a macro is called.

SETTING SYMBOLS WITH THE SCRIPT COMMAND

You can use the SYSVAR option of the SCRIPT command when you want to pass values to the input file from the SCRIPT command line. You can set the SYSVAR symbols from the SCRIPT command line to specify processing techniques to be used for that formatting run.

The symbols that you can set with the SYSVAR option have names starting with "SYSVAR" and have one alphameric character appended: 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 by the operating system. Consequently, when you include the symbols in an input line, always use the uppercase symbol name and character-string values.

In a sample input file named OUTLINE, &SYSVARA is used to bypass parts of the document and &SYSVAR2 is used to terminate processing before completion:

```
.if &SYSVARA = ATYPE .go aproc
.if &SYSVAR2 = NOGO .qu
... aproc
```

When you use &SYSVAR symbols, it is good practice to put comments at the beginning of your input file so that other users who process the file are aware of each &SYSVAR symbol and the meanings of its values.

For details about the SYSVAR option of the SCRIPT command, see "Chapter 2. Using the SCRIPT Command" on page 13.

SYMBOLS SET WHEN A FILE IS IMBEDDED OR APPENDED

You can pass values to an imbedded file and an appended file with the .IM [Imbed] and .AP [Append] control words. The format of the control word, for example, might be:

```
.im finance George 125 $21.50 '18-7'
```

A numeric value can be passed without enclosing it in quotes. However, a character string that contains imbedded blanks or special characters must be enclosed in single quotes, as shown.

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 symbol values passed. Up to fourteen symbol values can be passed when a file is imbedded or appended. These symbol values 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 10. Combining SCRIPT/VS Files" on page 105 for details about imbedding and appending files.

SYMBOLS SET WHEN A MACRO IS PROCESSED

You can pass values to a macro when your input file calls the macro. However, the values are for 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	4
&*1	fries
&*2	shake
&*3	nosauce
&*4	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 a single token can contain embedded blanks and special characters if it is enclosed in single quotes. Also, macro tokens are not limited to eight characters; tokens passed by the .IM [Imbed] and .AP [Append] control words cannot be longer than eight characters. See "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137 for details about specifying symbols within macro instructions.

Note: Symbols whose names begin with an asterisk (*) are treated differently than other symbols. Other symbols are globally available to all files and macros, but symbols whose names begin with * are local to a particular macro at a particular level of nesting. The symbols "&*" and "&*0" through "&*n" that are used to pass tokens to a macro are "macro-local symbols." Each time a macro is called, a new set of macro-local symbols is established for it. The set lasts until the macro completes.

Unlike other symbols, macro-local symbols, when undefined, are replaced during symbol substitution with the null string.

SOME THINGS YOU CAN DO WITH SYMBOLS

There are many uses for symbols (that is, symbolic names) in SCRIPT/VS input files. As you become more familiar with SCRIPT/VS, you will find many applications for symbol processing. Some techniques you might want to use are described in the following sections.

SETTING THE CURRENT PAGE NUMBER

You can set a symbol to be equal to the value of the current page number when the .SE [Set Symbol] control word is encountered. For example,

```
.se pagenum = &
```

A single ampersand on the right-hand side of the equal sign of a .SE control word is replaced with the character string of the current page number, including its prefix, if any. Elsewhere in your document, you can refer to the page number with its symbol name. To continue the example,

```
For details, see page &pagenum..
```

Whenever the &pagenum symbol occurs in your document, SCRIPT/VS replaces it with whatever the page number was when the .SE [Set Symbol] control word was processed.

You can use this technique to create page numbers for an index, to build a list of figures that includes the page number for each figure, and to refer to another page in your document. When you refer to a page further along in your document, the symbol you use has not been set yet. To overcome this and resolve all valid symbols, use the TWOPASS option of the SCRIPT command when you format the document for printing.

NUMBERING FIGURES

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. You can have separate numbering sequences for figures, photographs, tables, and other types of artwork. You can have a separate numbering sequence for each chapter or section. You can even build a list of figures, including figure numbers and page numbers, that you can update easily. Most important, you can rearrange the figures as often as you please without having a monumental task of renumbering them each time.

To number figures, use a "counter": a unique symbol name that refers to (and contains the value of) the figure number. The figure number symbol (that is, the counter) is set at the beginning of the input file, or in a separate file that is imbedded at the beginning of the input file.

The sequence of the figure number symbols reflects the actual sequence of the figures as they appear in your document. Each figure number symbol can be anything you care to name it.

Each figure number symbol is set based on the current value of a counter. The counter is incremented before the next figure number is set. For example,

```
.se fcnt = 1
.se afigno = &fcnt.;.se fcnt = &fcnt + 1
.se bfigno = &fcnt.;.se fcnt = &fcnt + 1
.se cfigno = &fcnt.;.se fcnt = &fcnt + 1
     .
     .
     .
.se zfigno = &fcnt.;.se fcnt = &fcnt + 1
```

Each figure number symbol is set to the current value of the figure counter, &fcnt. Each input line in the figure number sequence (above) consists of two control word statements: the first sets the value of the unique figure number symbol; the second increments the figure counter, which is used to set the value of all figure number symbols.

You can establish a separate numbering sequence (that is, a separate counter) for tables, photographs, and other kinds of artwork, giving each type of illustration its own symbolic sequence. When you enter figure captions or figure references in text, you can use the symbol name instead of a figure number. For example,

```
Figure &efigno.. Organization Chart
     .
     .
     .
See figure &mfigno. for a ....
     .
     .
     .
Figure &mfigno.. Promotion Scheme
```

You can also assign page number symbols as you create the input file. The page number symbol can relate to the figure number symbol it is associated with. For example,

```
Figure &afigno.. The Executive Board
.se afigpa = &
     .
     .
     .
Figure &bfigno.. The VP's Staff
.se bfigpa = &
```

In this example, symbols of the form "&figpa" represent symbols for the figure page numbers, and are associated with a corresponding figure number symbol of the form "&figno".

The List of Illustrations can refer to the figures in the following manner:

```
.sx /Figure &afigno.. The Executive Board/ ./&afigpa./
.sx /Figure &bfigno.. The VP's Staff/ ./&bfigpa./
          .
          .
          .
.sx /Figure &efigno.. Organization Chart/ ./&efigpa./
.sx /Figure &mfigno.. Promotion Scheme/ ./&mfigpa./
          .
          .
          .
.sx /Figure &fffigno.. Subsidiaries/ ./&fffigpa./
```

When you want the List of Illustrations to appear at the front of your document, the page number symbols might be set after they are needed. To print your document with SCRIPT/VS-generated page numbers in the List of Illustrations, you can use the TWOPASS option of the SCRIPT command. For details about the TWOPASS option, see "Chapter 2. Using the SCRIPT Command" on page 13.

Prefixes and Suffixes For Figure Numbers

You can have a figure number in the form "2-1", where "2-" (a prefix) might refer to the chapter or section that contains the figure. You can also have a figure number in the form "5A", where "A" (a suffix) might refer to part A of figure 5.

When you use a prefix or suffix, you are taking a character string (the prefix or suffix) that can't be part of an arithmetic expression and adding it to a character string that can (the value of the figure number symbol or counter). The result is a character string that can no longer be processed as part of an arithmetic expression (that is, you cannot increment the figure-number part of it).

Therefore, when you want to use prefixes and suffixes, you must save the numeric component of the figure number to pass along to the next figure number symbol that increments it.

When you use a counter and include a prefix or a suffix, you merely modify the figure number symbol's input line:

```
.se cfigno = '2A-&fcnt.';.se fcnt = &fcnt + 1
     .
     .
     .
.se cfigno = '&fcnt.-B';.se fcnt = &fcnt + 1
```

Note: A period (in the above examples) is used to mark the end of a symbol name. A single quote does not delimit a symbol name.

EXTENDED SYMBOL PROCESSING

Although a control word or macro must ordinarily begin in column 1, you can invoke a control word or macro at any point in the input line by setting it as the value of a symbol, preceded by the control word separator. When a symbol value begins with the control word separator (;), the rest of the value is treated as though it began on a new line. Therefore, a control word (set as the value of a symbol) can be processed by SCRIPT/VS as though it 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 must begin in the first character position of an input line to be recognized as a control word. However, the symbol &BR, defined with

```
.cw
.se BR = ';.br ;'
.cw ;
```

can be interpreted by SCRIPT/VS, and result in a break, no matter where it occurs in an input line (assuming symbol substitution is on). The symbol &BR is interpreted as though you had a new input line starting with ".BR;". For example, the input line

```
This is line one.&BR.This is line two.&BR
```

is formatted as though the file had the following four input lines:

```
This is line one.
.br
This is line two.
.br
```

Note that the symbol &BR is delimited with a period. Without the period, SCRIPT/VS would attempt to substitute the value of the symbol &BRThis (which might be undefined).

When you set symbols to take advantage of the extended symbol processing capability of SCRIPT/VS, remember to change the control word separator, as in the example above.

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 of the element. When you format your document for output, the entire array of values can be referred to by a single symbol name. An array symbol is defined with the .SE [Set Symbol] control word. For example,

```
.se name() = value
```

The parentheses indicate that this is an element of an array and "value" is any expression that can legally appear on a .SE [Set Symbol] control word line. The notation () is a shorthand way to specify the "next" element of the array.

When SCRIPT/VS encounters the array symbol value in the form:

```
&name(*)
```

it replaces "&name(*)" with the values of all the currently defined array elements, in the order in which they are indexed. A comma and blank separates the individual elements. You can specify different array separator characters using the .DC [Define Character] ASEP control word (for details, see the discussion of ".DC [Define Character]" on page 217).

If you are creating an index for a document, you can use array symbols to identify the page numbers for an index item that is referred to on many pages. When the input line

```
.se bslist() = &
```

appears in a number of places in a document, then the expression

```
best sellers  &bslist(*)
```

might result in the index entry

```
best sellers  10, 12, 20, 42
```

When the output line is too long because of the expansion of an array symbol, the line's first part is used as one output line and the remainder is printed on the next output line.

You can also cancel an array symbol by using the OFF parameter of the .SE [Set Symbol] control word. If the symbol is an array symbol and no subscript is provided, the entire array is cancelled.

CONTROLLING THE ARRAY ELEMENTS

Each element in an array has a value associated with it. In the example used above, 10 is the value of the first element, because it is the first one encountered; 12 is the value of the second element; and so on.

You can refer to any element of the array with the array's symbol name and the element's index number in the form

```
&name(n)
```

where "n" is the positive integer that identifies the position of the element within the array.

An array symbol reference can be used anywhere that a non-array symbol can be used. If the element "n" exists in the array, its value is substituted just as a normal symbol's value would be. If the symbol exists but has no element "n", a null value is substituted. If the symbol is not defined at all, the symbol is treated as an undefined symbol.

You can specify which array element you wish to set by including a number (identifying its location within the array) within the parentheses. For example, the input line

```
.se list(1) = &
```

sets element number 1 of the array with the current page number. When you list all the elements of the array, this entry will be listed first, even if it is not the first one set.

This is convenient for setting primary index entries (that is, page numbers you want listed first). Here's another example:

```
.se name(1) = 1
.se name(47) = 2
.se name(25) = 3
.se name(2) = 4
.se name(3) = 5
```

The expression

```
&name(*)
```

results in "&name(*)" being substituted as follows:

```
1, 4, 5, 3, 2
```

In other words, SCRIPT/VS places the array element values in ascending element index order, not in the order in which they were defined. In this example, there are many available but undefined element numbers in between those that are defined. Any undefined elements in an array are ignored when the array's values are substituted.

The array element number can be another symbol. For example,

```
.se elem = 1
.se array(&elem) = &
```

The index symbol name must not be separated from the right parenthesis by a symbol delimiter (that is, a blank or a period). 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 will substitute correctly only if they are simple, not compound, symbols.

Accessing the Index Counter

Every array has an element zero, represented by the symbol name

```
&name(0)
```

Element zero is an index counter, and tells SCRIPT/VS which element to set next if you didn't specify one.

Setting the Index Counter

The expression "name()" is treated as an index counter as well as a symbolic expression. Each time SCRIPT/VS encounters the expression, it assumes that the next element of the array is to be filled. If you never specify a number within the parentheses of an array symbol, SCRIPT/VS begins numbering with element 1.

It is possible to set the initial value of the array index counter, as follows:

```
.se name(0) = n
```

where n is any non-negative integer. Then, the first occurrence of ".se name()" with no element specified, would be equivalent to ".name(n+1)" and the counter would be incremented from there.

In this way, you can start the automatic indexing of an array at element 5, for example, and reserve elements 1 through 4 for explicitly specified definitions.

If you do not set the index counter explicitly, it will be incremented from the index value of the element last set. For example,

```
.se name() = first
.se name(3) = second
.se name() = third
```

The first element of the array is set to the value "first", element 2 has a null value, element 3 has the value "second", and element 4 has the value "third".

For substitution of arrays, you can make SCRIPT/VS substitute all elements of the array (except element zero), or you can make it substitute just a single element.

The notation &name(5) causes only element 5 to be substituted. The notation &name(*) causes all elements of the array to be substituted, as previously described.

Any symbol is potentially an array symbol. The symbol &XYZ, for example, is actually element zero of a possible array. &XYZ(0) refers to the same symbolic value as &XYZ. If, after using a symbol like &XYZ, you set another element with:

```
.se XYZ(5) = 'last letters'
```

be careful about the value previously set in element zero (that is, in symbol &XYZ). If the value is not a number, you will get an error message if you ever use the shorthand notation where element zero is supposed to contain the current index.

CHAPTER 12. WRITING SCRIPT/VS MACRO INSTRUCTIONS

SCRIPT/VS allows you to define your own processing controls, called macro instructions. The definition of a macro instruction can consist of SCRIPT/VS control words, GML markup, symbols, text lines, and macros.

You can define macros for GML processing (APFs), to provide additional formatting controls, or to modify the action taken by a SCRIPT/VS control word.

WHEN SHOULD YOU USE MACROS?

Many macro-like functions can be performed by symbols that are defined as control word strings. Sometimes, though, you may need to define a macro to perform a function that symbol processing alone cannot provide. For example, the control word sequence

```
.se x = &x + 1;.se y = &x
```

is intended to increment the symbols x and y. Because SCRIPT/VS always scans an entire input line for symbols first, &y is set equal to the current value of &x and only &x is incremented.

You can perform this sequence properly by defining a macro. For example,

```
.dm increment /.se x = &x + 1/.se y = &x/
```

After SCRIPT/VS processes the macro

```
.increment
```

&x and &y have equal values, since the two .SE [Set Symbol] control words are processed sequentially.

Macros can also be very useful when you want 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 are provided with SCRIPT/VS, you might want to define additional head levels.

HOW TO DEFINE A MACRO

Define macros with the .DM [Define Macro] control word. Since SCRIPT/VS processes macros as control words, an undefined SCRIPT/VS macro is treated as an invalid control word.

When you define a SCRIPT/VS macro, you must name the macro and specify the input lines to be processed whenever the macro is called. Any SCRIPT/VS control words can be redefined by macros. For example, the .PP [Paragraph Start] control word can be defined as

```
.dm pp /.sk/.il 3/&*/
```

The macro definition elements (usually control words) are separated by delimiters. The delimiter, usually a slash (/), 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 special symbol &* represents "the following text" (that is, the line passed to the macro for formatting). For example, when the input line

```
.pp On second thought,
```

is processed, &* has a value of "On second thought,".

The single line form of the .DM [Define Macro] control word is shown above, and is restricted to one input line. The input line is broken at delimiter characters into separate macro lines. Each component becomes a separate line of the macro.

The subscripted form of the .DM [Define Macro] control word allows you to define macro lines on separate input lines. Each macro line can contain several control words, by using the control word separator. For example, you could redefine the .PP [Paragraph Start] control word as follows:

```
.dm pp(5) /.sk/
.dm pp(10) /.il 3/
.dm pp(15) /&*/
```

The macro line number in parentheses is also called the subscript. If the number is omitted from the parentheses, SCRIPT/VS will automatically use an increment of 10, starting at 10. Macro line numbers do not have to be defined in any particular order, nor do they have to be sequential numbers. However, when the macro is used, it is executed in subscript sequence, which is not necessarily the sequence in which the macro lines were entered. When you use the subscripted form of the .DM control word, you can modify it later without respecifying all parts of the macro definition. For example, to increase the indention caused by a previously defined .PP macro, you can issue:

```
.dm pp(10) /.il 5m/
```

or you can cause the .PP macro to start an inline keep by specifying

```
.dm pp(12) /.kp 3/
```

You cannot mix the two forms of the .DM control word. That is, if you select the subscripted form you must put only one macro line with the control word.

MACRO NAMING CONVENTIONS

The macro name can be up to 10 characters long, but cannot contain special characters or imbedded blanks. The name can be the same as the two letter name of a control word, in which case its definition redefines the function of the control word. When you enter a macro name as part of your input file (after you've defined it), enter it as though it were a control word, with a period in column 1.

LOCAL SYMBOLS FOR MACROS

In SCRIPT/VS, most of the input to be processed is text. The text can contain any character string, including strings that look like control words or symbols. Tags, macros, control words, and symbols are merely character strings that have special meaning based on the context in which they occur. Therefore, an undefined symbol is regarded, by SCRIPT/VS, as an ordinary character string.

For logical or arithmetic operations, you might prefer an undefined symbol to be replaced with the null value. Nice as that would be, SCRIPT/VS doesn't replace all undefined symbols with the null value because the "symbol" might be a valid character string of text instead.

However, within macros SCRIPT/VS replaces some undefined symbols with the null value. All symbol names that start with "&*" are local symbols: these become null if not defined. There is a different set of local symbols for each macro, and for each occurrence of a macro call. Any symbol name that does not begin with "&*" is a global symbol; the same value for global symbols is used for all macros and outside macros. For symbol substitution within a macro, the following rules apply:

- All global symbols when substituted in a macro are considered text character strings if undefined as symbols.
- All local symbols are considered null if not defined.

When SCRIPT/VS processes a macro, it assigns values to certain designated local symbols based on the macro's input text line. The local symbols are named &*0, &*1, &*2, and so on. Values are assigned to a new set of local symbols each time a macro is called.

The symbol &* contains the entire character string on the macro's input line (except for the macro name). The symbol &*0 represents the number of words that make up the character string. The symbol &*1 contains the first word, the symbol &*2 contains the second word, and so on. For example, when SCRIPT/VS encounters the following input line

```
.process fileb 10 filea no
```

it sets the following values for the macro's local symbol values (&*, and &*1 through &*n are called "tokens"):

Symbol	Value
&*	fileb 10 filea no
&*0	4
&*1	fileb
&*2	10
&*3	filea
&*4	no
&*5-&*n	(null value)

When a macro symbol beginning with &* is not set by the input line, it has a null value. When you want to assign a null value to a macro symbol without also assigning null values to all subsequent tokens on the input line, use the percent sign (%) to represent the null-value token. For example, the macro input line

```
.insert filea 10 % fileb 15
```

results in the symbols being set as:

Symbol	Value
&*	filea 10 % fileb 15
&*0	4
&*1	filea
&*2	10
&*3	(null value)
&*4	fileb
&*5	15
&*6-&*n	(null value)

Note: In SCRIPT/370, the symbols set when a macro was called were &0 through &9; these symbols are no longer set by SCRIPT/VS when a macro is called. See "Chapter 18. Compatibility with SCRIPT/370" on page 177 for information on converting local symbols in a SCRIPT/370 macro to SCRIPT/VS local symbols.

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: Local symbols are replaced with null values if undefined, but only when the current input source is a macro.

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.

Once a control word is redefined, its function exists in two forms: as a SCRIPT/VS control word, and as a user-defined macro. However, when it is encountered in the input file, the function defined by the macro always takes effect when macro substitution is on (.MS ON).

When macro substitution is on, you can specify that the SCRIPT/VS control word function is to be effective, even when a macro of the same name is defined, by stipulating that it is to be executed as a control word, using the .EC [Execute Control] control word. For example, the input line

```
.dm sk /.sp &*./.il 5/
```

redefines the .SK [Skip] control word, to space lines and indent the first output line after the line space.

When you want the .SK [Skip] control word to be effective (as SCRIPT/VS has defined it), and you do not want to turn off macro substitution, issue

```
.ec .sk 4
```

to skip four lines and not indent the next output line. In other words, the .EC [Execute Control] control word executes the control word even if a macro of the same name exists.

When macro substitution is off (.MS OFF) and you want the macro function to be effective (instead of the SCRIPT/VS control word of the same name), 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. The .EM [Execute Macro] control word can also call a macro that does not have the same name as a control word, even when macro substitution is off (.MS OFF).

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, you want to redefine the .PP [Paragraph Start] control word to put two line spaces between paragraphs instead of one:

```
.dm pp(1) /.sk/
.dm pp(2) /.ms off/
.dm pp(3) /.pp &*/
.dm pp(4) /.ms on/
```

Had we not turned macro substitution off with the .MS OFF control word, statement 3 would substitute the macro definition for .PP until SCRIPT/VS flagged it as a severe error and terminated your document's formatting.

However, sometimes turning off macro substitution is not an adequate solution to the problem. For example, you can change the .IM [Imbed] control word so that the name of the imbedded file is typed (or displayed) when it is imbedded. If you define the macro (that is, redefine the control word) in the following way:

```
.dm im(1) /.ty &*/
.dm im(2) /.ms off/
.dm im(3) /.im &*/
.dm im(4) /.ms on/
```

macro substitution is turned off to prevent an endless loop from occurring. However, when macro substitution is turned off, substitution is prevented for any macro that might be part of the imbedded file (as well as files it might imbed).

Instead, you should use the .EC [Execute Control] control word to tell SCRIPT/VS that the input line is to be treated as a control word even though a macro of the same name might be defined. For example, the following lines

```
.dm im(1) /.ty &*/
.dm im(2) /.ec .im &*/
```

redefine the .IM [Imbed] control word, allowing for macro substitution in the imbedded file.

HOW VALUES ARE SUBSTITUTED FOR SYMBOLS WITHIN A MACRO DEFINITION

When symbol substitution is on, the .DM [Define Macro] control word line is scanned for symbol names. If you define a macro that contains a symbol, you usually want the symbol's value substituted for the symbol name when the macro is encountered as an input line, rather than when the macro is defined. Therefore, turn off symbol substitution (using the .SU OFF control word) before you define the macro, to allow the symbol (rather than its value when the macro is defined) to be part of the macro definition. For example,

```
.su off
.dm ofs /.sk/.of &offset./
.su on
```

In this example, &offset is a symbol that might have a value when SCRIPT/VS processes the .DM [Define Macro] control word. If substitution is ON, the symbol's value becomes part of the macro definition instead of the symbol &offset. The macro .OFS would then result in a hanging indention of that amount, rather than of the value of &offset when SCRIPT/VS encounters the macro .OFS.

REDEFINING SCRIPT/VS FORMATTING CONVENTIONS

A control word, in SCRIPT/VS, is used to request a specific SCRIPT/VS function. You can use a macro to redefine the function of a SCRIPT/VS control word.

SCRIPT/VS has implicit formatting functions, too. Input lines that are null reset line continuation, and those that begin with a blank or tab character cause a break. You can use a macro to redefine these functions.

PROCESSING INPUT LINES THAT ARE EMPTY

When SCRIPT/VS encounters a null input line (that is, a line that contains nothing at all), it generates and executes a which resets line continuation.

To redefine the SCRIPT/VS implicit formatting convention for null lines, define a .NL [Null Line] macro that will be executed whenever a null line is encountered. For example,

```
.dm nl /.sk 2/
```

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

You can also define the null line to be completely ignored by SCRIPT/VS:

```
.dm nl /.*/
```

PROCESSING INPUT LINES THAT BEGIN WITH A BLANK OR A TAB

When an input line begins with a blank (called a leading blank) or a tab (called a leading tab), SCRIPT/VS does not concatenate the line with the previous input line. That is, a break occurs.

Breaks are provided by executing the .LB [Leading Blank] control word when a leading blank is encountered, and by executing the .LT [Leading Tab] control word when a leading tab is encountered. Both of these control words function exactly the same as the .BR [Break] control word. However, after the break occurs, the leading blank or tab remains on the input line and is processed as part of the line.

As with null lines, you can control the actions to be taken for leading blanks and tabs by defining a .LB and .LT macro.

When you want the leading blank and leading tab to be processed by SCRIPT/VS as just a blank (or just a tab) that happens to occur as the first character (that is, not processed differently than other blanks or tabs), redefine the control words with:

```
.dm lb /.*/
.dm lt /.*/
```

The tab or blank at the beginning of the input line will be concatenated with the previous input line. It will not necessarily appear at the beginning of an output line.

Note: The .NL [Null Line], .LB [Leading Blank], and .LT [Leading Tab] functions are not performed for a line that would otherwise call for them when the line is processed in literal mode (that is, preceded by the .LI [Literal] control word). Null text lines still reset line continuation if the previous line ended with a continuation character, but the .NL control word or macro is not executed.

SPECIFYING A MACRO LIBRARY

When a macro name cannot be resolved (because there was no previous definition set with a .DM [Define Macro] control word), SCRIPT/VS can look for its definition in a macro library.

Each macro definition is a member of the macro library. The member name is the macro name without the leading period, restricted to eight uppercase characters. Symbol definitions and macro definitions may be members of the same library. However, only the first line of a 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.

As with symbol definitions, the rightmost non-blank character of each line is treated as a delimeter, and is deleted.

You can use the macro library in two ways:

- To explicitly set a macro name, by using the .DM control word to cause 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 no definition, the library is searched for a member with the same name as the macro.

When your input file contains macros that are defined in a macro library, you can specify that SCRIPT/VS is to search the macro library for any unresolved macro it encounters:

```
.ly on
  or
.ly mac
```

The ON parameter of the .LY [Library] control word specifies searching the macro library for unresolved macros and symbols. The MAC parameter allows library searching only for unresolved macros. You can use the OFF or SYM parameters (described above) to turn off library searching for unresolved macros.

Since searching macro libraries for unresolved symbols is expensive in terms of processing time, it is recommended that .LY MAC be used except for short periods when you expect symbol definitions to be returned; then .LY SYM or .LY ON should be used.

See "Using the SCRIPT Command" for details about the LIB option and macro libraries.

This section explains how the GML starter set profile and APFs were written and suggests how you can create your own profiles and Application Processing Functions (APFs) to support processing requirements at your installation. The purposes and advantages of using GML are not discussed here, nor is the analysis of a document that is needed to define tags and determine which APFs you need to write. Please refer to the Document Composition Facility: Generalized Markup Language (GML) User's Guide for these discussions, as this chapter assumes you are familiar with general GML concepts and terminology.

The GML starter set consists of the following:

- A document profile called GDOCPROF which has two main purposes:

  1. To establish the GML tag to APF relationship.
  2. To establish the formatting style of a "General Document."

- A macro library which contains the implementations of the APFs invoked when GML tags are encountered in a document.

THE ROLE OF A DOCUMENT PROFILE

You will probably need a number of document profiles if you plan to use documents that you have marked up with GML for a number of different purposes. The profile provided as part of the starter set is rather general in that its actions can be to some extent controlled by:

- use of the SYSVAR option.
- the logical device for which processing is taking place.
- setting various symbols in the profile which are used in the the macros provided in the starter set macro library to control the style of formatting for document elements (for example, the skips between paragraphs).

CREATING YOUR OWN PROFILES

You can obtain a listing of the starter set GDOCPROF profile to help you understand the APF processing associated with each starter set GML tag, and to aid you in creating your own document profiles.

If you wish to use your own profile in conjunction with the starter set, one of the last entries in it should be the .IM [Imbed] control word to imbed the starter set profile (GDOCPROF). Your profile will be processed ahead of the starter set profile, allowing use of both your tags and the starter set of GML tags.

This technique permits the mapping for any GML tag defined in a private profile to override the mapping for the same tag in an installation's shared profile when the two profiles are used together.

Note: If you wish to use the starter set profile by imbedding it in your profile, and you want to override the values of the starter set APF variables, you should remove the .SE [Set Symbol] statements from the starter set profile which define these values for the starter set APF variables. (These are described in the discussion of "Symbols Within Starter Set APFs" on page 147.) If you do not do this, the values that you set will be overridden by the starter set profile.

GML TAG TO APF MAPPING

The main function of a document profile is to relate GML tags and attribute labels to the processing that is to be performed on the elements they describe.

A SCRIPT/VS GML tag is a symbol that is folded to uppercase before it is substituted. This folding occurs because the tag is delimited with a GML delimiter. The default GML delimiter is a colon (:), but this may be changed by use of the .DC [Define Character] control word.

Starter set GML tags are associated with APFs in the starter set profile using the .SE [Set Symbol] control word. If you look in the starter set profile (GDOCPROF), you will notice that in each set statement which does this mapping, the tag name on the left side of the equal sign (=) is in uppercase. If this were not done, the tag would not be recognized when the tag was encountered in the document, as the name is folded to uppercase by the symbol processor before substitution is attempted.

Many tags in the starter set map to an APF which consists of a macro. You must ensure that when such a GML tag is recognized and substituted, the macro that results from the substitution will be processed as if it had been entered at the start of the line. You can do this by means of the SCRIPT/VS extended symbol processing capability. When a symbol whose value starts with a control word separator is substituted, the results are treated as if they had occurred at the start of a line.

In the example below, the tag "PREFACE" is mapped to the ₫pref macro by the .SE [Set Symbol] control word. The semicolon is the control word separator:

```
.'se PREFACE = ';.₫pref '
```

Note: Since the .SE [Set Symbol] line itself will be scanned for control word separators before it is processed, this will only work if one of the following techniques is used:

- The control word separator is undefined by using .DC CW OFF.
- The control word modifier (.') form of the set statement is used as shown in the example.
- The control word separator character is temporarily redefined to be something other than the character used for the control word separator in the set statement.

APFS FOR TEXT ITEMS

A GML tag normally identifies a document element which is a paragraph unit or larger. However, some tags (for example, phrase (:PH)) identify elements of the document that occur within a paragraph (or other text). These elements are known as "text items."

When processing a text item, it is important that all the characters (including blanks) that were entered in the document are preserved for the user. This is done by the use of a continuation character as the first character of the tag symbol value.

When the tag is substituted, the continuation character is appended to any preceding text and any blanks are preserved. However, as we have seen above, a tag whose APF is a macro must be processed as if it were entered at the beginning of the input record. For this, a control word separator must be the first character of the value.

The dilemma is resolved by storing the control word separator character in a symbol so that it will only be substituted during the substitution of the tag. In the example below, the symbol &₫cont has the value of the continuation character, and &₫cw is undefined but will be set to the control word separator character when the document is being processed.

```
.se Q = '&@cont.&@cw..''em .@q '
```

When the tag is found in a document, the following stages of substitution result:

```
Mary said .'em .@q Hello; how are you?
Mary said &@cont.&@cw..'em .@q Hello; how are you?
Mary said +&@cw..'em .@q Hello; how are you?
Mary said +
.em .@q Hello; how are you?
.@q Hello; how are you?
```

Note: Since the text item may contain as text the character used as the control word separator character, it is necessary to stop it from being misinterpreted when the APF macro is invoked. For control words, this can be done using the control word modifier (.') form. For macros, you can achieve the same effect by using the .'EM [Execute Macro] control word in front of the macro as shown in the example above.

See the discussion of "Use of the .LI [Literal] Control Word" on page 149 for a further discussion of text items.

SYMBOLS WITHIN STARTER SET APFS

By convention, all global symbols used in the starter set APFs, other than the tag symbols, start with the commercial "AT" character (@). These symbols contain variables that are used to direct the formatting. They are often used as symbolic parameters on the actual control words issued by the macros. The default values for these symbols (shown in parentheses below) are used by the starter set APFs unless overriding values are supplied.

Symbol Name	Function (Default)
&@cols	number of columns (1 for non 3800 logical devices; 2 for 3800 logical devices)
&@cont	continuation character (+)
&@cw	control word separator character (;)
&@dlterm	definition term length (10M)
&@dsk	definition list skip (1 line)
&@dthi	definition term highlight (2)
&@duplex	controls duplex printing (no)
&@eframe	default figure end frame (.sx //-//)
&@fin	figure indention (0)
&@headctr	heading counter (no)
&@ll	line length (default SCRIPT/VS line length except for 3800 when it is 6.6 inches)
&@oin	ordered list indention (4M)
&@osk	ordered list skip (1 line)
&@psk	paragraph skip (1 line)
&@puncmove	controls movement of punctuation around quotation marks (yes)
&@qin	quotation indention (0)
&@sframe	default figure start frame (.sx //-//

&@sin simple list indention (4M)

&@ssk simple list skip (1 line)

&@tipage controls printing of title page (yes)

&@uin unordered list indention (4M)

&@usk unordered list skip (1 line)

&@xin example indention (0)

&@1pin first line paragraph indention (0)

You can expand or change this list to reflect your installation's requirements.

In addition to the symbols shown above, there are a number of macros in the starter set which are important in the definition of the basic style of the document. Macros that perform the processing for attributes have the same name as the label of the attribute they handle. Macros that are invoked by the symbols for the tags themselves, and inner macros used as "subroutines" by the primary macros, have names that start with the "AT" character (@). You may wish to consider modifying or replacing these macros to achieve the style that you need:

Macro Name	Function
@format	This macro is invoked to establish the line formatting style for a document. For example, if you wish to format the document in ragged right form you can redefine this macro to include a .FO [Format Mode] control word to do this.
@fontset	This macro establishes the form of the various levels of highlighting. If you do not like the starter set conventions for the levels of highlighting, or wish to increase the number of levels, you should change this macro and perhaps define some new tags in the profile.
stitle	This macro processes the stitle attribute and hence its name does not start with an @ character. It controls the style of the running footings that are generated whenever an stitle attribute is used (either explicitly or implicitly). Redefine this macro if you wish to change the style of the running footings produced by the starter set.
sec	This macro processes the sec attribute and hence its name does not start with an @ character. It controls the style of the running headings that are generated whenever a sec attribute is used. Redefine this macro if you wish to change the style of the running headings produced by the starter set.
@tipage	This macro produces the title page. To change the format and content of the title page produced by the starter set you must modify this macro or change the mapping of the :etitlep tag in the profile to an APF of your own.
@2col	The form of two column output produced by the starter set depends on the calculations performed by the @2col macro using the line length value that was set in the symbol &@ll in the profile. These calculations assume a certain gutter width. You can redefine the @2col macro to achieve the two column formatting you want. Of course, you could also generalize the @2col macro concept and extend it to include the ability to format the document in many columns (SCRIPT/VS allows a maximum of nine columns).

STARTER SET MACROS FOR ATTRIBUTE PROCESSING

The @scan and @exatt macros provided as part of the GML starter set may be of use in building your own APFs. You should refer to the comments in the commented form of these macros for a more complete description of their function:

@scan scans the line given to it as an argument for attributes. Having scanned the line @scan returns the text that follows the attributes in the symbol &@line for use in the invoking macro.

For example:

```
.@scan &*
```

will cause the @scan macro to scan the string in the symbol &* and to extract the attributes. The attribute names and their values are stored in an element of the array symbol &@att(*). This symbol is examined by the @exatt macro.

Note that one of the checks made by the @scan macro when looking for attributes is that the line given to it contains at least two leading blanks. If this is not so, the @scan macro will assume that there are no attributes in the line to be scanned. This condition will always be satisfied when the @scan macro is invoked from a macro with the APF for a GML tag. For example:

```
.se PREFACE = ';.@pref '
```

provides the two blanks on the &* line to the macro .@pref because there is one blank after the macro name (@pref) and there will always be one additional blank if the user specifies attributes.

@exatt causes any macros with names given by the attribute labels found by the @scan macro to be processed. For example:

```
.@exatt
```

will cause the @exatt macro to execute all of the macros with names identical to the attribute labels found by @scan.

In addition, the @exatt macro can be requested to execute only a certain attribute by giving the macro name as a parameter. For example:

```
.@exatt frame
```

This will cause the frame attribute macro to be executed if it is found by the @scan macro.

The @exatt macro can also be requested to execute all of the attribute macros found by the @scan macro except for a specified one. For example:

```
.@exatt ~ id
```

This will cause all attribute macros found by the @scan macro to be executed except for the **id** attribute macro.

USE OF THE .LI [LITERAL] CONTROL WORD

If you look at the definitions of many of the macros from the starter set you will notice that the text which forms the macro parameter (contained in &* if there are no attributes being scanned, or in &@line if the @scan macro was invoked) is always preceded by the .LI [Literal] control word. For example:

```
.*******************************
.* Reset indents etc           *
.*******************************
.in -&@qin
.ir -&@qin
.li 1
&*
```

This is for two reasons:

- The text may start with a period. In this case, SCRIPT/VS would treat the line as a control word, and the user would be given a message saying that an invalid control word had been found.
- The text may start with leading blanks. If the tag is identifying a text item, this would have the effect of causing a break and erroneous output would result.

Use of the .LI [Literal] control word will prevent these effects.

Note: For text items, when no text follows the tag, &* or &@line will have a null value. When these symbols are substituted, a null line will be generated which will end continuation. This is appropriate, since the user has implied the end of a word by placing the tag as the last data on the line.

DEBUGGING YOUR APFS

Before you start writing your own APFs you should have a clear understanding of the following SCRIPT/VS control words:

- .DM [Define Macro]
- .IF [If]
- .IT [Input Trace]
- .SE [Set Symbol]
- .SU [Substitute Symbol]

The .SE [Set Symbol] control word can perform substitution on symbolic values on the right side of the equal sign, even if substitution is off. In this way, the .SE control word can treat complex character strings as single parameters when the character string is the value of a symbol. If substitution were on, the character string would replace the symbol name before the .SE control word processed the line, and each word in the string would appear as a separate parameter. With substitution turned off, the symbol name remains on the line as a single parameter until the .SE control word retrieves its value.

GML macros often use the "index" and "substring" capabilities of the .SE [Set Symbol] control word, and take advantage of the ability of .SE [Set Symbol] to perform its own substitution while general substitution is off.

You can use the .IT [Input Trace] control word to trace the execution of your macros, or to understand the operation of starter set macros.

Sometimes you may have doubts about the way in which a particularly complex symbol will resolve. In these circumstances, it is useful to experiment to understand what will happen. You can create a file that contains various test cases, and trace the results using the .IT [Input Trace] control word:

- at your terminal in foreground environments.
- to an output file or listing using the MESSAGE (DELAY) option of the SCRIPT command.

Another useful technique you may use in foreground environments is to create a file that contains nothing but the .TE [Terminal Input] control word with the ON parameter specified. When you SCRIPT this file, your terminal will be opened for reading, and you can imbed files, excecute macros from your macro libraries, and invoke any other SCRIPT/VS functions.

Since the output that you get using the .IT [Input Trace] control word with the ALL parameter can sometimes be quite voluminous, you may want to trace just the execution of a specific macro. Since the .IT [Input Trace] control word cannot trace the excecution of a specific macro, you must modify the macro definition to add .IT ALL at the beginning of the macro definition, and .IT OFF and the end. If the macro definition comes from a macro library you must make sure that the definition is retrieved from the library using

```
.dm macroname lib
```

before attempting these modifications. You can determine the current contents of the macro definition at any time using:

```
.it snap macroname
```

CHAPTER 14. USING SCRIPT/VS WITH OTHER PROGRAMS

You can use SCRIPT/VS to format an input stream prepared by another program. You can also use SCRIPT/VS as a preprocessor, to prepare an input file for processing by another text processing system or by an application program.

USING SCRIPT/VS AS A POSTPROCESSOR

You can use SCRIPT/VS to format reports using data from data processing files. An application program could access these files, perform the necessary computations, and create an output file. The output file could contain GML markup just as if it had been created with normal text entry procedures. You will then be able to process it with the same flexibility as any of your other documents.

An alternative to the technique just described would be to have the application program 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. For example, you can automatically prepare index entries: you might define a secondary attribute called "index descriptor," make up a suitable tag (for example, "desc"), and create an APF for it. You would specify this attribute for those phrases in your document which were suitable entries for an index. For example,

```
This process, done with :ph desc='yes'.binary numbering:eph.,
can also be done with :ph desc='yes'.decimal numbering.:eph..
```

The APF would write the phrase and current page number to a file, together with appropriate document type tags identifying each.

You could sort and arrange the file in a manner appropriate to an index. The sorted file would be a SCRIPT/VS document which, when processed by appropriate APFs, would generate an index.

USING SCRIPT/VS AS A PREPROCESSOR

When you use SCRIPT/VS as a preprocessor, you want SCRIPT/VS to produce an output file that can be processed by some other text formatter or application program. To use SCRIPT/VS as a preprocessor, you must first thoroughly understand the text formatter that is to receive the output file prepared by SCRIPT/VS.

Your SCRIPT/VS input file contains 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 end-of-sentence characters). You must build a profile and APFs that interpret the SCRIPT/VS markup and generate appropriate formatter controls. (See "Chapter 13. GML Support in SCRIPT/VS" on page 145 for details about profiles, APFs, and mapping tags to APFs. See "Chapter 11. Symbols in Your Document" on page 117 for details about symbols, and "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137 for details about macros.)

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 a convention like that described in the Document Composition Facility: Generalized Markup Language (GML) User's Guide. Familiarity with the markup procedures portion of Chapter 3 of that manual is assumed.

SCRIPT/VS has a great variety of general document-handling functions which can be used independently of formatting. You can use these functions to create APFs that will translate a GML document into suitable input for another program, such as a formatter that can support photocomposers.

For example, the starter set APFs for ordered lists and list items automatically generate numbers (or letters) for the items on an ordered list. This is a desirable function, since it permits the list to be revised without renumbering all the items.

The SCRIPT/VS control words for a list item APF are shown below. They include general processing control words which maintain the number counter and insert numbers or letters into the output text stream. They also include formatting control words which indent the list and leave spaces between the items. The symbols used were set with previous control words, either in the profile or in the APF for the list.

```
.*********************************
.* Skip 1 line before the item  *
.*********************************
.sk &lskip
.*********************************
.* Keep the first three lines   *
.* together.                    *
.*********************************
.kp 3
.*********************************
.* Indent and undent first line.*
.*********************************
.in +&@listin
.un &@listin
.*********************************
.*  Increment item counter.     *
.*********************************
.se @lctr = &@lctr + 1
.*********************************
.* Insert counter and text into *
.* data stream.                 *
.*********************************
(&@lctr.)&$TAB.&*
```

Figure 12. APF for List Items

You would create a modified version of the APFs which would incorporate the general processing functions, but eliminate the SCRIPT/VS control words that result in formatting. In the example, these are .SK [Skip], .KP [Keep], .IN [Indent], and .UN [Undent]. You would insert the appropriate formatting controls of the postprocessor into the output stream in place of these control words. The SCRIPT/VS symbol substitution capability can still be used to calculate parameters for the postprocessor's formatting controls (or even to select one of a number of possible control words, although this use is not shown here).

Some of the logical sequence of formatting controls might have to be changed, however. The graphic effect of having the first line of the 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 followed by the .UN [Undent] 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 in the APF (for example, for &@&@ltyp.sk, &@ind, and &@&@ltyp.in) to achieve different formatting values.

In addition to creating APFs, you would also create a profile which would map to the new APFs. The profile would also issue control words that would turn off justification and page numbering, and the like, so the output would look like a source file.

```
.tm 0
.bm 0
.fo off
```

In the APF for the end of the document, you would "drain" the page buffer and terminate processing. This prevents SCRIPT/VS from adding blank lines to the end of the output as it normally would do to reach the end of the page.

```
.cd 1
.qq
```

You might also need to translate special characters which might be unacceptable to the postprocessor.

By having two sets of APFs and two profiles, you could continue to print draft copies of the document on a line printer while getting final output on a photocomposer via the postprocessor.

REDEFINING SYMBOLS

Many symbols used in source document markup will not require redefinition. For example, those used:

- As abbreviations for lengthy character strings.
- As references to generated information which is not format-dependent (such as a figure or section number -- but not a page number).
- To enter unkeyable characters which are represented by the same codes in both SCRIPT/VS and the postprocessor.

HANDLING DIRECTLY ENTERED CONTROL WORDS

Observing a GML convention for direct entry of control words, like that described in the Document Composition Facility: Generalized Markup Language (GML) User's Guide, makes it easy to prepare your document for another processor. The following discussion will refer to the specific conventions recommended in that book, but the information will be applicable to conventions that may be adopted by your own installation.

Source Document Management

The .CM [Comment] and .IM [Imbed] 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.

The .RC [Revision Code] control word is different, because it has a formatting effect (it inserts a revision code character to the left of an output line). If the postprocessor has a comparable function, you can define a macro called .RC which generates the corresponding postprocessor controls.

If the "revision code" function is not available, you can deactivate it by defining the .RC macro to be a comment. For example,

```
.dm rc /.cm/
```

(The technique described above could be used for all control words if the one-to-one conversion approach is taken.)

Modifying Unpredictable Processing Results

You should create a process name which reflects a specific postprocessor, device, and application. Modifications for other processes will automatically be ignored.

If the postprocessor is a formatter, you may have a need to create patches for its output. In this case, the control words in the patch would be those of the postprocessor.

Special Graphic Effects

Create a version of the graphic for this process by using the control words of the postprocessor, just as for a patch. Control words in versions of the graphic for other processes would automatically be ignored.

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 GML interpretation and in addition use the BIND option to prevent SCRIPT/VS from adding blank characters to the front of each output record. In summary:

- Use the LIB option of the SCRIPT command to refer to the appropriate symbol and macro library.
- Use the PROFILE option of the SCRIPT command to refer to the appropriate profile.
- Use the BIND(0) option of the SCRIPT command to prevent SCRIPT/VS from inserting blank characters at the beginning of each output record.

While the file produced by SCRIPT/VS will contain the correct text and markup for your postprocessor, it will not necessarily have the correct physical characteristics. Some postprocessors may require record lengths and formats, or other characteristics, that differ from those produced by SCRIPT/VS. You might have to use a utility program, or code your own, to handle such interface requirements.

CHAPTER 15. AUTOMATIC HYPHENATION AND SPELLING VERIFICATION

SCRIPT/VS provides automatic functions to both hyphenate and verify the spelling of words. Words that occur at the end of an output line can be hyphenated, and each word in your document can be checked for correct spelling.

HOW AUTOMATIC HYPHENATION WORKS

When SCRIPT/VS is formatting a line and the next word does not fit in the current line and hyphenation is on, SCRIPT/VS tries to hyphenate the word.

To find out where the word should be hyphenated, SCRIPT/VS searches the main and addenda dictionaries to see if there is an entry for the word. The SCRIPT/VS main and addenda dictionaries are discussed below.

ALTERING THE HYPHENATION PARAMETERS

You can increase or decrease the frequency of hyphenation by changing two parameters:

- THRESH, which is the hyphenation threshold (the minimum number of blank characters that must remain on the line before SCRIPT/VS attempts to hyphenate the word).
- MINPT, which is the minimum hyphenation point (the smallest number of characters acceptable as a hyphenation point for the word).

The default values are 7 for THRESH and 4 for MINPT. To change them, use the THRESH and MINPT operands of the .HY control word:

```
.hy set minpt 3
.hy set thresh 6
.hy on
```

After these control words are encountered, there must be at least six blanks left on the line before SCRIPT/VS tries to hyphenate the next word.

HYPHENATING SINGLE WORDS

Regardless of whether SCRIPT/VS is using automatic hyphenation or not, there may be occasions when you would like a word to be hyphenated if it occurs at the end of a line. The .HW [Hyphenate Word] control word allows you to specify that you want a word hyphenated, if necessary, and to specify how it should be hyphenated.

This may be convenient for long words that are normally hyphenated, or for words that occasionally need hyphenation. For example,

```
Guinevere's
.hw lighter--than--air
laughter was heard
.hw through-out
the kingdom.
```

When this line is processed, SCRIPT/VS uses the hyphens supplied as hyphenation points and suppresses the hyphens it does not need:

```
Guinevere's  lighter-than-air
laughter was heard  throughout
the kingdom.
```

Note that since "throughout" did not require hyphenation when the line was formatted, the hyphen was suppressed. For the hyphenated expression "lighter-than-air," two hyphens are used with the .HW [Hyphenate Word] control word so SCRIPT/VS prints the necessary hyphens. Note that the hyphenation rules for a .HW word apply only in this instance, and not anywhere else.

HOW SPELLING VERIFICATION WORKS

The spelling of words in your input file can be checked by the SCRIPT/VS spelling verification function when:

- You include the SPELLCHK option in the SCRIPT command, and
- You specify the .SV [Spelling Verification] ON control word in your input file.

Spelling verification is accomplished by attempting to find each word in the input line in the SCRIPT/VS main dictionary or user-created addenda dictionary.

For purposes of spelling verification, a "word" is a string of characters delimited by "word delimiters." The default word delimiters are listed in Figure 35 on page 319. You may change the word delimiters for spelling verification with the .DC [Define Character] WORD control word.

Punctuation characters are considered part of the word if they appear within it, but are removed before spelling verification is performed if they appear at the end of the word. The default punctuation characters are the hyphen (-) and apostrophe ('). You can change the punctuation characters with the .DC [Define Character] PUNC control word.

If the word in the input line contains a prefix or a suffix, SCRIPT/VS removes the prefix and suffix (to yield the word's "root") before attempting to locate the word in the dictionary. If the word's root is not found in the main or addenda dictionary, SCRIPT/VS attempts to locate the original word in the dictionary.

Spelling verification is normally performed using the main and addenda dictionaries with stem processing. Words that contain numbers are not checked unless requested with the NUM parameter of .SV.

You can specify that:

- The addenda dictionary is not to be used:

```
.sv noadd
```

- Full word processing rather than stem processing is to be performed:

```
.sv nostem
```

- Words that contain numbers are to be checked:

```
.sv num
```

THE SCRIPT/VS DICTIONARY

The SCRIPT/VS dictionary is used for hyphenating words and for spelling verification. It consists of a main dictionary provided by IBM and an addenda dictionary. The addenda dictionary is created with the .DU [Dictionary Update] control word.

THE MAIN DICTIONARY

The main SCRIPT/VS dictionary contains about 10,000 "root words." Because suffixes and prefixes are removed before a word is verified, the effective dictionary size is significantly larger. The main dictionary cannot be modified.

THE ADDENDA DICTIONARY

Documents usually contain several words that are not in the main dictionary. These words reflect the nature of the business and include jargon, acronyms, and names of people and places. Usually a technical document contains several words that pertain only to the subject of that document or of a group of related documents.

Words that are not widely used can be contained in the addenda dictionary. You create an addenda dictionary using the .DU [Dictionary Update] control word. For example, you can build an addenda dictionary that includes the names of some Greek letters:

```
.du add alpha beta gamma delta
```

The "addenda dictionary" is a dynamic extension to the SCRIPT/VS main dictionary. It is built in main storage by SCRIPT/VS via the .DU control word. You can create a separate file that consists of the .DU [Dictionary Update] control words needed to build the dictionary and then imbed the "addenda dictionary" file at the beginning of the input file.

If you use this technique to create an addenda dictionary, you can modify the dictionary with .DU [Dictionary Update] control words in the input file. For example, you can add words to the dictionary with

```
.du add epsilon zeta eta
```

and delete words from it using

```
.du del beta delta
```

Each word specified with the .DU [Dictionary Update] control word is delimited with blanks. The word can contain lowercase and uppercase alphabetic characters, the integers 0 through 9, the hyphen (-), and the single-quote mark (').

When you include single hyphens in a word, SCRIPT/VS assumes they are potential hyphenation points. 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."

Spelling verification can also be used to verify that proper names start with an initial capital letter.

When verifying, words are first checked using the case (upper, lower, or mixed) as it occurs in the input line. If no match is found, the word is next checked with all of the characters except for the initial capital (if any) changed to lower case. Again, if no match is found, a final check is made with all of the characters changed to lower case. If all three checks fail, the word is regarded as misspelled.

For example, if an entry is made in the addenda dictionary as follows,

```
.du add Paul
```

then "Paul" and "PAUL" will both be correctly spelled. However, "paul" will be regarded as misspelled.

BUILDING AN ADDENDA DICTIONARY

The addenda dictionary supplements the SCRIPT/VS main dictionary. The easiest way to determine the words needed for an addenda dictionary is to format the input file and locate each word not included in the main dictionary (because it will be flagged as "misspelled").

To create an addenda dictionary for a file (FILE10) in a CMS environment:

1. Format the file at the terminal. Make sure your input file begins with the .SV ON control word. Type

   ```
   SCRIPT FILE1 ( MESSAGE (DELAY ID) SPELLCHK
   ```

   Messages are issued to inform you of all words not correctly verified.

2. When messages are delayed, they are accumulated in the SCRIPT/VS utility file called (in CMS) DSMUTMSG SCRIPT A1 to be displayed or printed after the formatted file is completed. You can access and display the DSMUTMSG file with your text editor, or look at the output listing.

3. Look for messages that identify unverified words, identified with DSMBRG459I. By examining these messages, you can derive a list of words that are not included in the SCRIPT/VS main dictionary and are appropriate for your addenda dictionary, after eliminating words that are truly misspelled.

4. After correcting the misspelled words in the input file, you can create a file that is used to build an addenda dictionary.

   The file consists of many .DU [Dictionary Update] control words, each of which specifies a word's correct spelling and hyphenation points.

   For example, the addenda dictionary can include the correct spelling of various town names necessary for your business:

   ```
   .du add San-ta Cruz So-quel
   .du add Ap-tos Dav-en-port
   .du add Cap-i-to-la Zay-an-te
   .du add Ben Lo-mond Lom-pi-co
   .du add Bon-ny Doon
   .du add Scotts Val-ley
   ```

5. Imbed the addenda dictionary file at the beginning of the input file whenever spelling verification is desired.

STEM PROCESSING

The stem processing function attempts to generate one or more possible root words from which the input word might be derived. Suffix and prefix processing are both performed on the input word. The stem processing function does not generate a root word, or stem, less than three characters long.

When a word's prefix is removed, the resulting stem is not changed. However, when a word's suffix is removed the stem processing function derives the word's stem based on English spelling rules. For example, the word "churches" yields the stem "church", and the word "flames" yields the stem "flame."

Prefixes Removed from Words

SCRIPT/VS checks each word for a prefix and, if found, removes the prefix. The prefixes recognized by SCRIPT/VS are:

ANTI	EN	MEGA	OUT	SUB
ANY	FORE	MICRO	OVER	SUPER
BACK	IN	MILLI	PRE	TELE
COUNTER	INTER	MINI	PRO	TRANS
CROSS	INTRA	MIS	RE	UN
DE	KILO	MULTI	SEMI	UNDER
DIS	MACRO	NON	SOME	UP
DOWN				

Suffixes Removed from Words

SCRIPT/VS checks for seven types of suffixes. Each type is processed based on a set of English grammar rules to yield a root word, or stem.

Words may be processed repeatedly to remove the suffixes and yield a stem. For example, the word "conceptions" would lose two suffixes ("s" and "ion") to yield the stem "concept." The suffixes recognized by SCRIPT/VS are:

' (apostrophe)	ALLY
S	ING
ED	ION
AL	

FALLIBILITY

SCRIPT/VS spelling verification is not infallible. A misspelled word with a suffix or prefix could possibly yield a correctly spelled word after stem processing. For example, "disbooked" (with the stem "book"), and "missteak" (with the stem "steak") are both "correctly" spelled.

If a correctly spelled word has not been included in the main dictionary or the addenda dictionary, it will be identified as misspelled.

The diagnostic aids presented in this chapter are tools to help you find and correct problems caused by incorrectly specified or missing control words. Diagnostic aids in this chapter might also be useful to the Programming Service Representative (PSR). However, this chapter is directed toward the person who needs to find out why the sequence of specified GML tags and control words is not resulting in the desired output.

To use SCRIPT/VS effectively, you need to know the function of each GML tag and control word you use. Many formatting problems can result from tags and control words used incorrectly. Other chapters in this book provide detailed descriptions of each SCRIPT/VS control word. The IBM-supplied APFs that define the GML tags are documented with comments in the APFs themselves.

SCRIPT/VS diagnostic aids are provided as options of the SCRIPT command and as SCRIPT/VS control words.

DEBUGGING WITH THE SCRIPT COMMAND

Some of the SCRIPT command options are useful (for diagnostic purposes) when specified to format an input file that might contain errors.

CONTINUE: CONTINUE PROCESSING AFTER AN ERROR OCCURS

The CONTINUE option prevents SCRIPT/VS from terminating the formatting of your file unless a "severe" or "terminal" error occurs.

DUMP: ENABLE THE .ZZ [DIAGNOSTIC] CONTROL WORD

The DUMP option is useful when the .ZZ control word is included in the input file. The .ZZ [Diagnostic] control word specifies control blocks and data areas to be displayed (or printed) only when the DUMP option is specified.

DUMP has no effect unless your file contains .ZZ control words.

The DUMP option and the .ZZ [Diagnostic] control word are useful only for debugging the SCRIPT/VS program product. See the .ZZ control word description for details about data areas being dumped.

MESSAGE: CONTROL INFORMATION IN ERROR MESSAGES

The MESSAGE option allows you to specify when messages are printed, whether or not the message number is to be included, and how the line causing the error was imbedded. The MESSAGE option parameters are:

- DELAY, which accumulates all error messages in the file named DSMUTMSG. SCRIPT/VS prints the message file at the end of the formatted document or includes a message output file with the document's output file if the FILE option of the SCRIPT command is specified.

  DELAY is useful, especially when the input file is printed, because messages are normally sent to your terminal (that is, not printed).

- ID, which identifies each message with its message number. The message number can be used to refer to a more detailed description of the error.

  The message number is also needed by the PSR (Programming Service Representative) if you should need help via an APAR or the RETAIN SEARCH facility.

- TRACE, which enables the trace-back function for those messages that require it. If an error occurred within a file that is imbedded by many other files, the TRACE parameter identifies the previously imbedding files.

NOSPIE: PREVENT ENTERING SPIE EXIT ROUTINES

SCRIPT/VS ordinarily establishes a SPIE (Specified Program Interrupt Exit) before formatting begins. Subsequently, if a program check occurs, the system control program passes control to the SPIE routine, allowing SCRIPT/VS to terminate itself. The NOSPIE option inhibits this function, allowing the system control program to perform its own program check processing.

NUMBER: PRINT THE FILE NAME AND LINE NUMBER

The NUMBER option tells SCRIPT/VS to print the file name and line number of the last-read input line next to each output line. If an error occurs, you can easily locate the area where the error was detected.

PAGE: SELECTIVELY PRINT PAGES

The PAGE option allows you to print (or display) part of your formatted document, rather than requiring you to print the entire document.

SPELLCHK: VERIFY SPELLING

The SPELLCHK option enables the .SV [Spelling Verification] control word. Any word not found in the main or addenda dictionaries is listed with an error message, along with the input line that contained the word.

Because the list of spelling "errors" might be long, you should delay message printing when you specify the SPELLCHK option:

```
SPELLCHK MESSAGE (DELAY)
```

The SPELLCHK option should be used infrequently, to identify spelling errors and to identify words that should be in the addenda dictionary. You might want to print, edit, and revise several draft copies before formatting the final draft with spelling verification.

TWOPASS: PROVIDE TWO FORMATTING PASSES

The TWOPASS option allows you, in an interactive environment, to see all error messages before the formatted document is displayed. (The MESSAGE (DELAY) option displays the messages after the document is displayed.)

When you are working with a very long input file, you can format it in an interactive environment just to detect and correct any errors. With the TWOPASS option, SCRIPT/VS formats but does not display the file on the first formatting pass. All detected errors are displayed, however, before the second formatting pass starts. By inserting the .QQ [Quick Quit] control word at the end of your input file and using the TWOPASS option, you can display all detected errors and not begin the second formatting pass.

If you do not use the CONTINUE option, formatting will stop with the first error. The second pass (and actual output) will not occur unless the file is error-free.

UNFORMAT: PRINT ALL INPUT LINES WITHOUT FORMATTING

The UNFORMAT option allows you to print an input file without formatting it. All input lines (control words, GML tags, macros, symbols, and text input lines) are printed as entered. In addition, other input lines are included as a result of processing the .IM [Imbed] and .AP [Append] control words.

CONTROL WORDS TO ASSIST DEBUGGING

Some of the SCRIPT/VS control words that are useful when diagnosing problems in an input file are described below.

SPELLING VERIFICATION

The .SV [Spelling Verification] control word is used to check the spelling of words in an input file. The .SV control word is enabled by the SPELLCHK option of the SCRIPT command. See "Automatic Hyphenation and Spelling Verification" for details about verifying spelling.

DISPLAYING THE SEQUENCE OF SCRIPT/VS PROCESSING

One of the most powerful SCRIPT/VS control words is the .IT (Input Substitution Trace) control word. This allows you to see the steps taken by SCRIPT/VS when it substitutes a value for a symbol name. You can also see the step-by-step processing of the control words that make up a macro or GML tag's APF. The .IT control word has many other capabilities that allow you to trace specific events during SCRIPT/VS processing.

The Output Line Generated by Input Tracing

When input tracing is activated, SCRIPT/VS generates one or more output lines that describe the sequence of processing required for the input line about to be executed. These lines are displayed as though they were messages: they are written to the same output destination as messages. Each generated output line is in the form:

```
*◆* [name] [nn]   x  <source line>
```

where:

◆ a code that identifies why the "current source line" is being traced:

C: Control word trace

M: Macro substitution trace

S: Symbol substitution trace

name identifies the name of the "current source line." This is usually the name of the file or macro currently being processed. If the name is in parentheses, for example, (.RT n), (RHEAD), or (RFOOT), the current source line comes from a previously saved running title (.RT n), running footing (RFOOT), or running heading (RHEAD), and not from the file or macro currently being processed.

nn the line number of the "current source line," either within a file or within a macro.

x the length (number of characters and blanks) in the "current source line. "

current source line the line being traced by SCRIPT/VS. The following description assumes that all traceable events are being traced: control word tracing, symbol substitution tracing, and macro substitution tracing (as specified with .IT ALL):

- When the current source line contains only text, it is not displayed as part of the input trace.
- When the current source line contains a control word (*C*), SCRIPT/VS displays the current source line and then performs the control word function. However, if the STEP parameter of .IT is specified, you can change a "control word" current source line 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 one or more symbols (*S*), SCRIPT/VS:
  - Displays the line as it is (*S*) before any symbols are substituted.
  - Displays the line repeatedly, each time showing the next stage of substitution, until each defined and null-valued symbol has been replaced with its value (*S*). Undefined symbol names are regarded as text.
  - At this point, the line is processed as a line of text, or is traced as a control word current source line (*C*) (as described above).
- When the current source line is from a macro expansion (*M*), SCRIPT/VS:
  - Displays the line as it exists in the macro (*M*).
  - If the line contains one or more symbols, SCRIPT/VS traces the line as described above for symbol substitution tracing.
  - At this point, the line is processed as a line of text, or is traced as a control word (*C*) as described above.

Capabilities of the .IT Control Word

The above description made assumptions that allowed a simplified presentation of input substitution tracing. However, the .IT (Input Substitution 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 not other traceable events) specify:

  ```
  .it sub
  ```

- When you want to trace only macro expansions (and not trace other traceable events) specify:

  ```
  .it mac
  ```

  Symbols that are part of the macro expansion are traced. However, symbols that are not part of a macro expansion will not be traced.

- When you want to trace occurrences of control words that interest you, specify:

  ```
  .it ctl .xx .yy .zz
  ```

  For example, to trace each occurrence of the .IN [Indent], .IL [Indent Line], and .OF [Offset] control words, specify:

  ```
  .it ctl .in .il .of
  ```

  The .IN, .IL, and .OF control words are added to the list of control words currently being traced, called the "control word table."

  When you want to stop tracing for control words, but want to continue the input trace for other kinds of input items previously specified, issue

  ```
  .it ctl
  ```

  The CTL parameter of the .IT control word clears the list of control words being traced.

- When you want to stop tracing control words, but leave the control word table intact for tracing later, issue

  ```
  .it off
  ```

- When you want to turn off all input tracing, specify:

  ```
  .it off
  ```

  As noted above, the OFF parameter stops tracing, but does not clear the control word table. When you want to resume tracing the control words currently in the table, issue:

  ```
  .it on
  ```

  To add control words to the control word table, issue:

  ```
  .it ctl .xx .yy
  ```

- When you want to display the current value of a macro or symbol, specify the SNAP parameter of the .IT control word. For example, if you want to find out the current definition of the @LIST macro specify:

  ```
  .it snap @LIST
  ```

  The current definition of any symbol, as well as any macro by that name, is displayed only once, not continuously. The SNAP parameter does not affect other parameters of the .IT control word, and can be specified even when input tracing is turned off.

Stepping through an Input Trace

The discussion so far assumed that merely displaying the sequence of SCRIPT/VS operations is sufficient for diagnostic purposes. However, SCRIPT/VS allows you to "step through" the lines being traced in an interactive fashion. Each line of the trace is displayed. When the "current source line" contains a control word (*C*), it is not executed immediately after display. SCRIPT/VS displays the line and waits for your response from the terminal. You specify the "step through" function with:

```
.it step
```

Other input trace functions remain in effect. When a traceable event displays a control-word current source line, SCRIPT/VS displays the line and waits for your response. Therefore, the STEP function cannot occur when you format the file with the MESSAGE (DELAY) option specified. The traced lines are displayed as

though they were messages, and you cannot respond to a "delayed" message.

The procedure performed for step-by-step control word tracing is:

1. Display the control-word current source line (*C*) at the message destination (that is, your terminal).
2. Wait for your response.
3. Process the response, which might result in:
   a. Executing the traced control-word current source line, or
   b. Displaying a new current source line to be processed before, after, or instead of the traced control-word current source line.
   c. Identifying the function currently reading input from the terminal.

The responses that you can provide interactively are:

- Null (press the ENTER or RETURN key): the traced control-word current source line is executed and SCRIPT/VS continues processing until it encounters the next traceable control-word line.
- STK input-line: means "stack this input line." The traced control word line is executed. The stacked input line is put on a stack and becomes the next control word to be executed (or, if it is traceable, traced before it is executed) after the currently traced control word line completes its execution.

  For most control words, the traced control word executes and then the stacked input-line control word executes. However, if the traced control word is .IM [Imbed], traceable control word lines from the imbedded file are traced and executed before the stacked input line executes. The control words .TE [Terminal Input] and .AP [Append] prevent the immediate sequence of the input line in a similar way.
- PRE input-line: executes the input line before the traced line. The traced current source line is put on a stack. The PRE input line becomes the current source line. If the PRE input line is not a traceable control word, SCRIPT/VS executes it and continues processing until it encounters the next traceable control word line.

  When the PRE input line is a traceable control word line, it is displayed and SCRIPT/VS waits for your response. Your response can be any response allowed for a traced line.
- REP input-line: replaces the traced line with the input line. The traced input line is not executed nor is it put on a stack. Instead, the REP input line becomes the current source line. If the REP input line is not a traceable control word line, SCRIPT/VS executes it and continues processing until it encounters the next traceable control word line.

  If the REP input line is a traceable control word line, it is traced before execution (like any other traceable control word line). You can now enter any response allowed for a traced line.
- Data line: "PRE input-line" is assumed. SCRIPT/VS processes the data line as described above for PRE.
- ?: does not affect the input trace line. SCRIPT/VS identifies the "reader" of terminal input: either "TERMINAL INPUT" when the .TE [Terminal Input] function is expecting your input, or "CONTROL TRACE" when the .IT STEP function is expecting your input.

Note: To get out of step-by-step input tracing, enter a STK, PRE, or REP with one of the following as a data line:

".it off", to turn off all input substitution tracing,

or

".it run", to resume normal input substitution tracing (that is, to stop the STEP function),

or

".qq", to terminate the SCRIPT/VS formatting job immediately.

Using Terminal Entry to Test a Control Word Sequence

A useful tool for testing SCRIPT/VS control word sequences is the two-line input file (user-created) called TEST:

```
.ty Enter SCRIPT input:
.te on
```

When you process the file with the SCRIPT command in an interactive environment,

```
SCRIPT TEST (CONTINUE NOPROFILE)
```

you get the message

```
Enter SCRIPT input:
```

which resulted from the .TY control word. You can enter control words, macros, symbols, GML tags, and text. Each terminal-entered input line is processed immediately by SCRIPT/VS and is used to build a page. When the page is full (or when a page eject occurs), SCRIPT/VS displays the completely formatted page before accepting additional input lines from the terminal.

To end processing and exit to your interactive environment (CMS or TSO), you can enter:

.QQ [Quick Quit] to end all formatting immediately.

.EF [End of File] to end formatting and close the terminal input file.

.TE [Terminal Input] OFF, to turn off the previous .TE ON and (within the context of the TEST file) end formatting.

.QU [Quit] to display the current output page and then end formatting.

The .QQ [Quick Quit] control word ends processing immediately without the final output page being displayed; you will not see the data that has been formatted for the final output section but not yet displayed.

For testing and diagnosing macros and control word sequences, the first input line you enter might be:

```
.it all
```

All subsequent traceable input lines are traced.

When the page eject occurs (you can force a page eject with the .PA [Page Eject] control word), SCRIPT/VS displays the formatted output accumulated so far. You can then resume terminal input.

Be sure to write down whatever you want to save for future use. The "input file" during terminal input is your terminal keyboard. What you enter is not saved in a disk file. You can create a disk file with input line sequences you want to repeat, and then imbed the file from the terminal whenever you want it, by typing

```
.im filename
```

PUTTING MESSAGES IN MACROS

When you build a macro (or an APF), you can use the .IF control word to detect errors in input or syntax. The .MG control word allows you to notify the user that an error occurred. Your error message should include a message number and a brief, clearly-written description.

DISPLAYING CONTROL BLOCKS

One of the diagnostic functions SCRIPT/VS provides is the ability to display control blocks and data areas at specified points in the input file. The .ZZ control word identifies the data to be displayed. The DUMP option of the SCRIPT command enables the .ZZ control word. For further details about SCRIPT/VS internal data areas, see the .ZZ control word description.

EasySCRIPT is an early implementation of GML that existed in SCRIPT/370. Before deciding to use EasySCRIPT, you should review the current SCRIPT/VS GML, which is described in the Document Composition Facility: Generalized Markup Language (GML) User's Guide -- particularly, the appendix on "Getting Started with GML."

EasySCRIPT functions are built into the formatter. You don't use any profile or symbol and macro library with EasySCRIPT. EasySCRIPT is designed to be easy to use, but not flexible. Since the EasySCRIPT functions are built in, you can't tailor them to your own installation's requirements, as you can with SCRIPT/VS GML.

EasySCRIPT provides formatting shortcuts that take advantage of SCRIPT/VS to offer a simple way to format many documents. EasySCRIPT tags can be freely intermixed with standard SCRIPT/VS control words. Using these shortcuts, you can:

1. Produce numbered, unnumbered, or bulleted lists automatically.

2. Automatically format headings and a table of contents. And, if you want, you can have EasySCRIPT number your headings using a decimal numbering system. Then, when you add or delete information, the numbering is changed for you.

3. Format text in paragraphs aligned with the current indention level of a list or heading section.

The built-in EasySCRIPT functions can be invoked in either of two ways:

1. As parameters of the .EZ control word. For example, to get the "B" EasySCRIPT function, which formats a bulleted item, you could enter

   ```
   .ez B text of the bulleted item.
   ```

2. As EasySCRIPT "tags." One of the functions of EasySCRIPT is to define a series of symbols that act as tags to substitute the appropriate .EZ control word. These are not true GML tags in the sense that they are delimited with the symbol delimiter (&), not the GML delimiter (:). The reason for this is that EasySCRIPT tags have different meaning if entered in uppercase than if entered in lowercase. GML tags are not sensitive to the case in which they are entered. The control word

   ```
   .ez on
   ```

   enables the EasySCRIPT tags. Each EasySCRIPT tag has the same name as the equivalent parameter of the .EZ control word. The EasySCRIPT tags are included in SCRIPT/VS to allow documents already marked up with them to be processed by SCRIPT/VS.

You can use the EasySCRIPT functions in your own symbols and macros.

EASYSCRIPT TAGS

There are five EasySCRIPT tags. Each tag provides two different sets of functions depending upon whether it is capitalized or not. The rule is that the capitalized version provides more function.

The five basic tags are:

1. &Hx -- Inserts a decimal numbered heading of level x where x is 1, 2, 3, 4, 5, or 6.

To create documents without the decimal heading numbers, type the "h" in the heading tag in lowercase.

2. &P -- Starts a new major paragraph. A major paragraph resets the indention to zero and produces the necessary spacing.

   To maintain the current indention for a minor paragraph (that is, within a list,) type the paragraph tag with a lowercase "p".

3. &Nx -- Inserts a numbered item of level x where x is 1, 2, 3, or 4.

   If you do not want items numbered, enter the tag with a lowercase "n". A list is itemized at the level of indention associated with the number in the tag (level 1, 2, 3, or 4).

4. &B -- Inserts a bulleted item (one that begins with a •) under the current paragraph or numbered item.

   Sub-bullets (items that are introduced with hyphens) may be entered under bulleted items by typing the bullet tag with a lowercase "b".

5. &toc -- Generates a table of contents.

As you can see, all five EasySCRIPT tags begin with an ampersand (&). A tag may be connected to the line that follows with a period or with one or more blanks:

```
&TAG.line
```

is the same as:

```
&TAG line
```

EASYSCRIPT FORMATS

The EasySCRIPT tags for numbered headings, lists, and paragraphs keep track of the current number of an item and the level of indention. It is good practice, if you are using EasySCRIPT, to use it consistently throughout a document. If you duplicate the function of EasySCRIPT, for example, by manually numbering an item, you will lose the benefit of having the other items numbered automatically.

There is no problem, of course, using any SCRIPT/VS control words that do not duplicate the functions of EasySCRIPT.

HEADINGS

Within the text, headings are automatically numbered (when requested) and formatted by EasySCRIPT, regardless of whether you enter them with uppercase or lowercase characters. However, headings placed in the table of contents by EasySCRIPT appear with the number (if requested) and in the same case as they were entered in the input file.

The numbering scheme used by EasySCRIPT when you invoke the uppercase heading tags is a decimal system: headings may be numbered 1.0, 1.1, 1.1.1, 1.2, 1.2.1, 1.2.2, 1.2.3, and so on.

For documents not requiring decimal numbering, enter the heading tag in lowercase (&h1 through &h6). Decimal-numbered headings and non-decimal numbered headings may be mixed.

When headings are processed, all indentions (from numbered items and bullets, for instance) are reinitialized and all numbered item counters are reset.

A variation of the &Hn. tag is &An, which may be used to automatically number appendixes with letter prefixes such as A.0, B.1.1, C.2.1.3, and so on.

SETTING THE HEADING COUNTER

If you need to manually control the number of a particular heading (for example, if you turn EasySCRIPT off and then want to turn it back on again), you can specify the number of the last heading on the .EZ control word:

```
.ez on &xref
```

The symbol &xref is the counter used by EasySCRIPT to keep track of the numbers it is using.

To set a number explicitly for EasySCRIPT to use as the last heading number, you can enter:

```
.ez on 3.0
```

After this control word is processed, the next level two heading (&H2) will be numbered 3.1, the next level one heading will be numbered 4.0, and so on. If you do not specify a number or &xref, then the last heading is considered to have been 0.0.

EASYSCRIPT HEADING DEFAULTS

The default characteristics for headings associated with EasySCRIPT vary from those used by the .H0 - .H6 [Head Level 0 - 6] control words. If you enter the .EZ ON control word, these values are in effect for .H1 through .H6. When you enter .EZ OFF, the normal values are restored.

Figure 31 on page 314 gives the default characteristics of headings used in EasySCRIPT. Remember that if the heading tag is entered in uppercase, the heading is assigned a number; if entered in lowercase, it is not numbered. Otherwise, the characteristics are the same.

You can change the default characteristics of EasySCRIPT heading tags with the .DH [Define Head Level] control word. The change affects only the EasySCRIPT head level, or only the non-EasySCRIPT head level, whichever is currently in effect.

CROSS-REFERENCES TO EASYSCRIPT HEADINGS

EasySCRIPT has a cross-reference feature you can use to refer to the heading numbers that are generated by EasySCRIPT. The symbol "&xref" is the counter used by EasySCRIPT to keep track of the heading level. If you set another symbol using this symbol, for example,

```
.se intro = &xref
```

Then you can refer to the symbol &intro in your text:

```
Introductory material is in section &intro..
```

When SCRIPT/VS substitutes this line, the result may be something like:

```
Introductory material is in section 3.1.
```

EXAMPLES OF EASYSCRIPT FORMATTING

The following shows how you might enter EasySCRIPT tags to control the formatting of a document.

PARAGRAPHS

A paragraph is designated when the first three characters of a line are either "&P." or "&p.". There should be either a period or at least one blank before the first character of the paragraph text. The EasySCRIPT paragraph tags insert a blank line between paragraphs.

The paragraph above is entered as follows:

```
&P A paragraph is designated...
There should be no space between...
The EasySCRIPT paragraph tags...
```

If the paragraph tag is capitalized ("&P"), a major paragraph is indicated; this resets enumeration counts and returns the indention to zero. Major paragraphs are used to break out of a series of numbered items.

If the paragraph tag is not capitalized ("&p"), a minor paragraph is indicated. You should use minor paragraph tags within numbered items because a minor paragraph tag does not reset the indention or list item counter.

AUTOMATIC ITEM NUMBERING

Up to four levels of items can be numbered or lettered. The numbering range is from 1-99 and the lettering range is from a-z. The use of any level of numbered items reinitializes item counts of deeper levels.

An item at the first level of indention is formed when a line begins with "&N1". Each successive use of "&N1" results in a blank line to separate items, the next higher item number, and its indented text.

The second level of numbered items results from using "&N2" in a similar way to "&N1". In like manner, the remaining levels are obtained by using "&N3" and "&N4". Following is an example of the numbered and indented, and bulleted items:

- Here is a bulleted item at level one. (A bulleted item is formatted at the current indention.)

3. This is item one of a first level numbered list.

4. This is item two of a first level numbered list.

   a. This is item one of a second level numbered list.

   b. This is another item of a second level numbered list.

      This is a minor paragraph placed underneath a level two numbered item to illustrate how the indention is maintained.

      - We can put bulleted items under any level of indention.

        - Sub-bullets, too.

UNNUMBERED LISTS

Unnumbered lists can be formatted using the &n1 through &n4 tags. Following are some examples of unnumbered lists:

This is item one of a level one unnumbered list.

This is item two of a level one unnumbered list.

This is item one of a level two list.

BULLETS

Bullets and sub-bullets can be used instead of numbers and letters for indented items. The format of the EasySCRIPT bulleting tags is:

```
&B.Text of bulleted item.
&b.Text of sub-bulleted item.
```

Bullets and sub-bullets may be used beneath any level of indention (see examples above).

TABLES OF CONTENTS

A table of contents is automatically generated (with any calculated decimal numbering) and inserted at the location of the "&toc" tag (similar to the .TC [Table of Contents] control word). All you need to enter is

```
&toc
```

and SCRIPT/VS formats and prints the table of contents.

CHAPTER 18. COMPATIBILITY WITH SCRIPT/370

This chapter is provided for the convenience of users of SCRIPT/370 Version 1 (Program Number 5796-PAF) and SCRIPT/370 Version 3 (Program Number 5796-PHL).

Figure 13 shows SCRIPT command compatibility (that is, the similarities and differences between the options for SCRIPT/VS and for SCRIPT/370).

SCRIPT/370 control words that are changed are listed in Figure 14 on page 182. (SCRIPT/370 control words that are obsolete are listed separately, in Figure 16 on page 186, along with the equivalent SCRIPT/VS control word.) Control words and options with identical meaning in SCRIPT/370 and SCRIPT/VS are not listed. Control words that are new in SCRIPT/VS or SCRIPT/370 are listed in Figure 15 on page 186.

The following codes are used:

New/3 The control word or option was changed for SCRIPT/370 Version 3.

New/VS The control word, feature, or option was changed for SCRIPT/VS.

Invalid The option is no longer valid. Its function is performed by a new option in SCRIPT/VS. SCRIPT/VS does not accept or process the old option.

The listed changes are cumulative. That is, SCRIPT/VS incorporates and includes changes introduced in SCRIPT/370 Version 3 and earlier versions of SCRIPT/370. For details on the functioning of individual SCRIPT/VS control words and SCRIPT command options, see the appropriate chapter in this book.

CHANGES TO THE SCRIPT COMMAND

Figure 13 shows the SCRIPT command options that are different (from previous versions of SCRIPT/370) for SCRIPT/VS.

Option	Code	Changes
2PASS	Invalid	Specify TWOPASS instead.
ADJUST	Invalid	Specify BIND instead.
ADJUSTnn	Invalid	Specify BIND instead.
BIND	New/VS	Shift the page image to the right.
CENTER	Invalid	Specify BIND instead.
CENTERnn	Invalid	Specify BIND instead.
CHARS	New/VS	Specify up to four fonts (Valid for the 3800 Printer only.)
DEBUG	Invalid	Specify NOSPIE instead.
DEVICE	New/VS	Specify a logical output device.
DUMP	New/VS	Enable the .ZZ [Diagnostic] control word to snap SCRIPT/VS control blocks.
LIB	New/VS	You can specify up to eight library names.

Figure 13. SCRIPT/370 Command Option Compatibility (Part 1 of 2)

Option	Code	Changes
MARK	Invalid	
MESSAGE	New/VS	Control the timing and destination of messages.
NOPROF	New/3	Supress the PROFILE option.
NUMBERnn	Invalid	Specify NUMBER instead.
OFFLINE	Invalid	Specify PRINT instead.
OPTIONS	New/VS	Specify a file that contains additional SCRIPT Command options.
PAGE	New/VS	You can specify one or more ranges of pages to print, or that you want SCRIPT/VS to prompt you to enter page number ranges.
PAGEnnn	Invalid	Specify PAGE (nnn) to print from page nnn.
PROFILE	New/VS	You can specify the name of a profile.
SEARCH	New/VS	Specify a library to be searched for imbedded files. (Not valid in CMS.)
SINGLE	Invalid	Specify PAGE (nn ONLY) to print a single page, nn.
SPELLCHK	New/VS	Enable the .SV [Spelling Verification] control word to perform spelling verification.
SYSVAR	New/3	Set Symbol values from the Command Options.
TERM	New/3	Display formatted output at the terminal.
TRANSLATE	Invalid	Specify UPCASE to have lowercase translated to uppercase.
UNFORMAT	New/3	Processes the .IM [Imbed], .AP [Append], and .EF [End of File] control words, and reads lines from imbedded files to include in the unformatted listing. Symbol substitution is performed, but the input line is printed as entered.

Figure 13. SCRIPT/370 Command Option Compatibility (Part 2 of 2)

CHANGES TO SCRIPT/370 CONTROL WORDS

Figure 14 on page 182 shows the changes made to the SCRIPT/370 control words to enhance them for SCRIPT/VS. Other differences between SCRIPT/370 and SCRIPT/VS control words are:

- SCRIPT/VS accepts only the two-character form of control words. SCRIPT/370 allowed you to specify either the two-character name or the control word's long descriptive name.
- SCRIPT/VS sets the symbols &0 through &9 only when the .IM [Imbed] and .AP [Append] control words are processed. SCRIPT/370 also set the same symbols when a macro was invoked. (Actually, in SCRIPT/VS, the number of tokens available with .IM [Imbed] and .AP [Append] has been increased from 9 to 14.)
- SCRIPT/VS sets the symbols &*0 through &*n when a macro is invoked. You should convert all your SCRIPT/370 macros to the SCRIPT/VS form, since the SCRIPT/VS macro processor is much more powerful than the SCRIPT/370 macro processor. In the meantime, you can cause SCRIPT/VS to transfer the macro

parameters in &*0 through &*9 to the symbols &0 through &9 to allow unmodified SCRIPT/370 macros to operate correctly. To do this, you must provide a .DM macro that will process all SCRIPT/370 .DM [Define Macro] control words. The macro can be included in your PROFILE file. You also need a dummy file that will be imbedded by the .DM macro. The dummy file can contain a single comment line. The macro can be defined with the following:

```
.su off
.**********                    **********
.* .dm macro sets tokens &0 - &9:       *
.* (requires dummy file be present)     *
.**********                    **********
.'dm dm off
.'dm dm() /.ec .sa
.'dm dm() /.ec .su off
.'dm dm() /.ec .se *a=index &V'&* &V'&*2
.'dm dm() /.ec .se *b=substr &V'&* &*a
.'dm dm() /.ec .se *s=substr &V'&*2 1 1
.'dm dm() /.ec .su on
.'dm dm() /.ec .dm &*1 &*s..ec .im dummy &*&V'&*b
.'dm dm() /.ec .re
.*
.su on
```

- For .SE [Set Symbol] control words that set a symbol to the current page number the form ".se name = &", SCRIPT/VS sets the symbol to the current page number, including its prefix, if any, in its character string form. You should be careful not to use the page number in arithmetic expressions when a non-numeric prefix would cause an error.

- SCRIPT/VS accepts space units when you specify a control word's parameter that defines a horizontal or vertical space or displacement. With SCRIPT/370, the amount of space could be specified only as a number of characters or lines. See "Specifying Vertical and Horizontal Space Units" in "Chapter 1. An Introduction to SCRIPT/VS" on page 1 for details about space units.

  Exceptions to this are the .HS [Heading Space] and .FS [Footing Space] control words, which specify the number of lines available for top and bottom titles.

- SCRIPT/VS maintains a page's layout parameters until it completes formatting the page. Control words that affect a page's layout always take effect on the next page.

  With SCRIPT/370, a control word that affected the page layout (for example, the .PL [Page Length] control word, which changed the number of lines on a page) would take effect on the current page if possible. Otherwise, it would take effect on the next page. The resulting output was sometimes difficult to predict and plan for.

  With SCRIPT/VS, you can establish the next page's layout and then eject to that page. The current page is not affected by the new page layout parameters.

  Control words that always take effect on the next page are:

  .BM [Bottom Margin]
  .BT [Bottom Title]
  .EB [Even Page Bottom Title]
  .ET [Even Page Top Title]
  .FM [Footing Margin]
  .FS [Footing Space]
  .HM [Heading Margin]
  .HS [Heading Space]
  .LL [Line Length]
  .OB [Odd Page Bottom Title]
  .OT [Odd Page Top Title]
  .PL [Page Length]
  .PN [Page Numbering Mode]

```
.RF [Running Footing]
.RH [Running Heading]
.RT [Running Title]
.TM [Top Margin]
.TT [Top Title]
```

When SCRIPT/VS begins to process your input file, the first page has not yet been started. Formatting for the first output page begins when there is text for it, or when a control word (for example, .SK [Skip]) that requires the page to be started is processed.

You can specify all of the above dimensions for page one when you put their control words before formatting begins for page one, while page one is still the "next page" to be started.

A new chapter commonly begins with a .PA [Page Eject] control word. In SCRIPT/370, some page layout control words were usually placed before the .PA, and others after it. Assuming the page layout (including new top- and bottom-titles) is to take effect for the new chapter, the proper sequence in SCRIPT/VS is to place all of these control words before the .PA control word. You can include the following .PA macro definition in your PROFILE file to allow the SCRIPT/370 sequence to operate properly:

```
.su off
.************                          ************
.* .pa macro synchronizes page ejects:    *
.************                          ************
.'dm pa off
.'dm pa() /.if &e'&*1 = 1 .'pn &*1
.'dm pa() /.ec .pa nostart
.*
.su on
```

- The control words .UC [Underscore and Capitalize], .UP [Uppercase], and .US [Underscore] have been changed in SCRIPT/VS to accept the parameters ON, OFF, or a number. In SCRIPT/370, these control words accepted only a single line of text. In SCRIPT/370, the control word ".US 80" would underscore the line consisting of the characters "80", but in SCRIPT/VS the same control word will underscore the next 80 lines. The following macros can be included in your PROFILE file to make these three control words operate as the SCRIPT/370 equivalents:

```
.************                          ************
.*  make .up, .us, .uc act as in SCRIPT/3 *
.************                          ************
.'dm up /.'uc off/:'us off/.'up 1/.'li &*
.'dm us /.'up off/.'uc off/.'us 1/.'li &*
.'dm uc /.'up off/.'us off/.'uc 1/.'li &*
.*
```

- Certain page layout parameters, whose values were constant in SCRIPT/370, are based on the logical device type in SCRIPT/VS. These parameters and the control words that affect them are:

```
Top margin (.TM)
Bottom margin (.BM)
Footing margin (.FM)
Footing space (.FS)
Heading margin (.HM)
Heading Space (.HS)
Line length (.LL)
Page length (.PL)
```

THE SCRIPT/370 DICTIONARY

SCRIPT/370 supported a hyphenation exception dictionary called SCRIPT XDICT. The XDICT dictionary was used to determine how to hyphenate words that were not correctly hyphenated by the hyphenation algorithm. The user could create and modify his own hyphenation exception dictionaries using the HYPEDIT command.

SCRIPT/VS does not support either the HYPEDIT command or exception dictionaries. Instead, SCRIPT/VS provides a comprehensive dictionary that supports both spelling verification and automatic hyphenation.

You can also create and update a temporary dictionary for use when your document is being formatted, called the addenda dictionary, using the .DU [Dictionary Update] control word.

Control Word	Code	Description
...	New/VS	Allowed in SCRIPT/VS macros.
.AP	New/VS	Up to 14 tokens can be passed to the appended file. The tokens are not reset when a macro is called. &0 contains the number of tokens passed.
.BC	New/3	Operands ON and OFF restore and cancel column balancing.
.BM	New/3	The bottom margin can be specified as an increment to or a decrement from the current value. Default based on logical device. Control word always takes effect on the next page.
.BX	New/3	Draw automatic boxes. New options for drawing a box within a box, for drawing fragments of boxes, and for drawing parallel boxes.
.CC	New/VS	Ejects to a new column only when not already at the top of a column
.CD	New/3	You can define up to nine displacements for columns, even if you initially specify only one column The remaining displacements are used when you later increase the number of columns.
.CE	New/3	Accepts input text as a parameter.
.CL	New/3	The column width can be specified as an increment to or a decrement from the current value.
.CO	New/3	Operands ON and OFF restore and cancel concatenation.
.CP	New/VS	Ejects to a new page only when not already at the top of a page.
.DH	New/VS	You can specify a font for each head level.
.DM	New/VS	You can specify a macro with more than one input line, and store macros in a library. When a macro is invoked, SCRIPT/VS sets local symbols &*0 through &*n (n being the number of tokens passed to the macro; &*0 contains the value n). The macro can set local symbols that begin with &*.
.EF	New/3	CLOSE operand.
.EZ	New/VS	Allows EasySCRIPT tags and control words to be more freely mixed.
.FM	New/3	The footing margin can be specified as an increment to or a decrement from the current value. Default is based on the logical device. Control word always takes effect on the next page.
.FN	New/VS	New parameter: LEADER, allows you to define leading text lines to precede the first footnote of each page. Column balancing occurs on pages with footnotes. Footnotes are formatted to the line length instead of to the column width.

Figure 14. Changes to SCRIPT/370 Control Words (Part 1 of 4)

Control Word	Code	Description
.FO	New/3	Operands ON and OFF allow you to restore and cancel formatting (concatenation and justification). New parameters, LEFT, RIGHT, CENTER, EXTEND, FOLD, and TRUNC.
.FS	New/3	The footing space can now be specified as an increment to or a decrement from the current value. Control word always takes effect on the next page.
.GO	New/VS	Allowed in SCRIPT/VS macros.
.Hn	New/3	Head-level control word for head levels 0 through 6. The default characteristics of each head level are changed when you specify the .EZ ON (Enable EASYSCRIPT) control word, and are restored when you disable EASYSCRIPT (with the .EZ OFF control word).
.HM	New/3	The heading margin can be specified as an increment to or a decrement from the current value. Default is based on the logical device. The control word always takes effect on the next page.
.HS	New/3	The heading space can be specified as an increment to or decrement from the current value. The control word always takes effect on the next page.
.HY	New/VS	Hyphenation is performed using a dictionary.
.IM	New/VS	Up to 14 tokens can be passed to the imbedded file. The tokens are not reset when a macro is called. &0 contains the number of tokens passed.
.IN	New/3	An indent can be specified as an increment to or a decrement from the current value.
.JU	New/3	Operands ON and OFF restore and suspend concatenation.
.KP	New/VS	New parameter: INLINE, allows you to keep text with preceding and following text. You can specify .KP n, to keep the following n lines together as an INLINE keep. The ON parameter causes a break (INLINE doesn't cause a break). When a keep is part of a column, the column remains eligible for balancing. When the column is balanced, the keep is treated as a single entity and is not divided.
.LI	New/3	Accepts a data line as a parameter. Operands ON and OFF establish and suspend literal interpretation.
.LL	New/3	The line length can be specified as an increment to or a decrement from the current value. Default is based on the logical device. The control word always takes effect on the next page. When the column width is changed (implicitly) by .LL, the change is reflected in the current page's layout.
.OF	New/3	An offset can be specified as an increment to or a decrement from the current value. Any new .OF [Offset] control word resets the previous offset value.

Figure 14. Changes to SCRIPT/370 Control Words (Part 2 of 4)

Control Word	Code	Description
.PA	New/VS	New parameter: NOSTART, ends the current page without starting the new page. You may then modify the page layout, headings, footings, etc. The next page is not started until SCRIPT/VS encounters either a control word that requires it or input text. New parameters: ODD, EVEN, ON, and OFF, allow you to page eject when an odd or even page is encountered.
.PL	New/3	The page length can be specified as an increment to or a decrement from the current value. Default is based on logical device. The control word always takes effect on the next page.
.PN	New/3	New parameter: FRAC, initiates fractional page numbering (decimal point pages). New parameter: NORM, restores normal (ascending integer) page numbering and causes a page eject. New parameter: PREF, specifies a character string prefix for all page numbers. New parameter: ALPH, allows you to specify alphabetic page numbering. New parameter: n, allows you to reset the page number. The control word always takes effect on the next page.
.PT	New/VS	The document's table of contents is generated as a separate file, named DSMUTTOC, whose format can be influenced by the user.
.RI	New/3	Accepts a data line as a parameter.
.RV	New/VS	A user can type in characters without having to enclose them in quotes. You can specify a file as the terminal input file by using the .DD [Define Data File-id] control word.
.SE	New/3	The OFF operand cancels a symbol value. You use two single quote marks to achieve a single quote within a symbol value. New parameter: INDEX, allows you to locate one character string within another. New parameter: SUBSTR, allows you to use part of a character string. New parameter: LIB, specifies that a library contains the symbol' s value.
.SK	New/3	New parameter: A, specifies absolute spacing. New parameter: C, specifies conditional spacing. New parameter: P, specifies page-width skips. If a conditional skip is followed by another .SK or .SP, the longer of the two is used, not the second (as in SCRIPT/370). .SL governs the size of skip requests expressed in lines, even if A is specified. .SK requests expressed in other space units are not affected by the setting of .SL.
.SP	New/3	New parameter: A, specifies absolute spacing. New parameter: C, specifies conditional spacing. New parameter: P, specifies page-width space. If a conditional space is followed by another .SP or .SK, the longer of the two is used, not the second (as in SCRIPT/370). .SL governs the size of space requests expressed in lines, even if A is specified. .SP requests expressed in other space units are not affected by the setting of .SL.

Figure 14. Changes to SCRIPT/370 Control Words (Part 3 of 4)

Control Word	Code	Description
.SU	New/3	Initial value is ON.
.TB	New/VS	You cannot specify more than 16 tab positions.
.TC	New/VS	The table of contents file, DSMUTTOC, is built and formatted by SCRIPT/VS as a normal input file. Rules for n and input line parameters have been changed.
.TM	New/3	The top margin can be specified as an increment to or decrement from the current value. Default is based on the logical device. The control word always takes effect on the next page.
.UC	New/VS	New parameters: ON and OFF, allow you to capitalize and underscore large blocks of text.
.UD	New/VS	The "required blank" is normally not underscored. The required blank defaults to hexadecimal 41.
.UN	New/3	An undent can be specified as an increment to or decrement from the current value.
.UP	New/VS	New parameters: ON and OFF, allow you to capitalize large blocks of text.
.US	New/VS	New parameters: ON and OFF, allow you to underscore large blocks of text.

Figure 14. Changes to SCRIPT/370 Control Words (Part 4 of 4)

New Control Words in SCRIPT/370 Version 3	New Control Words in SCRIPT/VS
.DH [Define Head Level]	.BF [Begin Font]
.DM [Define Macro]	.DC [Define Character]
.FN [Footnote]	.DD [Define Data File-id]
.GO [Goto]	.DU [Dictionary Update]
.HW [Hyphenate Word]	.EC [Execute Control]
.HY [Hyphenate]	.EM [Execute Macro]
.IF [If]	.IT [Input Trace]
.IL [Indent Line]	.IR [Indent Right]
.KP [Keep]	.LB [Leading Blank]
.MC [Multicolumn Mode]	.LT [Leading Tab]
.MS [Macro Substitution]	.LY [Library]
.PP [Paragraph Start]	.MG [Message]
.PT [Put Table of Contents]	.NL [Null Line]
.QQ [Quick Quit]	.OC [Output Comment]
.RV [Read Variable]	.PF [Previous Font]
.SC [Single Column Mode]	.RF [Running Footing]
.SK [Skip]	.RH [Running Heading]
.SY [System Command]	.RT [Running Title]
.TC [Table of Contents]	.SF [Save Font]
.UC [Underscore and Capitalize]	.SL [Set Line Space]
.UD [Underscore Definition]	.SV [Spelling Verification]
.UP [Uppercase]	.SX [Split Text]
.US [Underscore]	.TI [Translate Input]
	.WF [Write To File]
	.ZZ [Diagnostic]

Figure 15. New Control Words: These control words were introduced into the SCRIPT control word set in the indicated release.

Obsolete Control Word	SCRIPT/VS Equivalent Control Word
.BT	.RT [Running Title] BOTTOM
.EB	.RT [Running Title] BOTTOM EVEN
.EP	.PA [Page Eject] EVEN
.ET	.RT [Running Title] TOP EVEN
.FI	.FO [Format Mode] ON
.FT	.RT [Running Title] BOTTOM
.HE	.RT [Running Title] TOP EVEN
.HN	.RH [Running Heading]
.LS	.SL [Set Line Space]
.NB	.BC [Balance Columns] OFF
.NC	.CO [Concatenate Mode] OFF
.NF	.FO [Format Mode] OFF
.NJ	.JU [Justify Mode] OFF
.OB	.RT [Running Title] BOTTOM ODD
.OP	.PA [Page Eject] ODD
.TT	.RT [Running Title] TOP

Figure 16. Obsolete Control Words: SCRIPT/VS continues to recognize and support these control words, but their functions have been subsumed by more general control words as indicated.

CONVERTING ATMS-II DOCUMENTS TO SCRIPT/VS FORMAT

ATMS-II to SCRIPT/VS conversion encompasses three separate functions.

- The conversion processor, which runs as an attribute processor under the control of the SCRIPT/VS Document Library Facility. This processor scans ATMS-II documents for ATMS-II formatting controls and substitutes SCRIPT/VS symbols or macros.
- The profile, used when invoking SCRIPT/VS to format documents that were converted from ATMS. The profile defines to the formatter the substitutions required for the symbols generated by the conversion processor.
- The macro library of SCRIPT/VS controls gathered together as macros that emulate the original ATMS-II control functions.

The conversion processor is designed to convert all the ATMS controls and implicit keying conventions to recognizable SCRIPT/VS symbols and macros. The output of the processor can then be formatted by use of a supplied ATMS conversion profile along with a collection of SCRIPT/VS macros that emulate as closely as possible the original ATMS functions by use of SCRIPT/VS controls.

Because of a small number of basic differences between ATMS and SCRIPT/VS, there are some functions in ATMS that are not directly convertible. The user should be aware of these when planning to use the process because editing of the document may be necessary to achieve the desired output.

The areas of ATMS to SCRIPT/VS limitations include:

- floating skips
- floating keeps
- STAIRS paragraph numbering
- dynamic page margins
- hyphenation
- left reference numbering
- widow zone control
- text block indention
- text line indention
- overstriking
- revision markers
- line controls within split text
- GML

In many of these areas the most noticeable difference is the fact that the implementing macro causes a line break in SCRIPT/VS that does not occur in the original ATMS.

CONVERSION TECHNIQUE

This section describes ATMS-II formatting controls identified by the occurrence of an Application Control Definition (ACD) (usually !) and an Application Type Definition (ATD) (t,l,m,f,i,x), and their conversion to SCRIPT/VS symbols and macros by the SCRIPT/VS ATMS-II attribute processor.

Non-control conversion is also considered. The document header records in the ATMS-II FT00 format contain information that must be communicated to the formatter. The page width, page depth, and tab settings are extracted from the document headers and inserted into the output as SCRIPT/VS macros and symbols.

Hyphenating Words

In ATMS-II, 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 formatter control to obtain the same effect.

Hyphens in the middle of an input line are retained by ATMS.

Conversion Program Operation

The ATMS-II file(s) in FT00 format may be imported into the SCRIPT/VS Document Library Facility or used directly as input to the formatter in batch mode.

Conversion of the ATMS-II controls and ATMS-II GML into SCRIPT/VS symbols and GML can be accomplished during an IMPORT operation, the SCRIPT/VS formatting process, or a READ operation. 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 Document Library Facility. This input processing program converts the ATMS-II controls and ATMS-II GML as described above. When the record conversion is complete, the formatter or the IMPORT or READ routine gains control in order to continue with the task.

NON-FORMAT COMMAND CONVERSION

The following describes the conversion of each ATMS-II non-formatting control.

End of Embedded Control

The !x is deleted.

ATMS-II 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.

Subdocument Identifier

The subdocument identifier !i is converted to a .SE and some .DM control words with all of the units that follow the !i being converted to elements of the macro.

The macros thus defined must be known to SCRIPT/VS when formatting documents that reference the macros through the ATMS-II !m syntax. To accomplish this, the subdocuments containing the macros may be specified on the SCRIPT command statement through the use of the SYSVAR option. For example, to SCRIPT an ATMS-II document (ATMSDOC) that contains !m's that are defined by another subdocument (SUBDOC), the following command is required:

```
SCRIPT ATMSDOC (PROFILE(ATMSPROF) SYSVAR(A SUBDOC))
```

The IBM-supplied ATMS-II profile document (ATMSPROF) examines SYSVARs A through J to determine if they have been set. If so, their values are taken as the names of documents to be imbedded prior to the start of formatting of the primary document. This limit of 10 names can be changed by the user by altering ATMSPROF at his own installation.

Master Document Fragment

The master document fragment identifier !f is converted to a "&F" symbol that is resolved to an .DD and .IM control words at formatting time.

Tab Setting

Tab settings from the document header are resolved by the input processor and converted into the formatter symbol form of &@TTABS t1 t2 etc. Since the current tab settings are used as parameters to other ATMS-II commands, it is necessary to assign the tab values to set symbols for use in the resolution of other ATMS-II commands.

FORMATTING CONTROL CONVERSION

In all of these descriptions, it must be understood that the ATMS-II controls are converted to equivalent SCRIPT/VS symbols by the processor as described below. The SCRIPT/VS controls that are mentioned in the descriptions are brought in by SCRIPT/VS at formatting time and do not exist in the output of the processor. Additionally, the controls mentioned are not necessarily all-inclusive. The complete contents of each symbol-to-macro-to-control substitution is contained in the ATMSPROF previously mentioned and the conversion macros delivered with SCRIPT/VS.

Unformatted Text Mode

The unformatted text mode command !tu in ATMS-II means no concatenation and no justification and corresponds to the SCRIPT/VS control .FO OFF.

Formatted Text Mode

The formatted text mode !tf in ATMS-II means concatenation only, not justification also, which makes it correspond to .CO ON.

Explicit Paragraphing Specification

The !tf command has a number of variations, which are discussed below:

- !tfn1;n2;n3;n4 allows the user to set parameters relating to the formatting of the page and paragraphs within it. The parameters correspond to formatter commands as follows:
  - n1 (indent of first line of paragraph) -- .IL value, where value is the character position on a line corresponding to the n1'th tab setting.
  - n2 (paragraph body indent) -- .IN &tb(n2)
  - n3 (line length) -- .CL n3
  - n4 (inter-paragraph spacing) -- .SK n4. This skip is taken at the beginning of the paragraph and not at the end as in ATMS.

- !tf; causes text to be formatted corresponding to the parameters set in the previous !tfn1;n2;n3;n4. Note that the command 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-II 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) commands.

The input processor will recognize these conditions in !tf (formatted mode) and insert the appropriate symbols.

Headings and Footings

The input processor converts the ATMS-II heading and footing text and controls into SCRIPT/VS format. ATMS-II recognizes the start of heading and footing information, by the commands !tuhnn, !tuhcnn, !tufnn, and !tufcnn. These heading and footing commands convert to formatter commands as follows:

- nn=12 (all pages) converts to .RF ON or .RH ON.
- nn=13 (all odd pages) converts to .RF ON or .RH ODD.
- nn=23 (all pages except first) resolves to .RF ON or .RH ON, and printing is suppressed for the first page.
- nn=24 (all even pages) resolves to .RF EVEN or .RH EVEN.
- nn=35 (all odd pages except first) resolves to .RF ODD or .RH ODD, and printing is suppressed for the first page.

The ATMS-II capability of specifying centering with the head or foot control will be accomplished by the generation of a .CE ON after the .RH or .RF control. A .CE OFF will then be generated at the next subsequent !tu or !tf.

Unformattable Center Text

The ATMS-II center control !tuc resolves to .FO OFF and .CE ON. The recognition of the concluding !tf or !tu is performed by the input processor.

Keep Text

The ATMS-II keep commands !t(, !t), !t)(, are converted to .KP ON;.KP OFF.

Skip Lines Conditionally

!t+nn translates to .SK nn.

(Note: The + can also be keyed as = by ATMS-II users and is supported.)

Unconditional Skip

The ATMS-II unconditional skip control !t+nn;u converts to .SP nn.

Floating Skip

The ATMS-II floating skip control has a number of variations defined below:

- !t+nn;a (top or bottom of page) resolves to .KP FLOAT; .SP nn a; .KP OFF.
- !t+nn;b (bottom of page) is not supported directly by SCRIPT/VS; therefore, SCRIPT/VS resolves this to the same as above.
- !t+nn;t (top of page) resolves to .KP DELAY; .SP nn a; .KP OFF (the formatter treats a delayed keep as a delayed floating keep).

Line Spacing

The ATMS-II controls for single (!tss), double (!tds), and triple (!tts) spacing resolve to .LS 0, .LS 1, and .LS 2, respectively.

Justification

The ATMS-II justification controls !tj and !tnj resolve to .JU ON and .JU OFF, respectively.

Paragraph Numbering

The ATMS-II paragraph identification and numbering control !tps;nxx has no equivalent in the formatter and is treated as a simple !tf.

Text Alignment Controls

- !tal (align left) in !tf mode resolves to .FO ON;.JU OFF and in !tu mode resolves to .FO OFF.
- !tar (align right) in !tf mode resolves to .FO RIGHT and in !tu mode resolves to .RI ON.
- !tac (align center) in !tf mode resolves to .FO CENTER and in !tu mode resolves to .CE ON.
- !taj (align justified) is allowed only in !tf mode and resolves to .FO ON.

These cause a line break in SCRIPT, unlike ATMS-II, where they do not.

Page Margin Control

The !tm control has no equivalent in SCRIPT/VS and is translated to a null operation. (Note: The user can use the BIND option at formatting time however or provide an alternative macro definition.)

Width and Depth Controls

!tw;n1;n2 (set page width and depth) resolves to .CL n1 and .PL n2, which do cause breaks in SCRIPT/VS unlike those in ATMS.

Include Floating Keeps

!tif has no equivalent in SCRIPT/VS and becomes a null operation. (Note: The user can provide an alternative macro definition.)

Comment Control

The ATMS-II include comment control (!tcm) resolves to .*.

Hyphenation Control

The ATMS-II hold hyphenation control (!thh) resolves to .HY SUP which suppresses hyphenation until the next space. The hyphenation level control (!thm;n) resolves to .HY ON if level n is greater than 0; otherwise it is .HY OFF.

Controlling Left Reference Numbering

The ATMS-II controls for controlling output with left reference numbering (!tls and !tle) in combination with their print command are not supported by the SCRIPT/VS formatter. They are consequently a null operation. Numbering can only be specified in the SCRIPT command and then only on the right.

Start New Page

The ATMS-II start new page control resolves to a .PA.

Widow Zone Control

The ATMS-II widow zone control (!twz) is a null operation, because there is no support in SCRIPT/VS for eliminating widows.

Page Definition

The ATMS-II page definition control (!tpd), when used before any body text, changes the default width and length from those specified in the header. After body text is started (as indicated by a !tu, !tuc, or any !tf), only the length is changed.

Text Block Indention

The ATMS-II indent block control (!tib) can only be partially supported in SCRIPT/VS. The second parameter, the number of blocks to be indented, is only supported for formatted paragraph blocks. In !tu mode it is not supported. The first parameter, the amount of indent for blocks, sets the indention value of all text of the same mode (!tu or !tf).

Text Line Indention

The ATMS-II text line indention control (!tir) is resolved to a .IR +n or a .IR 0 if no parameter is specified.

The !til control is resolved to .IN +n.

Overstrike

The ATMS-II overstrike commands !los;x and !loe resolve to .US ON and US OFF. However, in ATMS-II it is possible to specify the character to be used for overstriking by means of the "x" parameter. The SCRIPT/VS formatter has no way of changing the system default character of "_".

Page Number Control

The ATMS-II page number symbol !lpn resolves to the default SCRIPT/VS page number symbol &.

Stop Code

The ATMS-II typewriter input capability specified by !lsc resolves to a generated bullet character, the same as is done now for ATMS-II operations on the peripheral queues. This is consistent, because the input processor is preparing data for the formatter under SCRIPT/VS with the Document Library Facility. The optional spaces entered by the ATMS-II user will be removed by the input processor.

Text Split

The SCRIPT/VS formatter split text control (.SX) provides an equivalent function to the ATMS-II split text control (!lst), albeit with a different syntax.

There can not be any imbedded ATMS-II controls in the lines which are to be split. Therefore, any !l controls encountered in a split text line will cause unpredictable results.

Revision Markers

The ATMS-II revision markers are handled by three commands: !lre, !lrs;x, and !trs;n.

- !trs;n sets the space between the marker and the start of text on the output line. SCRIPT/VS has a set limit of two characters, so this control becomes a null operation.
- !lrs;x identifies the revision marker starting point in the text. If the x is not specified, it defaults to the OR symbol.
- !lre identifies the end of markers. There is no concept in ATMS-II of levels of markers -- only one is allowed at a time. Any !lrs replaces any previous one.

The inclusion of markers in the output is controlled in ATMS-II by the print command option (m). Similarly, when revision markers are to be printed by SCRIPT/VS in documents converted by the ATMS-II 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-II counters are handled by two controls, !tset and !lcn of thr 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
```

Hexadecimal Code	Character	Hexadecimal Code	Character	Hexadecimal Code	Character
4A	¢	AB	⌊	B5	5
4C	<	AC	⌈	B6	6
4F	|	AE	≥	B7	7
5F	¬	AF	•	B8	8
6E	>	B0	0	B9	9
8B	{	B1	1	BB	⌋
8C	≤	B2	2	BC	⌉
8F	+	B3	3	BE	≠
9B	}	B4	4	BF	—
9F	▪				

Figure 17. Character Codes Regcognized by ATMS-II Conversion: The triplet (character-backspace-character) conventions for special characters defined in ATMS-II Terminal Operations Guide are recognized and translated into a single hexadecimal character.

---

style is

a or la for upper- and lowercase alphabetic
r or lr for upper- and lowercase Roman
n for Arabic

Note: Counters may not be used in split text lines.

Date Control

The ATMS-II today's date callout controls (!lda and !lde) resolve to a macro that uses the built-in SCRIPT/VS system symbols to create equivalent output.

Uppercase Control

The ATMS-II uppercase start (!lus) and end (!lue) resolve simply to .UP ON and .UP OFF respectively.

Triplets and Backspaces

In ATMS-II there is an entry convention involving backspaces for characters which do not occur on the keyboards but which can be represented on the output printers by graphics. These entry conventions are defined in ATMS-II Terminal Operations Guide.

The input processor will convert defined triplets (character-backspace-character) to a single hexadecimal character that represents the triplet. All other instances of backspaces are left unchanged.

The special characters and their hexadecimal codes are listed in Figure 17.

ATMS CONTROL - SCRIPT/VS MACRO RELATIONSHIP

Figure 18 on page 195 identifies identifies the ATMS controls that are converted and the SCRIPT/VS symbols to which they are converted.

The substitution for the symbols on the right is contained in the profile "ATMSPROF" and should be looked at in conjunction with this list.

In addition, the contents of each macro which is eventually invoked by the substitution should be examined.

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	&@LRS
!lsc	&@LSC
text1!lst;xtext2	&@LST @text1@x@text2
!lue	&@LUE
!lus	&@LUS
!mname	:name
!t(	&@TBKP
!t)	&@TEKP
!t)(	&@TEBK
!t+nn;x	&@SKIP nn X
!tac;n	&@TAC N
!taj;n	&@TAJ N
!tal;n	&@TAL N
!tar;n	&@TAR N
!tcm	&@TCM
!tds	&@TDS
!tfn1;n2;n3;n4	&@TF n1 n2 n3 n4
!tfe	&@TFE
!thh	&@THH
!thm;n	&@THM n
!tib;n1;n2	&@TIB n1 n2
!tif	&@TIF
!til;n1;n2;n3	&@TIL n1 n2 n3
!tir;n1;n2;n3	&@TIR n1 n2 n3
!tj	&@TJ
!tle	&@TLE
!tls	&@TLS
!tm;n1;n2	&@TM n1 n2
!tnj	&@TNJ
!tnp	&@TNP
!tpd;n1;n2	&@TPD n1 n2
!tps;nxx	&@TPS Nxx
!trs;n	&@TRS n
!tset;id;val;style	&@TSET ID val STYLE
!tss	&@TSS
!ttab;n1;...;nm	&@TTAB n1 ... nm
!ttab-;n1;...;nm	&@TTABM n1 ... nm
!ttab+;n1;...;nm	&@TTABP n1 ... nm
!tts	&@TTS
!tu	&@TU
!tuc	&@TUC
!tufnn	&@TUFnn
!tufcnn	&@TUFCnn
!tuhnn	&@TUHnn
!tuhcnn	&@TUHCnn
!tw;n1;n2	&@TW n1 n2
!twz;n	&@TWZ n
!x	null

Figure 18. ATMS-II Controls to SCRIPT/VS Symbols Conversion

The "TSO Data Utility" program product provides users of the Time Sharing Option (TSO) of OS/VS2 with a FORMAT function. TSO/FORMAT allows TSO users to enter formatting controls into TSO text data sets that indicate the type of formatting required.

SCRIPT/VS provides TSO/FORMAT users with an easy migration to more powerful formatting. SCRIPT/VS control word syntax is identical for many TSO/FORMAT control words, and SCRIPT/VS control words which are new to TSO/FORMAT users provide many new or enhanced functions.

CREATING A TSO/FORMAT COMPATIBLE ENVIRONMENT

SCRIPT/VS provides a symbol and macro facility which allows you to process TSO/FORMAT documents without modifying the documents themselves. Figure 19 lists the TSO/FORMAT control words that are not directly supported by SCRIPT/VS. You can define a SCRIPT/VS macro with the name of the TSO/FORMAT control word that executes the equivelent SCRIPT/VS control word. See "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137 for details.

You can place your macro definitions in a Profile to ensure that they are always available when you process TSO/FORMAT documents. See "Chapter 2. Using the SCRIPT Command" on page 13 for details about the PROFILE option.

THE SCRIPT COMMAND IN TSO

The SCRIPT command is used to call SCRIPT/VS to format an input file, and is similar to the FORMAT command. See "Chapter 2. Using the SCRIPT Command" on page 13 for details about the SCRIPT command, its options, and its TSO naming conventions.

TSO/FORMAT Control Word	SCRIPT/VS Equivalent Control Word
.AD [Adjust]	.RI [Right Adjust]
.BL [Blank]	.TR [Translate Character]
.EN [End]	.CE [Center] OFF
.HI [Hanging Indent]	.UN [Undent], .OF [Offset]
.PI [Paragraph Indent]	.PP [Paragraph Start]
.RP [Reprint]	
.ST [Stop]	.QU [Quit]

Figure 19. Unsupported TSO/FORMAT Control Words: A SCRIPT/VS control word which provides an equivalent function is listed for each TSO/FORMAT control word, except .RP [Reprint]. SCRIPT/VS provides no equivalent of the Reprint function.

This section describes each control word in the SCRIPT/VS language. All parameters are shown with descriptions of their effect on processing. Usage notes and examples are included.

CONTROL WORD SYNTAX

All control words have two-character names. A control word is identified by a period (.) in the first character position of an input line, followed by the two-character name. If the control word accepts parameters, these follow the control word name, separated from the name and from each other by blanks:

```
.xx parm parm parm
```

Some control words accept a parameter that is a line of text or another control word line.

The control word separator character may be used to allow more than one control word to be entered on one line:

```
.c1 parm;.c2;.c3 parm parm
```

SCRIPT/VS scans every control word line to find the first control word separator character, and divides the line at that point. The part of the line before the control word separator is processed as a complete control word line, and the remainder, to the right of the control word separator, becomes the next input line. The period in ".c2" in this example appears in the first character position, allowing ".c2" to be recognized as a control word.

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

MACROS

SCRIPT/VS macros are invoked in the same way as control words with a period in the first character position of an input line. Macro names, however, need not be two characters long. Parameters may be specified on a macro line in the same way as on a control word line. If a macro called "mymac" were defined with the .DM [Define Macro] control word, it would be invoked as a control word:

```
.mymac parm parm parm
```

If a macro is defined with the same name as a control word, it is invoked instead of that control word, assuming macro substitution is ON. This is how you can redefine the function of a control word.

THE CONTROL WORD MODIFIER

The SCRIPT/VS control word processor recognizes a single quote (') after the period as a control word modifier. A control word, such as .CE, can be entered with the modifier as follows:

```
.'CE Center this line.
```

The control word modifier changes the usual operation of the control word processor in two important ways:

1. No macro search is done. Even if a macro of the given name exists, the control word is invoked, not the macro.

2. No control word separator scan is done. Any control word separators in the line are left there as ordinary text characters. Thus, a control word entered with the control word modifier must be the last control word on that line.

Since the control word modifier indicates that the name that follows is a control word name, it must be two characters long. Therefore, the name need not be delimited with a blank:

```
.'CECenter this line.
```

works in exactly the same way as:

```
.'CE Center this line.
```

Since no control word separator scan is done, a control word that accepts a line of text may be entered with the control word modifier to protect any separator characters that appear in the line as part of the text:

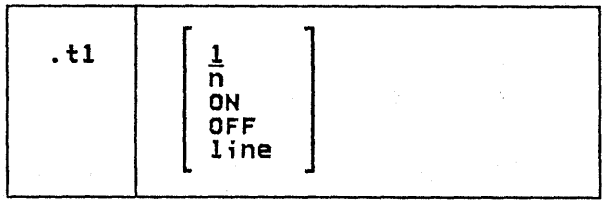
```
.'CE centered line; one line.
.'H3 Using the ; in text
```

TYPE 1 CONTROL WORDS

There are several control words that all have the same syntax and accept the same parameters, called Type 1 control words. All the Type 1 control words are analyzed by a common preprocessor before the individual control word processors get control, and they therefore

have certain things in common. (There are other control word types as well, and their syntax is explained in the individual control word descriptions.)

The fictitious control word .t1 is used in this discussion to represent any Type 1 control word:



Where:

n is a positive integer that indicates the number of input lines to be processed by the Type 1 control word. The default is 1, meaning that the next input line after the control word is to be processed by this control word.

ON starts an open-ended range of input lines to be processed by the Type 1 control word, until terminated with the OFF parameter.

OFF stops the effect of the Type 1 control word, whether it was started with the ON parameter, or with a number in "n" that has not yet been exhausted.

line is a single line that is to be processed by the Type 1 control word. The single input line

```
.t1 this is a line
```

is equivalent to the two lines

```
.t1 1
this is a line
```

The line given on a Type 1 control word is assumed to start with the first non-blank character. Thus, the following two forms operate identically:

```
.t1 this is a line
.t1   this is a line
```

The keywords ON and OFF and numbers given in "n" are recognized only if they are the only parameters on a control word line. If there are other parameters, it is assumed to be a "line" to be processed. Thus, Type 1 control words in the form:

```
.t1 On old Olympus' towering top
.t1 555 Bailey Avenue
```

are taken as control words that have a line of text, not as requests to process large numbers of input lines.

SPACE UNITS

All control word parameters that specify horizontal or vertical dimensions may be specified in any recognized space units, unless otherwise noted in the control word description. The recognized space units are:

```
Inches - aaI
Picas/points - aaPbb
Ciceros/Didot points - aaCbb
Em-spaces - aaM
Millimeters - aaW
Character spaces - aa
Line spaces - aa
```

where aa is any valid number.

NOTATIONAL CONVENTIONS

The format of each control word is described as shown above in a format box. The notation conventions used are:

1. Keywords that must be entered as shown are in UPPERCASE. If the keyword can be abbreviated, the abbreviation is shown in uppercase letters, and the rest of the word in lowercase, as in ROman. (You may enter the control word and the keyword in uppercase or lowercase.)

2. Parameters for which you must supply the value are shown in lowercase letters.

3. If there is a default value for a parameter, it is underscored. In some cases, one parameter can have a default but other parameters must be specified.

4. A single optional parameter is shown in [small brackets]. This parameter may be specified or omitted.

5. A parameter that allows you to choose one of several possibilities, or none, is shown as a list enclosed in large brackets, as in the Type 1 example above.

6. A list enclosed in {braces} indicates that one of the choices must be specified. For example, the notation

```
{ ON      }
{ OFF     }
{ INCLUDE }
{ IGNORE  }
```

signifies that this parameter must be specified as ON, OFF, INCLUDE, or IGNORE.

7. A single required parameter is shown without any brackets or braces.

8. If the format box has internal horizontal lines, as in the .DM [Define Macro] control word description, each segment of the box depicts an alternative form of the control word.

9. An ellipsis (...) indicates that a parameter can be repeated. The form "d1 ... d9" indicates that you may specify up to nine "d" values, separated by blanks. The form "c ..." indicates that you may specify as many values of c as will fit on that input line.

... [SET LABEL]

The ... [Set Label] control word marks a line of your SCRIPT/VS file or macro so that that line may be referred to in a .GO [Goto] control word.

...	label [line]

Where:

label is a name of up to eight characters that can be used to refer to this line of your SCRIPT file or macro.

line is the active part of this input line. The first nonblank character after the label is treated as if it were the beginning of the line; it may therefore be a control word, but a text line associated with a label may not begin with blanks. If the input line has a label only and no active line, then the next line to be processed is the one following the labeled line.

Default: None

Notes:

- When the ... control word is encountered, SCRIPT/VS saves the information necessary to enable it to find this line again if a .GO [Goto] control word is encountered. Any valid SCRIPT/VS input line may follow the label, or the label alone may occupy the line.
- Use of labels and the .GO control word is restricted to one input file or macro. That is, when a new file is imbedded or appended, a new set of labels is in effect while that file is being processed. SCRIPT/VS can only branch to a label within the same input file or macro.
- Every label in a particular file must be unique. If two identical labels are found in the same file, an error message is generated.

  Multiple labels with the same name are tolerated in macros, but when searching for labels in a macro, only the first occurrence of a label will be found.
- The .GO function can be relatively inefficient in files. You should use it sparingly in situations where it is the best way to achieve the required results. When going to a label that is later in the input file, it is most efficient when the label is not far from the .GO; when going to a label that is earlier in the file, it is most efficient when the label is near the beginning. Label processing in macros is a much more efficient operation than in files. However, it is more efficient to branch to a label that is earlier in a macro as labels in macros are always searched for from the top of the macro.
- A space is not required after the control word itself. (This is the only control word where this is true without the control word modifier.) To set a label called "HERE", either "... HERE" or "...HERE" may be used.
- The ... for a label must begin in column 1. That is, it must be the first control word on the input line.

Example:

Suppose you had a file called REPORT1 that contained a summary of activity for January, another file, REPORT2, for February, REPORT3 for March, and so forth. Now, if you wanted to create a year-to-date report by imbedding all the report files up to last month's report, you could use this sequence of SCRIPT/VS control words:

```
.se ctr = 1
...loop .im report&ctr
.se ctr = &ctr + 1
.if &ctr lt &SYSMONTH .go loop
```

The first time the .IM [Imbed] is processed, the value of the symbol "&ctr" is 1, so the filename "report&ctr" becomes "report1." The next control word adds one to the value of the symbol; it is now 2. If the month is later than March (month 03), then the value of the counter is less than the month number, and the loop is processed again. This time the filename "report&ctr" becomes "report2." The loop continues until the counter is equal to the current month number.

.AP [APPEND]

Use the .AP [Append] control word to insert an additional SCRIPT/VS file at the end of the file just printed.

.AP	file-id [token1 ... token14]

Where:

file-id specifies the file-id of the file to be appended to the file which has just been processed. This file-id may refer to an id which has been established using the .DD control word. A description of the file-id in the various environments in which SCRIPT/VS operates is given in Chapter 2.

tokens are positional values that may be passed to the appended file. The first token (word) becomes the value of the symbol &1, the second token becomes the value of the symbol &2, and so forth. The symbol &0 contains the number of tokens given. File tokens may be a maximum of 8 characters long; up to 14 can be specified.

Default: None.

Notes:

- When the .AP control word is encountered, the current file is closed, and the specified SCRIPT file is processed as a continuation of the SCRIPT/VS output from the previous file. Text or control words following the .AP control word in the current file are not processed.
- The .AP control word is especially useful for iterative processing of a file. See the example given under the description of the .EF [End of File] control word.
- The .AP control word closes the current file and starts reading input lines from the appended file. In this sense, the .AP control word marks the end of the file; since it is closed, SCRIPT/VS will not return to it when the appended file is finished. If the .AP control word is not actually at the end of the file, the lines after it are not read.
- The symbols &1 through &14 are reset whenever a .IM or .AP control word is processed, and the token &0 is reset to the number of non-null tokens. If you want to leave token &1 unset but set token &2, you may use a percent sign (%) in place of token1 (or any other token you want left unset).

Example:

```
.ap abc 10
```

The input file is closed. The contents of the SCRIPT/VS file ABC are processed immediately following the line of the current file which precedes the .AP request. The token 10 is passed to the appended file, so if the file ABC contains a control word of the form:

```
.in &1
```

the result is:

```
.in 10
```

.BC [BALANCE COLUMNS]

Use the .BC [Balance Columns] control word to cancel and restore column balancing for multiple column formatting.

.BC	[ON / OFF]

Where:

ON indicates that you want SCRIPT/VS to balance columns. ON is the initial setting as well as the default.

OFF indicates that you do not want SCRIPT/VS to balance columns when a page eject or column definition is encountered.

Initial Setting: ON

Default: ON

Notes:

- When column balancing is in effect, the number of lines in each column is made as equal as possible before the material on that page is printed.
- If a blank line that was generated by the .SK [Skip] control word ends up at the top of a column after balancing, it is discarded.
- When column balancing is off, the number of lines in each column is determined by explicit .CB [Column Begin] control words or by filling all columns, but no attempt is made to equalize the number of lines in all columns.
- A column that is ineligible for balancing may not have any lines moved into it or out of it to make it longer or shorter. A column is made ineligible if the next column is started explicitly by a .CB [Column Begin] control word or by a .H0 - .H6 [Head Level 0 - 6] control word that causes an eject to a new column.
- If a page eject occurs while processing multiple columns, this does not mark the current column ineligible for balancing. A column eject that changes the current column from the last column of a page to the first column of the next page is the same as a page eject. Unlike intra-page column ejects, it does not mark the old current column as ineligible for balancing.

.BF [BEGIN FONT]

Use the .BF [Begin Font] control word to begin a new font.

.BF	font

Where:

font is the name of the font. For the 3800 printer, the fonts that may be specified correspond to those fonts that have been defined to the formatter in the form of font availability and width tables. The font name is the same as that which may be specified on the CHARS option of the SCRIPT command.

Default: None.

Notes:

- This control word allows text to be formatted using a specified font (a set of characters of one size and style). When the .BF control word is encountered by the formatter, all subsequent text characters are formatted using the specified font. The specified font remains in effect until another .BF or .PF [Previous Font] control word is encountered with a different font.
- The system symbol &$char is an array symbol that contains the names of the fonts that are avail-

able for the .BF control word. The first name given with the CHARS option of the SCRIPT command or the name of the default font for the logical device is the value of &$char(1). The second, third, and fourth names given with the CHARS option are the values of &$char(2), &$char(3), and &$char(4). &$char(0) contains the number of fonts available.

Examples:

1. This line is in the normal font for this document.

   .BF GB12

   has this effect.

2. You can specify the font to be started symbolically:

   .bf &$char(2)

   starts formatting in whatever font was the second one named on the CHARS option of the SCRIPT command.

.BM [BOTTOM MARGIN]

Use the .BM [Bottom Margin] control word to specify the amount of space to be skipped at the bottom of output pages, overriding the initial value established for the device.

.BM	[v +v -v]

Where:

v specifies the amount of space to be skipped at the bottom of output pages. v must be large enough to accommodate the footing margin (.FM) and the footing space (.FS), both of which are allocated from the bottom margin area. If +v or -v is specified, the current value of the bottom margin is incremented or decremented. If no value is specified for v, the initial setting is restored. The maximum value for the bottom margin is the page length (.PL) less the top margin (.TM) less space for one line.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

- The value set by the .BM control word applies on the page after the one on which the .BM control word is encountered, and all subsequent pages until another .BM is encountered.
- The value given may not be so large that the top margin plus the bottom margin fill the entire page. An error message is issued if you try to set the bottom margin to equal to, or more than, the page length minus the top margin. If you intend to increase the bottom margin so that you can increase the footing margin or the footing space beyond what the old bottom margin would allow, be sure to do it in that order. The rule is, increase the bottom margin before the footing margin or footing space, but decrease the footing margin or footing space before the bottom margin.
- If you specify .BM 0, the footing margin and the footing space are also made zero automatically.
- The .BM control word is not allowed in a keep.

.BR [BREAK]

Use the .BR [Break] control word to ensure that the next input line is not concatenated with the previous line or lines.

.BR	

Notes:

- The .BR control word is necessary only when SCRIPT/VS is concatenating input lines. It causes the preceding line to be formatted as a short line, if it is shorter than the current column length.
- Many other control words have the effect of a break. No .BR control word is necessary when one of these is present. See Figure 25 on page 311 for a list of these control words.
- A leading blank or tab on an input line has the effect of a break.
- The .BR control word can ensure that some other control words are not effective too early or too late, for example:

```
.br;.tr $ 40
```

  may be used to prevent the translation from being effective on the preceding text line, and

```
.tr $ $;.br
```

  may be used to make sure the translation does not affect the next line.

Example:

```
Heading:
.br
New paragraph...
```

On SCRIPT/VS output, these lines appear as:

```
Heading:
New paragraph...
```

If the .BR control word were not included, the lines would print as:

```
Heading: New paragraph...
```

.BT [BOTTOM TITLE]

The .BT [Bottom Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the bottom of the current page, and each subsequent output page.

.BT	[n] /part1/part2/part3/

Where:

n is the number of the bottom title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even bottom title 1" sets the same storage buffer as "even top title 6." See the discussion of the .FS [Footing Space] control word for information on how to allocate space on your output page for bottom titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

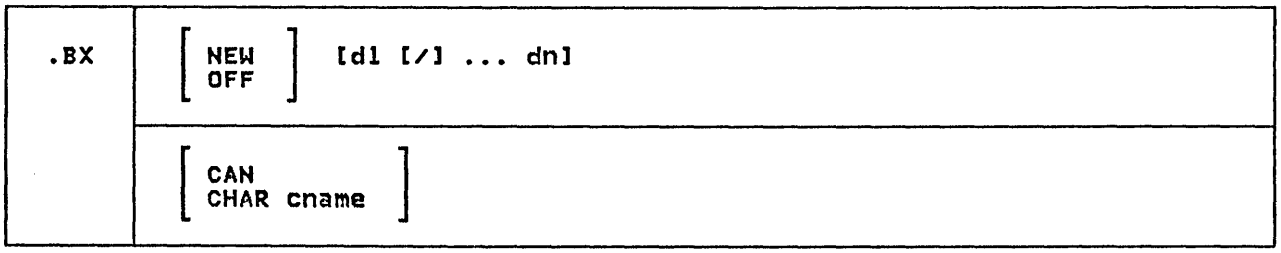
/ is any character that does not appear in part1, part2, or part3.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RT [Running Title]. See the discussion of the .RT control word for further information about running titles, including those for the bottom of pages.

.BX [BOX]

The .BX [Box] control word defines and initializes a horizontal rule for SCRIPT/VS output and defines vertical rules for subsequent output lines. With this control word, you can format tables, charts, or text within neatly formatted boxes. The BOX control word is designed to work only in non-mixed pitch situations.



Where:

d1...dn are the distances from the left margin where you want to place vertical rules in output text. This format of the control word initializes the box and draws a horizontal line, with vertical descenders at the columns indicated. A slash (/) indicates a discontinuity between columns, with no horizontal rule connecting these columns. Subsequently, entering the .BX control word with no operands causes SCRIPT/VS to print a horizontal line with intersections at the columns indicated. This allows two or more separate boxes to be drawn side by side. d1 - dn may be specified in space units but the designated values will be converted to M spaces, since the .BX control word is supported for monospaced fonts only.

NEW if no columns are given, or if no box is now going, the NEW function is ignored; otherwise, a new box is started, and the previous box is stacked. This capability allows boxes to be drawn inside boxes.

OFF causes SCRIPT/VS to finish drawing the box, by printing a horizontal line with vertical ascenders at the columns in effect. If this box was started as a 'NEW' box, the previous box is reinstated when this one is ended. If columns are specified with OFF, and no box is currently in effect, then a box bottom will be drawn according to the column specifications given.

CAN causes the box to be cancelled without a box bottom. If the box being cancelled is a nested box, the next higher box is reinstated.

CHAR Allows you to override the assumed box character set that SCRIPT/VS uses to draw boxes. When you specify CHAR, you must also specify the name of the box character set to be used.

cname is the name of the box character set SCRIPT/VS is to use for drawing all subsequent boxes. The valid names are:

- **TRM** terminal character set
- **32T** 3270 text characters
- **TNC** 1403 TN or 3211 T11 character set
- **38C** SCRIPT/VS 3800 character set
- **GPC** box characters for 3800 GP12 font
- **APL** APL box characters

Default: Repeat previous box definition.

Notes:

- The .BX control word describes an overlay structure for subsequent text that is processed by SCRIPT/VS. After the .BX d1 d2... line is processed, SCRIPT/VS continues formatting output lines as usual. However, after a line is completely formatted and before it is printed, SCRIPT/VS places vertical lines in the columns indicated by d1, d2, and so on.

  If a data character occupies the same position as a vertical line, the action taken depends upon the logical device for which the document is being formatted. For 3800

logical devices, the vertical line takes precedence over data characters in the same column; for other logical device types, the data character takes precedence.

- The .BX control word causes a break.
- The characters used to draw the box depend on the logical device for which formatting is taking place. SCRIPT/VS assumes an appropriate box character set for each logical device, but you can override this by using the CHAR parameter to force any of SCRIPT/VS's box character sets to be used instead.
- A .BX control word with different columns specified may be used while a box is being drawn. When this happens, vertical ascenders are put in for all the old columns and vertical descenders are used for all the new columns, a horizontal rule is drawn, and the vertical rules are then placed in all subsequent output lines in the new columns designated.
- The column specification for the .BX control word uses a different rule than is used elsewhere in SCRIPT/VS. In control words like .IN, .TB, .CD, the numbers in the control word represent not columns but displacements. The SCRIPT/VS control word .TB 5 means that a tab character should be expanded to enough blanks to fill up <u>through</u> column 5; the next word starts in column 6. In the .BX control word, .BX 5 means to put vertical rules <u>in</u> column 5. Thus, you can use the same numbers for a .TB control word as for a .BX control word, and the vertical bar will be placed in the column just before the beginning of the word following a tab. Further, you can define a box that is to be the full column width symbolically with the following control word:

  ```
  .bx 1 &$cl
  ```

  because the number represents the actual column where the vertical rules should be placed.
- Problems of vertical alignment will occur if the .BX control is used to draw boxes in mixed pitch situations. This will occur if the font being used has characters of different widths, or if monospaced fonts are used which have different widths.
- The characters to be used for box drawing must be in all fonts which are used within a box.

<u>Examples:</u>

1. There is a SCRIPT file called MARYHADA that looks like this:

   ```
   Mary  had a little lamb,
   Whose fleece  was white as snow,
   And everywhere that Mary went,
   The lamb was sure to go.
   ```

   The following input sequence could be used to center this material in a box that is the same width as the current column length:

   ```
   .bx 1 &$cl
   .ce on
   .im maryhada
   .ce off
   .bx off
   ```

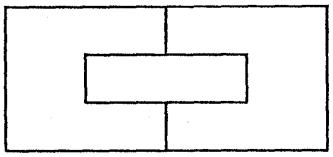
   The result:

   ```
   Mary  had a little lamb,
   Whose fleece  was white as snow,
   And everywhere that Mary went,
   The lamb was sure to go.
   ```

2. An example of a nested box:

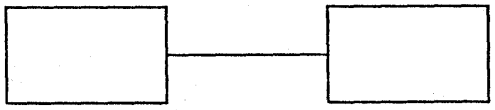
   A nested box was created using the following control word sequence:

   ```
   .bx 10m 20m 30m
   .sp
   .bx new 15m 25m
   .sp
   .bx off
   .sp
   .bx off
   ```

3. The following shows the effect of the slash (/) between column specifications:

   ```
   .bx 5m 15m / 25m 35m
   .sp
   .bx new 15m 25m
   .bx can
   .sp
   .bx off
   ```

.CB [COLUMN BEGIN]

The .CB [Column Begin] control word causes subsequent text to start a new column of output.

.CB	

Notes:

- Use the .CB control word when you want to make the following text appear at the top of a new column. If the current column at the time a .CB is encountered is the last column on the page, the column eject is the same as a page eject, since the next column is the first column of the next page. In this case, the fact of explicitly starting a new column does not prevent the earlier columns from being balanced. If the column eject ends an old column and starts a new column that are both on the same page, the old column is made ineligible for balancing so that the material following the .CB will be at the top of the new column.
- If a floating or delayed keep is waiting for the start of a new column, then the text that follows the .CB appears after the keep.
- A column eject may be performed by certain other control words if the conditions warrant it. If this happens, the function is the same as the unconditional column eject that is caused by the COLUMN-BEGIN control word. The other control words that can cause a column eject are:

  .CC [Conditional Column Begin]
  .H0 - .H6 [Head Level 0 - 6]
  .KP [Keep]
  .FN [Footnote]

- This control word acts as a break. It is not allowed in a keep.

.CC [CONDITIONAL COLUMN BEGIN]

The .CC [Conditional Column Begin] control word causes a column eject if less than a specified amount of space remains in the current column.

.CC	[v]

Where:

v is the amount of vertical space that must remain in the current column for processing to continue without a column eject. If v is omitted, and there is text in the current column, a column eject is performed to the top of the next column. This is equivalent to the effect of .CB [Column Begin]. However, if the current column is empty, then no column eject is done. If the previous column is exactly full, and the current column is empty, the .CC control word with no "v" specified does not cause an eject, and the previous column is subject to column balancing, if this is in effect.

Notes:

- When the .CC control word is encountered, SCRIPT/VS checks to see if there is enough space left in the column. If not, a break followed by a column eject is performed, and the column is made ineligible for balancing. However, if the specified amount of space or more remains in this column, and no column eject is done, subsequent column balancing may divide the text within the specified vertical space. To ensure that text is kept together, use the .KP [Keep] control word.
- This control word is not allowed in a keep.

.CD [COLUMN DEFINITION]

Use the .CD [Column Definition] control word to define how many columns of output are to be formatted on each page and where each column is to start.

.CD	n [p1 [p2 ... p9]]

Where:

n is the number of columns of output to be formatted onto each subsequent output page. It may be any number from 1 to 9.

p1...p9 are positions where the columns are to be placed on the output page, relative to the left edge of the paper (column 1). A position parameter of 0 indicates that a column should be flush with the left edge of the page text area, as defined with the BIND option of the SCRIPT command. Positions may be specified in space units.

Default: None.

Notes:

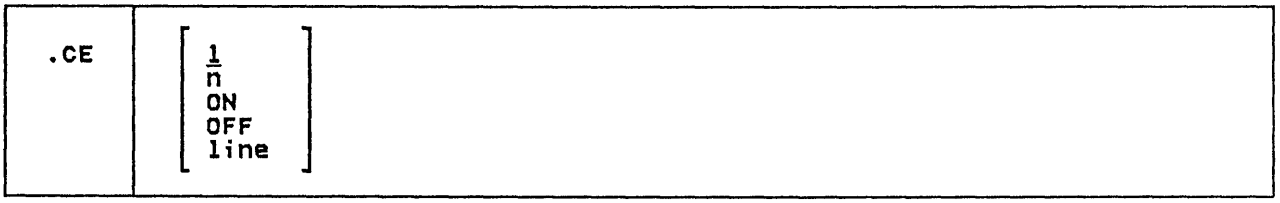
- The .CD [Column Definition] control word causes a **section break** when it is processed. This means that all the text up to that point is processed and positioned on the page using the old definition before the new definition becomes active, even if the new definition is the same as the old.
- The gutter between columns is obtained by defining the column width to a value less than the distance between column starting positions.
- The positions of the columns do not control how wide the columns are to be; you must set the column width, using the .CL [Column Width] control word, to control this. If the current column width is greater than the distance between columns, the results depend upon the logical device for which the document is being formatted.

  For logical devices for impact printers and terminals, SCRIPT/VS simply overlays part of the first column with the second. (It would be possible to define all columns to begin in the same position. If you did this, an entire column would be overlaid with the text of a later column.)

  For 3800 logical devices, no overlaying is possible, since the characters may be of different widths. In this case, the second column is abutted to the first, with no intervening gutter space. The practice is not recommended as the results may be unpredictable - especially when formatting with multiple fonts. You should take care to define compatible column width and starting positions.
- If you specify fewer positions than the number of columns, and had previously specified positions on another .CD control word, those values remain in effect for any columns not respecified. Whenever a .CD control word is used, there must be positions for each column available, either on this control word line, or previously given on another .CD. If you specify .CD n without specifying any positions, and no previous column definition has been specified, the arbitrary values 0, 46, 92, 0, 0, 0, 0, 0, 0 are used.
- If you use several different column formats in a document you can create symbolic names (with the .SE [Set Symbol] control word) or macros (with the .DM [Define Macro] control word) to establish column definitions, column widths, and so on. If you use a single one-column format and a single multiple-column format, you can switch back and forth using the .SC [Single Column Mode] and .MC [Multicolumn Mode] control words.
- This control word is not allowed in a keep.

.CE [CENTER]

Use the .CE [Center] control word to center output lines between the margins.



Where:

n specifies the number of lines to be centered. If omitted, 1 is assumed. If .CE n is specified when .CE ON is in effect, centering is turned off when n lines have been centered, or when .CE OFF is encountered.

ON specifies that subsequent text lines are to be centered.

OFF terminates centering mode if it was ON, or if n has been specified and has not been exhausted.

line is a line of text to be centered. The line is considered to start with the first nonblank character after the .CE control word.

Default: 1

Notes:

- The keywords ON and OFF, and a number of lines to be centered (n), must be the only parameter on the control word line. A string of words that happens to start with one of these is interpreted as a single line to be centered. For example, the control word lines:

  ```
  .ce on top of old smokey
  .ce 555 Bailey Ave.
  ```

  are taken to be of the ".CE line" form, not requests for large numbers of lines to be centered.

- The line(s) are centered between the current left margin, including any indent and offset values in effect, and the right margin. When centering is in effect, no formatting is done on the line. That is, the line is centered as it stands, and it is not filled from other input lines or justified. If a tab character appears in the line to be centered, the tab is resolved before the line is centered.

- This control word acts as a break.

- If the line to be centered is longer than the current column length, it is truncated, and the excess is used on a second line.

- The .RI [Right Adjust] control word is a variant of .CE. If either of these control words is processed, the other is cancelled.

- Contrast the .CE control word with .FO CENTER. The latter allows lines to be formatted by concatenating words until the line is nearly full, but then the filled line is centered instead of being justified, as would be the case with .FO ON.

Examples:

1. To center one line:

   ```
   .ce OFF THE RECORD
   ```

   When this line of the file is displayed, the characters ""OFF THE RECORD"" are centered between the margins:

   ```
                 OFF THE RECORD
   ```

2. To center several lines:

   ```
   .ce on
   IBM Santa Teresa Laboratory
   Bailey Avenue, San Jose
   95150
   .ce off
   ```

   Each of the 3 lines between ON and OFF is separately centered:

   ```
      IBM Santa Teresa Laboratory
        Bailey Avenue, San Jose
               95150
   ```

.CL [COLUMN WIDTH]

The .CL [Column Width] control word sets the width of each column of SCRIPT/VS output.

.CL	[0 / h / +h / -h]

Where:

h is the width of each column of formatted output. It may not be larger than the logical device width. It may be expressed in horizontal space units.

+h increases the current column width by the specified amount.

-h decreases the column width by the specified amount.

Initial Setting: Same as Line Length.

Default: Restore same width as Line Length.

Notes:

- The .CL control word should be used in conjunction with the .CD [Column Definition] control word to define the width of each column from the displacements given. If the column width is greater than the space left between columns, the columns may overlay each other. An intercolumn gutter is allocated by making the column width about three em-spaces shorter than the distance between columns.
- The left and right margins of top titles and bottom titles (running titles), and running headings and footings, are governed by the line length, not the column width. (The line length can be changed with the .LL [Line Length] control word.)
- If the column width has never been set explicitly, it has the same value as the line length. If you set the column width to zero (.CL 0), this makes it the same as though you had never explicitly set the column width. Note that changing the column width by means of the .LL control word means that the column width change will take effect immediately, even though the line length change will not take effect until the following page.
- This control word causes a break.

.CM [COMMENT]

Use the .CM [Comment] control word to place comments within a SCRIPT file.

.CM	[comments]

Where:

comments may be anything; this input line is not used in formatting the output. However, since this is a control word, the input line is scanned for control word separators.

Notes:

- The .CM control word allows comments to be stored in the SCRIPT file for future reference. These comments can be seen when you edit the file, or when you print it using the UNFORMAT option of the SCRIPT command.

  The comments may also be used to store unique identifications that can be useful when attempting to locate a specific region of the file during editing.
- If you want an entire line to be ignored, and not scanned for control word separators, you can use another form of comment. Any line that begins with ".*" is ignored. ".*" is not considered to be a control word, but .CM is.

- The .CM control word can be used in conjunction with the .IF control word to enable or disable strings of control words. For an example of how to do this, see the discussion of the .IF control word.

Example:

```
.cm   Remember to change the date.
```

The line above is seen when examining an unformatted listing of the SCRIPT file, and it reminds you to update the date used in the text.

.CO [CONCATENATE MODE]

Use the .CO [Concatenate Mode] control word to cancel or restore concatenation of input lines and truncation at the current column length.

.CO	[ON / OFF]

Where:

ON restores concatenation of input lines. ON is the initial setting, as well as the default value.

OFF cancels concatenation of input lines. If justification is still in effect, .CO OFF results in each line being padded with blanks to the column length.

Initial Setting: ON

Default: ON

Notes:

- When SCRIPT/VS is concatenating text, output lines are formed by shifting words to or from the next input line. The resulting line is as close to the specified column width as possible without exceeding it or splitting a word; if justification is OFF, output resembles normal typist output. Concatenation is the normal mode of operation for the SCRIPT command.

  When SCRIPT/VS is not concatenating text, there is a one-to-one correspondence between the words on the input and output lines. If SCRIPT/VS is still justifying text, each line that is less than the column length is padded with blank space to fill the column.

- Concatenation is one component of format mode, as controlled by the .FO [Format Mode] control word. The .CO control word is provided for those occasions when you must be able to control concatenation separately, but all ordinary formatting combinations are controlled by the .FO control word, and you should use it instead of .CO whenever possible.

- This control word acts as a break.

.CP [CONDITIONAL PAGE EJECT]

The .CP [Conditional Page Eject] control word causes a page eject to occur if less than the specified amount of space remains in the current column.

.CP	[v]

Where:

v is the amount of vertical space that must remain in the current column for additional lines to be processed without a page eject. If "v" is omitted, a break and a page eject will be done if necessary to get to the top of a page. A break and a page eject will not be done if the current page is empty.

Notes:

- The .CP control word can be used to guarantee that enough space (up to the maximum column depth) will exist in one column to accommodate blank space left by .SP [Space] for a figure to be inserted later.
- To keep formatted text together, use the .KP [Keep] control word.
- This control word is not allowed in a keep.

Example:

```
.cp 2i
```

If less than two inches of space remain on the current column, a page eject is issued before processing continues. If two inches or more remain, processing continues on the current column.

.CS [CONDITIONAL SECTION]

The .CS [Conditional Section] control word allows you to designate sections of the input file that are to be processed conditionally, or ignored.

.CS	n { ON }
	{ OFF }
	{ INCLUDE }
	{ IGNORE }

Where:

n specifies the conditional section code number from 1 to 9.

ON marks the beginning of conditional section n.

OFF marks the end of conditional section n.

INCLUDE tells SCRIPT/VS to process all the input lines between the ON and the OFF control words for conditional section n.

IGNORE tells SCRIPT/VS to bypass every line for conditional section n that falls between .CS n ON and .CS n OFF.

Initial Setting: All sections are INCLUDEd.

Notes:

- The .CS [Conditional Section] control word allows you to designate specific sections of your input file that may be ignored or included conditionally. You may have up to 9 separate section codes, and specify which section numbers are to be included and which are to be ignored. Each section code may be used for many sections. The .CS control word is used to designate conditional sections, and also to specify whether they are to be included or ignored. The ON and OFF operands identify the beginning and end of a conditional section; the INCLUDE and IGNORE operands indicate whether or not SCRIPT/VS should process the input lines within the conditional sections.
- You can use conditional section codes to separate sections of a document that apply to different versions, and specify which version is to be formatted. You may also use this technique to identify confidential sections of a manual that you may sometimes wish to exclude.

- Since the .CS control word does not cause an automatic break, you may turn conditional sections on and off within a paragraph or even within a sentence without disrupting normal output formatting.
- By default, all conditional section codes are assumed to be set to INCLUDE unless explicitly set to IGNORE.
- A conditional section may contain SCRIPT/VS control words as well as text. If the section is ignored, all control words contained in that section will be ignored, except the control word

  .cs n off

  which marks the end of the section.
- Conditional section definitions may be nested to a depth of 9 (that is, a conditional section may contain another conditional section). A nested section is included only if all outer nestings specify INCLUDE. Otherwise, the inner nesting is never noticed, since it is part of an outer section that has been ignored. If a conditional section is nested within another one, the entire section should be enclosed by the outer section.
- The .CS control word may be used in conjunction with the .RC [Revision Code] control word to mark the conditional sections. The .TE [Terminal Input] control word may be used in interactive environments to specify which sections are to be included while the input file is being processed.

Example:

```
.cs 1 ignore
.cs 2 include

In this version of the system there
can be only
.cs 1 on
256
.cs 1 off
.cs 2 on
1000
.cs 2 off
entries in a MACLIB file.
```

Since only conditional section code 2 is to be included, the generated output line is "In this version of the system there can only be 1000 entries in a MACLIB file".

.CW [CONTROL WORD SEPARATOR]

The .CW [Control Word Separator] control word allows you to change the symbol used to separate multiple control words on a single line. The default control word separator symbol is the semicolon (;) character. The .DC [Define Character] control word can also be used to alter the control word separator.

.CW	[c]

Where:

c specifies the character to be used as the "control word separator" character. Any character may be used. If the character "c" is omitted, no character is assigned as the control word separator, and therefore you cannot have more than one control word on a line.

Initial Setting: Semicolon (;).

Default: Nothing. (No separator character.)

Notes:

- All control word lines are scanned for control word separators before they are processed, unless they are specified with the control word modifier. The control word modifier allows the line that accompanies a control word to be treated as text, which may therefore contain control word separators as ordinary text characters.

  The control word modifier is a single quote immediately after the period. The control word ".CE one; two" is scanned before being processed into the two lines ".CE one" and "two". But the line ".'CE one; two" uses the control word modifier to allow the entire string "one; two" to be centered.

  The .CW [Control Word Separator] control word should always be specified with the control word modifier. A .CW control word line, like all unmodified control word lines, is scanned for control word separators before being processed. If you were trying to make sure that the control word separator was set to semicolon by issuing ".cw ;", just the opposite would happen if the semicolon happened to

be the current separator; the line would be separated before being processed into the line ".CW", followed by no more on that line. The control word, when processed, would undefine the semicolon as the separator.

If you always use the control word modifier with .CW, no separator scan will be done, and the character will be preserved as the parameter on the control word:

```
.'cw ;
```

will correctly ensure that the separator is set to ;.

- When the .CW control word is processed, the default control word separator (;) is reset. It may be necessary to change the control word separator character if it is inconvenient to type the default character, or if the default character is used as part of a control word operand, such as part of a symbol specification.
- If a symbol value begins with the control word separator, the rest of the symbol value is treated as though it occupied the first position of the line.
- Control word separators are recognized on a .CM [Comment] line, but not on a ".*" line.
- The following control words must begin in column 1 and may not be placed after a control word separator:

```
.cs n off
.di off
.wf off
.li off
...label
```

When SCRIPT/VS is ignoring a conditional section, preparing a delay imbed, writing to a file, reading literal lines, or searching for a label, no control word processing is done. Each input record is checked in column 1 for the presence of the control word that ends the special processing mode.

- Control words that accept text data (for example, .US or .CE), should not contain the current control word separator as text, unless the control word modifier is used to prevent scanning for the separator.

Examples:

1. Simple change:

```
.'cw ,
.sp 2,.of 5,This section...
```

The above line is equivalent to the lines:

```
.sp 2
.of 5
This section...
```

2. Temporary cancellation to get the separator character into a symbol value:

```
.'cw
.se 2col = ';.cd 2 0 46;.cl 43;'
.se 1col = ';.cd 1;.cl 89;'
.'cw ;
```

In the sequence above, the control word separator is temporarily canceled so that the regular separator (;) can be used as part of the .SE [Set Symbol] control word line. Since the symbols 2col and 1col contain the appropriate control words, they can now be used instead of the actual control words involved. Since the control words are in a symbol that begins with the control word separator, they can be recognized as control words even if the symbol is encountered in the middle of a line. Since the symbols end with control word separators, the effective next line can be concatenated to the symbol name. With the symbols 2col and 1col set as shown, the line:

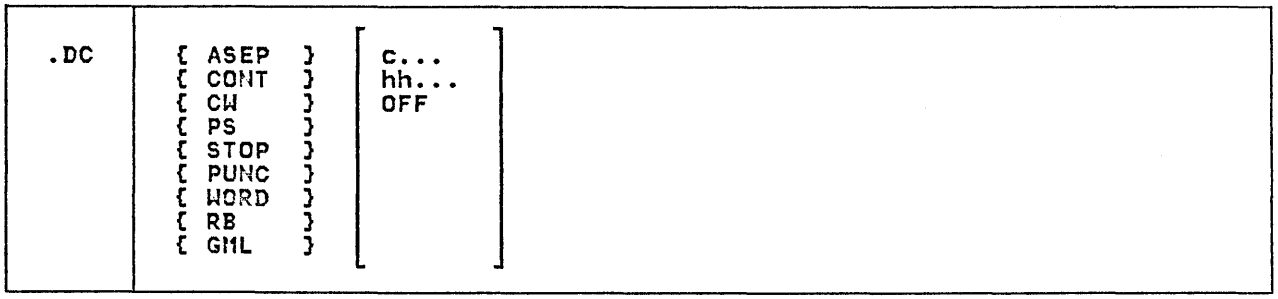
```
This is a line.&2col.Now start 2 columns.
```

Has the same effect as the sequence:

```
This is a line.
.cd 2 0 46
.cl 43
Now start 2 columns.
```

.DC [DEFINE CHARACTER]

Use the .DC [Define Character] control word to define various special characters that the formatter will recognize as having a special significance.



Where:

c specifies the character (or characters) to be recognized. The character may be any single character.

hh specifies the character (or characters) to be recognized, expressed as a two-digit hexadecimal code.

If a parameter is given with no following character or hexadecimal code, then the character is restored to its initial setting. For the ASEP, PUNC, and WORD parameters, more than one character or hexadecimal code may be specified, separated by blanks. In this case, single characters and two-digit hexadecimal codes may be intermixed on the same control word line.

OFF causes the character to be undefined. If for example, .DC CW OFF is specified, then there is no control word separator.

ASEP allows the definition of up to four characters which are to be used to separate array elements when an array is substituted in a document using the &name(*) form. All characters to be used to separate array elements must be specified, including blank characters (as 40). The initial and default values for the ASEP characters are , 40.

CONT defines a continuation character for text lines. The formatter normally considers that no word may span input lines. Use of the continuation character defined with the CONT parameter allows words to span input lines. When the last character of an input text line is a continuation character, the normal interword blank is not added when this line is concatenated to the next, but existing blank characters preceding the line continuation character are retained.

If the formatter control or text which follows the line continuation character causes a break, continuation is cancelled for that line. A null line also cancels continuation for the previous line.

The line continuation character is recognized at the end of a line, whether the line contains text or control words, or a mixture of both. The continuation character may not be used to extend a control word line, but it may extend the text data that is associated with that control word. There is no default line continuation character.

CW specifies the character to be used to separate control words on a single line. The default control word separator character is the semicolon (;). If the specified control word separator character is hexadecimal 00, the control word separator will be undefined. The effect is the same as if .DC CW OFF were specified.

STOP specifies the characters to be recognized as end of sentence characters when formatting for non-3800 logical devices. If any of the STOP characters occurs at the end of an input line, or precedes a " or) at the end of a line, and the line is not the last before a break, SCRIPT/VS will insert an extra blank before concatenating with the following input line. If the same character is defined as a continuation character and a stop character, its effect as a continuation character will be used if it occurs at the end of the line.

PUNC specifies the characters to be recognized as punctuation for spelling checking. Punctuation characters are defined as those characters which, when occurring in a word, will be retained when the word is checked against the dictionary, but when they occur at the end of a word, they will be removed before checking takes place. The default punctuation characters are the hyphen (hexadecimal 60), and the apostrophe (hexadecimal 7D). Punctuation characters given with this option will add to the currently defined default characters. The control word .DC PUNC OFF will clear the entire list, so that no punctuation characters are defined for spelling verification.

PS specifies the character to be used as the page number symbol. It may be any character other than blank. The default page number symbol is ampersand (&). Every page number symbol in running titles (.RT), running headings (.RH), and running footings (.RF) is replaced with the current page number every time the running title, heading, or footing is formatted to be placed on a new page.

WORD specifies the delimiters to be used in the recognition of words for spelling verification. The default word delimiters are shown in Figure 35 on page 319.

The characters given with this option will add to the currently defined word delimiter characters. The end of a line will always be recognized as a word delimiter unless the line continuation character is used. If .DC WORD OFF is specified, only the period (.) and blank will be recognized as word delimiter characters.

RB defines the character to be used as a required blank. Required blanks are not recognized as interword blanks for formatting, but they are translated to ordinary blanks after formatting is complete. The initial and default required blank is hexadecimal 41. The current required blank character is always available in the system symbol '&$RB'.

GML defines the character to be used as the GML delimiter or alternate symbol delimiter. It may be any character that is not allowed in a symbol name, except blank, period, or ampersand. That is, the GML delimiter may not be set to the characters blank, period, ampersand, a-z, A-Z, 0-9, or the characters #, @, and $. A .DC GML control word that attempts to set one of these characters as the GML delimiter causes an error message, except .DC GML 40, which is equivalent to .DC GML OFF.

Default: Restores the initial setting for specified character.

Notes:

- The .DC CW control word has the same effect as the .CW [Control Word Separator] control word, except that with .DC, you do not have to use the actual character specified on the control word line; you can specify it as a two-digit hexadecimal code. This capability is useful to prevent misinterpretation of the control word separator character in cases where it is already set to the value specified on the control word. See the discussion of the .CW [Control Word Separator] control word for additional information.
- The .DC PS control word has the same effect as the .PS [Page Number Symbol] control word. See the discussion of the .PS [Page Number Symbol] control word for additional information.

Examples:

1. Continuation Character

   In the following examples, the plus sign is used as the continuation character. The continuation character may not occur in the middle of a control word line. For example,

```
this is
.up part+
ially uppercase
```

   Results in:

```
this is PARTially uppercase
```

   The continuation character will, however, allow the user to create one "logical" line from a number of input lines. For example:

```
.ce 1
this is a sin+
.up gle line
```

   Will result in the line:

```
this is a sinGLE LINE
```

2. PUNC and WORD

   Note that there is only one delimiter table to hold both punctuation and word delimiter characters. The latest specification for a character will be that in effect. For example:

   .dc word +

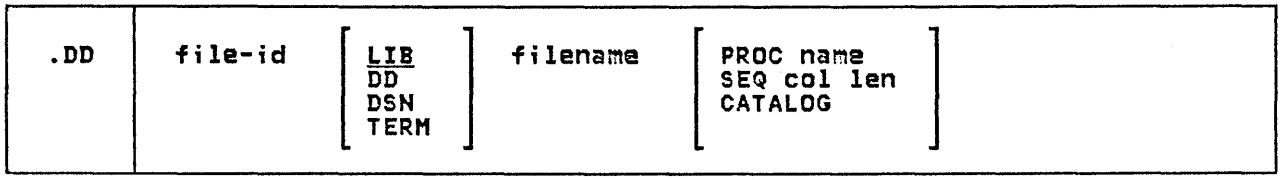
   will cause the + character to be recognized as a word delimiter character.

   .dc punc +

   will cause + to be recognized now as a punctuation character.

.DD [DEFINE DATA FILE-ID]

The .AP [Append] and .IM [Imbed] control words require a file-id for the file to be imbedded or appended. This file-id is an internal SCRIPT/VS name for a file or data set in the host environment in which SCRIPT/VS is executing. The .DD [Define Data File-id] control word allows you to associate a one to eight character internal SCRIPT/VS file-id with a real file or data set identifier. If no .DD has been issued for a file-id, a valid identifier is constructed from the file-id, based on assumptions and rules established for each operating environment.



Where:

file-id is a one- to eight-character internal SCRIPT/VS name for the file being defined. An error will result if file-id corresponds to the id of a file which is already in use as an imbedded file. If the file-id corresponds to a file that was previously read and terminated by a .EF [End of File] control word, that original file is closed before the redefinition is made. This file-id is described with the options below.

LIB is the default, and indicates that the file to be referred to exists in the library of the environment in which SCRIPT/VS is operating.

In CMS, the "filename" given is a normal CMS filename, followed optionally by a filetype and a filemode.

In the batch environment, LIB indicates that the filename is a Document Library Facility file, whose filename may be followed optionally by a library number and password.

In TSO, the LIB option indicates that the filename refers to a PDS member whose data set name is specified by the SEARCH option of the SCRIPT command.

DD specifies that file-id refers to a DD name that is specified in filename. This option is applicable only in the TSO and VS2 batch environments. Use of the DD option implies that the user has supplied a JCL DD card with a ddname of "filename", or preallocated the data set by use of the TSO ALLOCATE command.

DSN specifies that file-id refers to a fully or partially qualified data set name specified in filename. This option is applicable only in the TSO and batch environments.

TERM specifies that, for file-ids DSMTERMI and DSMTERMO only, the input or output is to be restored to the terminal. This option is useful when the terminal input or output has been changed with a previous .DD control word for either of these file-ids.

filename specifies the actual name of the file which is to be given the specified file-id.

PROC specifies that the file is to be processed with the attribute processor whose name is

given. This option is applicable only in the batch environment.

name is the name of the attribute processor to be invoked.

SEQ (batch only) specifies whether the records in the external data set to be read contain sequence numbers, and if so, describes them. If SEQ is not specified, SCRIPT/VS assumes that the records contain no sequence numbers. SCRIPT/VS will generate sequential line numbers for the read process.

If SEQ is specified, "column" and "length" define the starting position and the number of characters (maximum of 8) for the sequence number. The sequence numbers must be of fixed length with a maximum length as defined by the length parameter. Nonnumeric characters may precede the numeric values.

CATALOG specifies that the data set to which the file-id refers is to be cataloged when it is is closed. This option is valid only for SCRIPT/VS utility files (see Usage Notes) when used with the DSN option in TSO to create a new data set. In all other cases it is ignored. CATALOG is especially useful when creating output files with the .WF control word. It is possible to create many different .WF files by specifying a different data set name with .DD. Normally, these data sets would be deleted when closed.

Default: LIB

Notes:

- If the .DD control word is used with a parameter that is not allowed in the formatter's operating environment, a message will be issued and the control word will be ignored.
- The file-id PROFILE is used for the profile specified with the PROFILE option of the SCRIPT command.
- SCRIPT/VS has a number of file-ids that are used by the system. Some of these may be the subject of the .DD control word. These file-ids are:

  DSMTERMI - The terminal input
  DSMTERMO - The terminal output
  DSMUTDIM - The .DI file
  DSMUTMSG - The message file
  DSMUTTOC - Table of contents file
  DSMUTWTF - The .WF file

  For example, if the file-id DSMTERMI is associated with a disk file, then whenever a .RV or .RD is processed the data will be read from the specified file. This capability is of particular use in the batch environment.

  Whenever any of these file-ids is the subject of a .DD control word, the host system file name associated with the existing definition of that file-id is closed, and in TSO, it is also deallocated.

Examples:

1. To give a file-id of "file1" to a Document Library Facility file named "title", with a password of "p2301" which exists in library "13425", the .DD control word statement would be:

   ```
   .DD file1 LIB 13425 title/p2301
   ```

   In this example, the option LIB could have been omitted, as it is the default.

2. To give a file-id of "alpha" to member "mem3" of a partitioned data set named "userid.doc.text", the .DD control word statement would be:

   ```
   .DD alpha DSN doc(mem3)
   ```

3. In CMS, if there are two SCRIPT files called "names", one on your A-disk, and the other on your C-disk, the control word .IM NAMES would ordinarily imbed the one on the A-disk, following CMS search order. The file on the C-disk would be imbedded if the following .DD were in effect:

   ```
   .DD names names script c
   ```

   Note that in this example the keyword LIB was omitted because it is the default.

.DH [DEFINE HEAD LEVEL]

Use the .DH [Define Head Level] control word to override the default characteristics of the head levels that are generated with the .H0 - .H6 [Head Level 0 - 6] control words.

.DH	n [options]

Where:

n is the number of the head level to be defined. It may be a number from 0 to 6.

options are keywords that indicate how to change the definition of head level n. If no options are given, the default characteristics of the head level are restored. The options recognized are:

SKBF v v is the amount of space to be skipped before the head level.

SPAF v v is the amount of space to be skipped after the head level.

TCIN h h is the amount the table of contents entry associated with the head level is to be indented.

TO table of contents entry only - no head printed in the text.

NTO no "TO".

TC table of contents entry wanted.

NTC no "TC".

TS space before table of contents entry.

NTS no "TS".

US underscore the head level.

NUS don't underscore it.

UP put the head level in uppercase.

NUP don't put it in uppercase.

OJ outjustify the head level (this means right adjust it if it falls on an odd-numbered page).

NOJ don't outjustify the head level.

PA do a page eject before the head level if necessary (if not already at the top of a page).

NPA no page eject.

BR do a break after the head.

NBR no break.

FONT fontname specifies the font name of the font to be used for the heading or OFF. If OFF is specified, the previous specification of FONT will be set off. If the fontname and OFF are omitted and the FONT option is the last one specified, no syntax error will result. This is useful when using the &$CHAR system symbols as the font names, since these symbols have null values when there is no corresponding font.

Default: Restores the initial setting.

Notes:

- The .DH [Define Head Level] control word allows a maximum of 14 options on the line. If you wish to change more head level variables than can be done with 14 options, you must do it with more than one .DH control word. Each time .DH is processed, only those variables specified are changed. All other variables remain the same.
- If a head level control word is processed that causes an entry in the table of contents, the table of contents entry is saved with the specifications that are in effect at the time that head level is processed. If you change the definition of that head level later, the new definition only affects later occurrences of that head level control word.
- For a list of the default characteristics associated with the heading levels 0 through 6, see the discussion of .H0 - .H6 [Head Level 0 - 6].
- The .H0 - .H6 [Head Level 0 - 6] control words are actually implemented as macros, and the .DH control word merely causes these macros to be defined or updated. These macros are named DSMSTDHn (or DSMEZSHn if EasySCRIPT is in effect) and can be seen by use of

the .IT [Input Trace] control word. The macro for a given head level does not exist until that head level control word has been processed for the first time, or until a .DH for it has been processed.

- The use of .EZ ON will cause a new set of heading macros to be used. Therefore, each time .EZ ON is used, you must respecify the head definitions that you want to use.
- If you want to add a function to a head level control word that is beyond the scope of .DH [Define Head Level], you may add to the macro using the .DM [Define Macro] control word.

.DI [DELAY IMBED]

Use the .DI [Delay Imbed] control word to defer the inclusion of a portion of a SCRIPT file until the next page eject occurs.

.DI	[1 / n / ON / OFF / line]

Where:

n specifies the number of lines to be delayed. If omitted, 1 is assumed.

ON starts an open-ended delayed imbed. Subsequent lines, until a .DI OFF is encountered, are included in the delay imbed file.

OFF ends a delayed imbed, whether it was started with .DI ON or with a specified n that has not been exhausted.

line is an input line that is to be delayed.

Default: 1

Notes:

- The .DI [Delay Imbed] control word is especially useful for positioning diagrams and tables. The next n lines of the current SCRIPT file are saved in a temporary file called DSMUTDIM. When the top of the next output page is reached, this temporary file is imbedded and processed by SCRIPT/VS. After the inclusion of the saved lines, normal processing resumes. If there is a keep to be processed in the next page, then a further page eject is done after the keep before the delayed imbed is processed.
- This control word does not act as a break.
- An automatic page eject is not performed at the end of the inclusion. If you want SCRIPT/VS to resume normal processing on a new page, you should end the delayed section with a .PA control word.
- The .DI OFF control word must begin in column 1, not after a control word separator. When SCRIPT/VS is processing a delay imbed it is not processing input lines except to look for .DI OFF on a line by itself.
- No .DI control word is put into the delay imbed file.

Examples:

1. To delay the inclusion of one line:

```
.di .pa
```

   The single line ".pa" is written into the delay imbed file. At the end of the current page, a blank page, except for top and bottom titles, is generated. Output resumes on the page after the blank page.

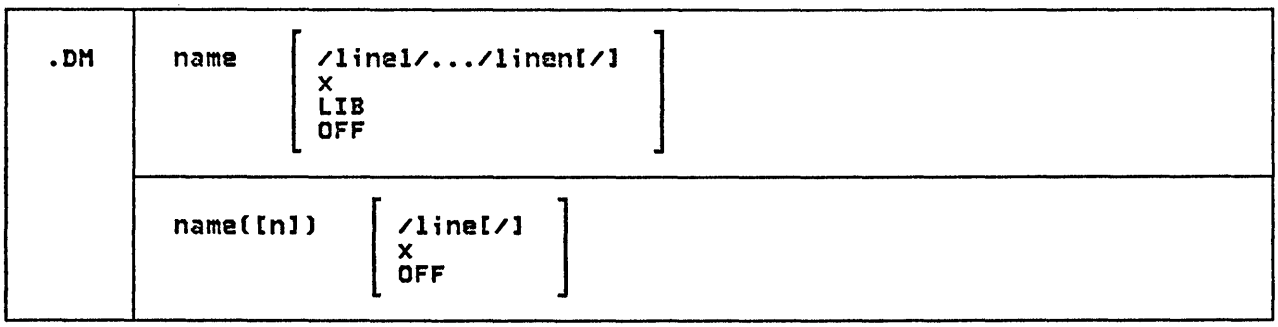
2. To include a figure at the top of the next page:

```
.di 3
.sp .5i
.im figure5
.sp 5
```

   The current page is processed as if the .DI and the three following lines had not existed. At the top of the next page, the three lines are processed. This results in spacing a half-inch, imbedding the file named FIGURE5, followed by spacing five lines.

.DM [DEFINE MACRO]

Use the .DM [Define Macro] control word to establish macro definitions for sequences of SCRIPT/VS control words or text lines. SCRIPT/VS macros are invoked by preceding them with periods, as SCRIPT/VS control words. No macro substitution is performed unless the .MS [Macro Substitution] control word has been processed to turn macro substitution ON.



Where:

name is the symbolic name you want to assign to the macro, so that you can invoke it with the control line:

.name

It may contain a maximum of 10 nonblank characters which may be upper- and lowercase alphabetic, numeric, and the characters @, #, and $.

name(n) indicates that the line that follows is to be stored as part of the macro definition in line n. By this means, multiple line macros may be defined. The values for n need not be sequential when the macro is defined, but if the same value for n is given on two uses of the .DM control word, only the latest value for the line will be stored. They are executed in numerical sequence. When a line number is given with the name, only one line of the macro may be given. Each line of the macro is defined with a separate .DM control word. (n) must follow the macro name without intervening blanks, and must be a positive integer or zero. If n is omitted, that is, "name()" is specified, macro elements are assigned with line number increments of 10. Macro element zero has the same significance as array element zero and can be assigned a number (using the .DM control word) which will control the start of automatic line number assignment. If you set macro element zero to other than a valid number, the value that you set will be lost. It will never be executed with the rest of the macro.

/ is any character used to delimit the individual lines in the macro. The final delimiter may be omitted.

line is any SCRIPT/VS control word line or line of data that you want to include in the macro definition. It may contain symbolic names, or any of the special macro variables &*, or &*1 through &*n (see Usage Notes). If line is omitted, the macro (or macro line if n is given) is stored as a null macro or macro line.

X indicates that you want the current value of a macro or macro line assigned to the symbol &x. x may be any single alphameric character. (If you give two or more characters, SCRIPT/VS treats the first as a delimiter and the others as a line to be inserted in the macro definition.)

LIB causes the macro to be defined by retrieving its value from a library. The name of this library may be defined using the LIB option of the SCRIPT command. If LIB is used to define a macro, the definition retrieved from the library completely replaces the current definition (if one exists). If LIB is specified, but no definition with the macro name given exists on

the library, the macro will be undefined. Since macro names are in uppercase only, names are folded to uppercase before the library is accessed. The LIB parameter sets up an entirely new macro definition; no line number may be given with the macro name. The LIB option may be used independently of the .LY [Library] control word.

OFF deletes a macro definition or a line from a definition.

Notes:

- The following symbols have special meanings within macros:

  &*: is the line passed to the macro when it is invoked. Thus, if the macro defined with:

  ```
  .dm typit() /.ty ***
  .dm typit() /.ty &*
  .dm typit() /.ty ***
  ```

  is invoked with the line

  ```
  .typit Hello!
  ```

  then the symbol &* has the value "Hello!". The processing of this macro results in the lines:

  ```
  ***
  Hello!
  ***
  ```

  being displayed at your terminal.

  &*0: Contains the number of tokens passed when the macro is called. Using the above example, the value of &*0 is 1.

  &*1 - &*n: Are the tokens passed to the macro when it is called. You can pass as many tokens to a macro as will fit on the input line. If the .typit macro is invoked

  ```
  .typit Processing section 5...
  ```

  then &*1 has a value of "Processing", &*2 has a value of "section", and &*3 has a value of "5...". The value of &*0 is 3.

- Macro calls are treated as invalid control words if you do not use the .MS [Macro Substitution] control word:

  ```
  .ms on
  ```

- Symbol names that are used in a macro definition are substituted at the time the .DM control line is processed, if substitution is on. If you want to use variable symbols in a macro to be substituted at execution time, you must use the control word

  ```
  .su off
  ```

  before defining the macro with the .DM control word.

- Values for the symbols &*1 through &*n are established whenever a macro is invoked. These values are local to the current level of macro invocation.

- A macro name may be the same as the two-letter name of a control word. Such a macro effectively redefines the control word by getting control whenever the control word is encountered.

- Macros may be invoked recursively. In order to avoid looping situations for recursive invocation SCRIPT/VS keeps invocation counts for macros. Any given macro may not be opened more than 99 times, and no more than 255 macros of any name may be open at the same time. If either of these situations occurs, a severe error message is issued and processing is terminated.

- If macros are defined with multiple macro lines on a single line of input, the macro will be stored as if it had been entered on separate lines with an increment of 10, and the new definition will completely replace any existing definition with the same name. However, subsequent macro lines defined using sequence numbers will behave as if all lines had been added in this way.

- Macros defined using sequence numbers may be defined using sequence numbers in any order. However, the macro will be executed as if the lines had been entered in sequence. Macro lines may be redefined at any time within a document, or lines added or inserted into an already existing macro definition. This addition or redefinition of lines will take place based on the sequence number specified.

- If an entire macro is assigned to a symbol "x" it will be stored in the form:

  ```
  #line1#line2#line3#linen#
  ```

  where # represents a separator character of hexadecimal FF. If only a single line of a macro is assigned to a symbol "x", it will be stored in the form #line#. If you want to print this symbol, the .TR control word must be used to convert this character to one that is available on the printer being

used, if this is required. When using the symbol assignment capability, remember that the maximum length for a symbol is 256 characters.

The symbol assignment capability can be used to test the existence of a macro or a macro line, as follows:

- if the macro (or macro line) does not exist, the symbol "&x" is assigned a null value (&L'&x=0).
- if the macro (or macro line) does exist, but has a null value, the symbol "&x" is assigned a value of hexadecimal FFFF, which is two consecutive separator characters (&L'&x=2).
- else, the symbol "&x" will have the value of the complete definition of the macro (or macro line) as described before (&L'&x>2).

- Use of the LIB option of the .DM control word allows a macro definition to be explicitly retrieved from the library. Use of the .LY control word allows macro definitions to be retrieved from the library when a macro is used in a document where a definition for it does not currently exist.
- Head level control words are actually SCRIPT/VS macros; you can define macros to perform the functions of head levels, or you can add to the existing macro definitions for the head levels to add function not within the scope of the .DH [Define Head Level] control word.

.DS [DOUBLE SPACE MODE]

Use the .DS [Double Space Mode] control word when you want your output to be double-spaced.

.DS	

Notes:

- This control word does not cause a break.
- The .DS control word doubles the line spacing set by the .SL control word. When double-spacing is in effect, each space or skip caused by a .SP or .SK control word is doubled (thus, .SP 2 yields four spaces). However, if the .SP or .SK control word indicates "absolute" spaces, the space count is not doubled.

Example:

```
.DS

Blank lines are inserted between output
lines below this point in the file, as
shown in these few lines.
```

.DU [DICTIONARY UPDATE]

Use the .DU [Dictionary Update] control word to add or delete words from the addenda dictionary which may be used in addition to the main dictionary for hyphenation and spelling verification. The changes to the dictionary which are specified using this control word are in effect only during the formatting of the current document.

.DU	{ ADD } word ... word { DEL }

Where:

ADD specifies that the word or words given with the control word are to be added to the addenda dictionary.

DEL specifies that the words given with the control word are to be deleted from the addenda dictionary.

word is a string of blank delimited words.

Notes:

- A .DU control word which requests that a word be added to, or deleted from, the addenda dictionary where that word is already in the addenda dictionary (for ADD), or not in the addenda dictionary (for DEL), will not cause an error message. The first ADD for a word will put the word in the dictionary, and all subsequent ADDs will be ignored. The first DEL for a word will delete the word from the dictionary, and all subsequent DELs will be ignored. You should be careful to avoid multiple ADD or DEL situations where a word may get added and perhaps also deleted in an imbedded file.
- For a description of the function and use of the formatter's spelling checking and hyphenation capabilities, see the discussion in "Chapter 1. An Introduction to SCRIPT/VS" on page 1.
- Words added to the dictionary using the .DU control word may include hyphens. In this case, the hyphens indicate potential hyphenation points for the word. Whenever hyphenation is in effect (specified by the .HY control word) these hyphenation points will be used unless the .HW control word has been used for the specific occurrence of the word, or use of the addenda dictionary has been suppressed with the NOADD option of the .HY control word.
- Words that contain hyphens, such as lighter-than-air, should be supplied to the .DU control word with double hyphens at these hyphen points, as described for the .HW control word.
- Words that contain word delimiters will be added to the addenda dictionary with the delimiters intact. You can redefine these delimiters with the .DC control word.
- Stem processing is used for verification against both the main and the addenda dictionaries when requested using the .SV control word.
- Words may be added to the addenda dictionary even when spelling verification is off, or is in effect against the main dictionary only.

.EB [EVEN PAGE BOTTOM TITLE]

The .EB [Even Page Bottom Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the bottom of the current page, if it is even-numbered, and each subsequent even-numbered output page.

.EB	[n] /part1/part2/part3/

Where:

n is the number of the bottom title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even bottom title 1" sets the same storage buffer as "even top title 6." See the discussion of the .FS [Footing Space] control word for information on how to allocate space on your output page for bottom titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any character that does not appear in part1, part2, or part3.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RT [Running Title]. See the discussion of the .RT control word for further information about running titles, including those for the bottom of even pages.

.EC [EXECUTE CONTROL]

The .EC [Execute Control] control word is used to cause SCRIPT/VS to execute the given line as a control word line, even if there is a macro defined with the same name, and macro substitution is ON.

.EC	control word line

Where:

control word line is a SCRIPT/VS control word line.

Notes:

- Use the .EC control word whenever you want to cause SCRIPT/VS to execute a control word even when a macro is defined with the same name. The .EC control word is useful within macros that have the same name as control words. Often, a macro that "redefines" a control word uses the control word function in addition to whatever other function it performs. In these cases, if the .EC function were not used, the same macro would be repeatedly invoked in a loop until SCRIPT/VS terminated it with a severe error message. Of course, macro substitution could be turned OFF, but that would prevent any other macro from being invoked until macro substitution was turned ON again.
- The control word modifier provides an implied .EC function. It also prevents the control word separator scan on that control word line. The control word modifier may be used with any control word; it consists of a single quote (') between the period and the name of the control word. (.'ce center this line).

Examples:

1. To define a macro called .IM to replace the .IM control word without using the .EC control word would require the following macro definition:

```
.dm im(1) /.ty &*/
.dm im(2) /.ms off/
.dm im(3) /.im &*/
.dm im(4) /.ms on/
```

In this example, macro substitution needs to be turned off to avoid an infinite macro substitution loop. Unfortunately, this has the effect of turning off macro substitution for the imbedded file, and all files that it imbeds. In this situation, the .EC control word should be used:

```
.dm im(1) /.ty &*/
.dm im(2) /.ec .im &*/
```

The control word modifier may be used in the same way:

```
.dm im(1) /.ty &*/
.dm im(2) /.'im &*/
```

The difference between the .EC form and the control word modifier form is that the .EC line is scanned for control word separators, while the control word modifier line is not. In this example, there is no difference between the two; the original .IM macro line will already have been scanned for separators.

2. The .EC control word will issue an error message if the subject control word line is not a valid control word line. To be a valid control word, it must start with a period, followed by two characters and a blank. (A line without a period in column 1 is usually treated as text, but, as the subject of .EC, it is treated as an invalid control word.)

3. The .EC control word will issue an error message if the control word line given is "valid", but refers to a non-existent control word, even if a macro exists with the control word name given.

.EF [END OF FILE]

The .EF [End of File] control word simulates the end of the current file.

.EF	[CLOSE]

Where:

CLOSE tells SCRIPT/VS not to hold your place in the current file, but to close it, so that the next time the file is imbedded, SCRIPT/VS begins processing at the top of the file, not at the line following the .EF control word.

Notes:

- The .EF [End of File] control word causes an end of file condition to be simulated on the current input file. If the current input file is not an imbedded file (see the discussion of the .IM [Imbed] control word), all processing is terminated. If the current input file has been imbedded, the .EF control word causes input processing to continue in the outer file. In this latter case, SCRIPT/VS remembers the position of the .EF control word; if the file is imbedded again, then SCRIPT/VS begins reading at the line following the .EF control word instead of the beginning of the file, unless the CLOSE operand is used.

- The .EF control word, in conjunction with the .IM [Imbed], .AP [Append], and .QU [Quit] control words, provides an easy and flexible mechanism for producing simple tables, as demonstrated in the example below.

Example:

In this example, a table is generated using two files. One file contains a single line that defines a table format and contains symbolic names for the table entries. The other file contains .SE [Set Symbol] control words to define the values of the actual entries in the table. This method of generating tables allows the format of a table or the contents of the table to be separately altered or updated.

Consider the following two SCRIPT files.

File: TABLE

```
.tb 3M 21M
.cs 2 on
.cs 1 ignore
.sp 2
.fo off
.bx 1 19M &$cl
.se bxoff=
.cs 2 ignore
.cs 2 off
.im tablsym
&$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
```

Now, 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 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 before the last 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

.EM [EXECUTE MACRO]

The .EM [Execute Macro] control word is used to cause SCRIPT/VS to execute the given line as a macro line even if there is a control word with the name given, and macro substitution is OFF.

.EM	**macro line**

Where:

macro line is an input line that invokes a SCRIPT/VS macro.

Notes:

- Use the .EM control word whenever you want to cause SCRIPT/VS to execute a macro when macro substitution is OFF. The .EM control word is useful when a control word must be replaced with a macro of the same name and macro substitution cannot be turned on.
- If the .EM control word specifies a macro for which no valid macro definition exists, it is treated as an invalid control word, even if there is a control word of that name.
- The control word modifier provides an implied .EC [Execute Control] function, and also prevents a control word line from being scanned for control word separators. If you want to prevent a macro line from being scanned for separators, you can use the control word modifier for the .EM control word:

  ```
  .'EM .mymac A;B
  ```

  The control word that is modified here is .EM, and this usage allows the macro 'mymac' to be executed, while preventing the data for the macro (A;B) from being misinterpreted as containing a control word separator.

.EP [EVEN PAGE EJECT]

Use the .EP [Even Page Eject] control word to cause either one or two page ejects, such that the new page is even-numbered, regardless of whether the current page is even- or odd-numbered.

.EP	[ON / OFF]

Where:

ON specifies that subsequent text is to be printed only on even-numbered pages. Odd-numbered pages are left blank, except for top and bottom titles, if any.

OFF resumes processing so that text appears on odd- and even-numbered pages.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .PA EVEN [ON|OFF].

.ET [EVEN PAGE TOP TITLE]

The .ET [Even Page Top Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the top of the current page, if it is even-numbered, and each subsequent even-numbered output page.

.ET	[n] /part1/part2/part3/

Where:

n is the number of the top title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even bottom title 1" sets the same storage buffer as "even top title 6." See the discussion of the .HS [Heading Space] control word for information on how to allocate space on your output page for top titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any character that does not appear in part1, part2, or part3.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RT [Running Title]. See the discussion of the .RT control word for further information about running titles, including those for the top of even pages.

.EZ [EASYSCRIPT]

EasySCRIPT is an early implementation of GML that existed in SCRIPT/370. The .EZ [EasySCRIPT] control word provides automatic formatting functions used by EasySCRIPT. These functions are available through a set of EasySCRIPT "tags" or through the .EZ control word directly. The EasySCRIPT tags are symbols that substitute to the appropriate .EZ control word. They are not true GML tags; they are delimited with the ampersand (&), not the GML delimiter (:). EasySCRIPT tags are included in SCRIPT/VS to allow documents already marked up with them to be processed by SCRIPT/VS.

.EZ	{ ON [headnum] } { OFF } { function line }

Where:

ON initializes the EasySCRIPT tags that provide the EasySCRIPT numbering, paragraphing, and heading functions. The names of the tags are the same as the parameters of the .EZ control word that provide the associated function. ON also switches the head level definitions from the standard ones to another set used only while EasySCRIPT is in effect. The .DH [Define Head Level] control word operates on whichever set of head levels (standard or EasySCRIPT) is currently in effect.

headnum is the decimal number of the last heading that would have been used. EasySCRIPT uses this number to set the counter it uses for numbered headings. If not specified, 0.0.0.0 is assumed. If you specify &xref, EasySCRIPT resumes numbering where it left off when .EZ OFF was last processed. (&xref is the symbol EasySCRIPT uses to keep track of the current heading number.)

OFF cancels the EasySCRIPT tags, so that they are not recognized by SCRIPT/VS. OFF also switches the head-level definitions back to the standard set.

function is the name of the EasySCRIPT function to be invoked. The line of text data that follows the function name is processed by the built-in function requested. The functions are summarized below. The names of the functions are case sensitive. For example, there are two different bulleted list functions: the "B" function, in uppercase, starts a regular bulleted item, and the "b" function, in lowercase, starts a sub-bulleted item.

line is an input line of data. It must be separated from the function name by at least one blank.

Notes:

- EasySCRIPT functions provide a fast, convenient way of formatting text and documents, particularly those that require decimal numbering. EasySCRIPT provides automatic numbering for heading levels and lists, if requested.
- The names of the EasySCRIPT functions are the same as the names of the tags set up by ".ez on". For example, the "N3" function identifies a numbered list item at level 3. This function can be invoked with the control word

  .ez N3 text of the numbered item

  or with the tag

  &N3.text of the numbered item

  but the latter is enabled only after .EZ ON has been processed.
- The symbol "&xref" contains the entire number of the current heading level, when EasySCRIPT's automatic numbering scheme is used. The symbols "&xref1", "&xref2", "&xref3", and "&xref4" contain the components of this number. For example, if "&xref" has the value "1.0", then "&xref1" will have the value "1", and "&xref2" will have the value "0".
- The EasySCRIPT functions are summarized below. Note the differences in the uppercase and lowercase versions of a function name:

EasySCRIPT Functions	
Hx	Begins a decimal numbered heading of level x (1 through 6).
hx	Begins an unnumbered heading of level x.
P	Begins a major paragraph by resetting the current indention.
p	Begins a minor paragraph at the current indention.
Nx	Begins a numbered item of level x (1 through 4).
nx	Begins an unnumbered item of level x (1 through 4).
B	Begins a bulleted item.
b	Begins a sub-bulleted item.
toc	Generates a table of contents.

.FM [FOOTING MARGIN]

Use the .FM [Footing Margin] control word to specify how much space to skip between the last line of text, on a full page, and the bottom titles, overriding the initial setting established for the device.

.FM	[v | +v | -v]

Where:

v specifies the amount of space to be skipped between the last line of text and the footings (bottom titles). If +v or -v is specified, the current value of the footing margin is incremented or decremented. If no .FM control word is used in the file, or if the .FM control word is used with no operand, the initial value is used. The minimum value that may be specified for the footing margin is 0. If a negative result is calculated for the footing margin, the value will be set to zero, and a message will be issued. The maximum value that can be used for the footing margin is equal to the bottom margin (.BM) minus the footing space (.FS).

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

- The bottom titles are placed a specified amount of space below the last line of text. The location of the last line of text is explicitly defined by the .BM [Bottom Margin] control word, whether that line is actually filled or not.
- This control word does not cause a break.
- The .FM control word will take effect on the page after it is encountered.

Example:

```
.fm .5i
```

A half-inch of space is left between the last line of text and the running bottom titles, if any have been defined.

.FN [FOOTNOTE]

Use the .FN [Footnote] control word to set aside up to ten lines of formatted output text to be positioned at the bottom of the current page, if possible, or at the bottom of the next page.

.FN	{ ON } { LEADER } { OFF }

Where:

ON marks the beginning of the material in the footnote.

LEADER allows the specification of up to 10 lines of leader to be placed at the top of the footnotes on the page to separate the footnotes from the text of the page. The initial leader is a space of one line and 16 horizontal box characters.

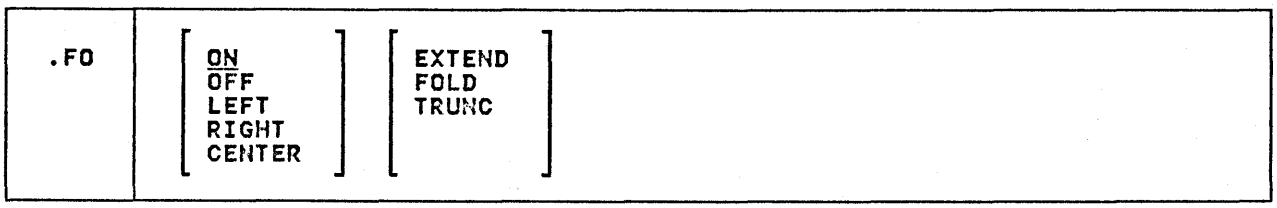
OFF marks the end of the footnote material.

Notes:

- .FN ON starts a footnote. All lines until the subsequent .FN OFF control are put in the footnote. If .FN OFF is encountered when no footnote is in process, it is ignored. If the maximum number of lines is exceeded, a message is issued, and the remaining lines until .FN OFF are discarded.
- The first footnote in a page is automatically started with a leader which may be redefined with .FN LEADER and .FN OFF.
- The .FN control word does not act as a break. The footnote is considered to fit on this page if the number of lines left is sufficient to accommodate the footnote itself plus the leader. To ensure that your footnote and the callout appear on the same page, put the footnote itself before the callout. Since the footnote does not cause a break, the sentence containing the callout may be interrupted for the footnote itself without disrupting the formatted output.
- A keep and a footnote may not be in process at the same time. The control words that are disallowed in a keep are also disallowed in a footnote. A footnote in process is terminated by any disallowed control word.
- A footnote does not have any automatic offset. You must include an OFFSET (.OF) control word if you want the footnote offset.
- Footnotes will run across the page in a single column. The line length may be changed in the footnote.
- When the footnote is started, offsets are cleared, and indention is set to the current .IN [Indent] value, without any added .OF [Offset] value. The column width is set to the line length. Any changes within the footnote to the indention, font, or certain other values in the formatting environment, are automatically restored when the footnote ends, to the values in effect before the footnote started. See Figure 32 on page 315 for a list of all the values that are automatically saved and restored around a footnote.

.FO [FORMAT MODE]

Use the .FO [Format Mode] control word to cancel or restore concatenation of input lines and justification of output lines. The .FO control word also controls whether lines may be allowed to extend beyond the column boundary.



Where:

ON restores default SCRIPT/VS formatting, including both justification and concatenation of lines. If the .FO control word is used with no operands, ON is assumed.

OFF cancels concatenation of input lines and justification of output lines. Subsequent text is printed "as is." If an input line is longer than the defined line length, the line may be allowed to extend beyond the right margin, and no message will be issued.

LEFT specifies that input lines are to be concatenated but not justified. The resulting output lines are left-aligned in the column. This format is sometimes called "ragged-right."

RIGHT specifies that input lines are to be concatenated but not justified. The resulting output lines are right-aligned in the column.

CENTER specifies that input lines are to be concatenated but not justified. The resulting output lines are centered in the column.

FOLD specifies that if an input line will not fit in the output column (in .FO OFF mode), it is to be broken and the remainder placed on the next output line(s). The line is broken at the last character that will fit on the column.

TRUNC specifies that in .FO OFF mode, the line is to be truncated at the last character that will fit in the column.

EXTEND specifies that in .FO OFF mode, if a line will not fit in the column, it is allowed to extend beyond the column width. This is the initial setting.

Initial Setting: ON EXTEND

Default: ON

Notes:

- The .FO control word is a shorthand way to specify the two control words .CO [Concatenate Mode] and .JU [Justify Mode]. The effect is the same as if these two control words were specified, except that the .FO control word will end centering or right adjust mode whereas the .CO and .JU control words will not. When format mode is in effect (.FO ON), lines are formed by shifting words to or from the next line (concatenation) and padding with extra space to produce an aligned right margin (justification).
- This control word acts as a break.
- Even when format mode is in effect, a line may exceed the current column width. This can happen if there is only one word on the line and this word is longer than the column width, and also if a word follows a tab and spans the right column boundary. The setting of the TRUNC, FOLD, or EXTEND option controls how these situations are handled.
- Note that the TRUNC, FOLD, and EXTEND options may be specified as the only options of the .FO control word. In this case, the current formatting mode will be unchanged although a break will be done. For example, if .FO CENTER TRUNC is specified, and this is later followed by .FO EXTEND, the output will continue to be centered.
- Options may be specified in any sequence. If contradictory options are specifed, only the latest one will be used.

Examples:

1. .fo off

   Justification and concatenation

are
completed for
the preceding line or lines,
but following
lines are
typed exactly as they appear
in the file.

2. .fo

   Justification and formatting are resumed with the next input line. Output from this point on in the file is justified to produce aligned left and right margins on the output page.

3. .fo trunc

   If the current formatting mode is OFF, any lines that are longer than the current column width are truncated at the column boundary. If the current formatting mode is RIGHT or CENTER, any words that would extend past the right column boundary are truncated. If the current formatting mode is ON, the TRUNC option becomes meaningful only if the first word on a line or the first word after a tab would extend beyond the column width.

4. .fo center fold

   Lines are concatenated and centered, and any lines that are longer than the column width are folded onto the next line. Note that the FOLD mode of operation will continue in effect until explicitly changed. For example, another .FO control word with only the OFF option will leave FOLD in effect.

.FS [FOOTING SPACE]

The .FS [Footing Space] control word allocates space from the bottom margin area for running bottom titles.

.FS	[n] [+n] [-n]

Where:

n is the number of bottom title lines you want to appear on this and all subsequent output pages. This number may be from 0 to 6. If no number is given, 1 line is assumed. **n must be an integer from 0 to 6. This control word does not accept space units.** This number must be less than the bottom margin (.BM) minus the footing margin (.FM). If you specify +n or -n, the current value of the footing space is incremented or decremented accordingly. If the net result is a negative number, zero is assumed and a message is issued.

Initial Setting: 1

Default: 1

Notes:

- The .FS [Footing Space] control word allocates space from the bottom margin for bottom titles. You only need to use this control word if you want more than one bottom title in your document. If the bottom margin is not big enough to accommodate the footing space plus the footing margin, an error message is generated.
- This control word does not cause a break, and takes effect on the page after it is encountered.
- The running bottom title control words merely cause a title line to be saved in a storage area for future use. Only the first bottom title (bottom title 1) is used at the bottom of output pages by default. To get more than one title at the bottom of your formatted output pages you must do two things: define the titles using the .RT [Running Title] control word, and then allocate space for the titles by using the .FS control.
- If you do not want any bottom titles at all, the best way to accomplish this is to define the footing space as 0 (.FS 0). This is more efficient than setting the bottom titles to null (.RT B ////), because SCRIPT/VS does not have to process any titles to determine that none are wanted.

Example:

If you want three running bottom titles in your document, you could use the following sequence:

```
.rt b 3 /Chapter 4//&/
.rt b 2 ////
.rt b 1 $$&SYSMONTH./&SYSYEAR.$$
```

At this point, only bottom title 1, the one nearest the bottom of the page, is used on formatted output pages because the default footing space of 1 is still in effect. Now that the three title lines have been saved, the following control word causes SCRIPT/VS to print all three:

```
.fs 3
```

.GO [GOTO]

The .GO [Goto] control word causes SCRIPT/VS to branch to another part of the SCRIPT/VS input file or macro.

.GO	label

Where:

label is the name of a line set elsewhere in this file or macro using the ... [Set Label] control word.

Notes:

- Use the .GO control word to branch to another place in your SCRIPT file or macro. If the label designated on the .GO control word is not defined elsewhere in the file or macro, an error message is issued, and processing terminates.
- This control word does not cause an automatic break. The input line preceding the .GO control and the line at the label designated in the .GO control word are processed as though they were two sequential lines from the SCRIPT file.
- Every .GO control word must refer to a label defined with the ... [Set Label] control word; but you may have more than one .GO referring to the same label.
- .GO is particularly useful when performed conditionally as the subject of an IF statement. See the discussion of the .IF control word.

Example:

Suppose you had a SCRIPT file that was designed to recognize the variable SYSVAR5. In this example, if SYSVAR5 is set to SMALL, you want SCRIPT/VS to format the output at 36 lines per page and 4.2 inches per line. Otherwise, the default values are to be used. This could be done with the following control words:

```
.if &SYSVAR5 ne SMALL .go default
.pl 36
.ll 4.2i
...default
(etc.)
```

.HM [HEADING MARGIN]

The .HM [Heading Margin] control word specifies the amount of space to be skipped between the running top titles and the first line of the text area, overriding the initial value established for the device.

.HM	[v +v -v]

Where:

v specifies the amount of space to be skipped after the top title lines. If +v or -v is specified, the current value of the heading margin is incremented or decremented. If the calculated value of the heading margin is found to be negative, the value is set to zero and a message is issued. The maximum value that may be set for the heading margin is equal to the top margin (.TM) minus the heading space (.HS). If v is not specified, the default value for the logical device is restored.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

- The last running top title line is placed a specified amount of space above the first line of text. If no .HM [Heading Margin] control word is included in the file, the default value is used, as determined for the logical output device.
- This control word does not cause a break, and will take effect on the page after it is encountered.

Example:

```
.hm 3
```

Three lines are left between the running title lines and the first line of text. If a top margin of 6 lines is in effect, the last top title is printed two lines from the top of the page, followed by three more blank lines (the heading margin), and then the text.

.HN [HEADNOTE]

This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RH [Running Heading].

.HS [HEADING SPACE]

The .HS [Heading Space] control word allocates space from the top margin area for running top titles.

.HS	[n +n -n]

Where:

n is the number of top title lines you want on each subsequent output page. This number may be from 0 to 6. If no number is given, 1 is assumed. **The number must be an integer from 0 to 6. This control word does not accept space units.** The size of the top margin (.TM) minus the heading margin (.HM) must be large enough to accommodate the heading space specified. If +n or -n is specified, the current value for the heading space is incremented or decremented. If the net result is less than zero, the heading space is set to zero, and an error message is issued.

Initial Setting: 1

Default: 1

Notes:

- The .HS [Heading Space] control word allocates space from the top margin for running top titles. You need to use this control word only if the default value of one top title is not adequate for your document. If the top margin is not big enough to accommodate the heading space plus the heading margin, an error message is generated.
- This control word does not cause a break, and takes effect on the page after it is encountered.
- The .RT [Running Title] control word merely causes a title line to be saved in a storage area for future use. Only the first top title (top title 1) is used at the top of output pages by default. To get more than one title at the top of your formatted output pages you must do two things: define the titles using .RT, and then allocate space for the titles by using the .HS control word.
- If you do not want any top titles at all, the best way to accomplish this is to define the heading space as 0 (.HS 0). This is more efficient than setting the top titles to null (.RT T ////), because SCRIPT/VS does not have to process any titles to determine that none are wanted.

Example:

If you want three running top titles in your document, you could use the following sequence:

```
.rt t 1 $$&SYSMONTH./&SYSYEAR.$$
.rt t 2 ////
.rt t 3 /CHAPTER 4//&/
```

At this point, only top title 1 will be used on formatted output pages, because the default heading space of 1 is still in effect. Now that the three title lines have been saved, the following causes SCRIPT/VS to print all three:

```
.tm 8
.hs 3
```

.HW [HYPHENATE WORD]

Use the .HW [Hyphenate Word] control word to specify how a single occurrence of a word should be hyphenated if needed.

.HW	text-word

Where:

text-word is the word that you want to hyphenate. It should be entered with hyphens showing where you want it broken.

Notes:

- The .HW control word is a separate function from the hyphenation facility; it works regardless of whether hyphenation is ON or OFF (via the .HY [Hyphenate] control word).
- The .HW control word does not define how a word should be hyphenated every time it is encountered. It specifies how to handle that word for this particular instance only. If you want a word hyphenated every time it occurs (if hyphenation is in use), then you must define hyphenation points for the word in the dictionary using the .DU control word.
- If, while SCRIPT/VS is formatting the line, it is not necessary to break the word, the hyphens are compressed out, and they do not appear in the output. If you want to indicate a hyphen that should remain in a "compound-word," use two hyphens:

```
This is a
.hw com-pound--word
that may be broken in
either of two places.
```

.HY [HYPHENATE]

Use the .HY [Hyphenate] control word to cause automatic hyphenation to be turned on and off.

.HY	{ ON } { NOADD } { OFF } { SUP }
	SET { THRESH } n { MINPT }

Where:

ON begins automatic hyphenation of SCRIPT/VS output lines.

NOADD specifies that the addenda dictionary created using the .DU control word is not to be searched for words to be hyphenated.

OFF causes hyphenation to be turned off.

SUP causes hyphenation to be suppressed until the next space. If hyphenation is OFF, then SUP does nothing, but if it is ON, then SUP turns it off temporarily. It automatically turns on again the next time a line space is generated (as a result of .SP, .SK, head-level control words, or a page eject). This allows you to suppress hyphenation at the end of a paragraph without having to turn it off and then on explicitly.

SET indicates that you are going to override one or both of the default hypenation values, THRESH and MINPT.

THRESH n is a character count indicating the hyphenation threshold. When SCRIPT/VS is formatting a line, at least the number of characters specified must remain before SCRIPT/VS attempts hyphenation. The initial value of THRESH is 7, and the minimum is 2.

MINPT n is a positive number indicating the minimum hyphenation you want to allow. The

initial value of MINPT is 4, which means that the first hyphenation point in a word must be at least four characters beyond the beginning of the word.

Notes:

- When SCRIPT/VS is formatting text, and the next word does not fit on the line, it ordinarily moves the word onto the next output line. When hyphenation is in effect, SCRIPT/VS attempts to break the word into two pieces: the longest piece that can fit on the line, and the remainder. If there are at least "THRESH" spaces left on the line, the word is examined by the hyphenator, which returns a number that is less than the number of remaining spaces. If this number is greater than MINPT, SCRIPT/VS breaks the word after that number of letters.

.H0 - .H6 [HEAD LEVEL 0 - 6]

The control words .H0 through .H6 automatically format topic headings in SCRIPT/VS output. The definition of a particular head level may also result in an entry in the table of contents for that heading. The definition of a head level may be changed with the .DH [Define Head Level] control word or with the .DM [Define Macro] control word.

.Hn	[text]

Where:

n is the number of the head level from 0 to 6.

text is the data to be formatted as a subject head and optionally placed in the table of contents.

Notes:

- The .Hn control words provide several automated functions for you. They can provide a topic heading that is underscored or capitalized with a specified number of skips before it and line spaces after it. They can cause the unformatted topic head to be saved, along with the current page number and revision code character, in the table of contents utility file for automatic table of contents generation. These functions may be redefined using the .DH [Define Head Level] control word.

  Whether you use the default values or redefine them, the topic head that is generated gives you the function of a keep for the size of the space after the heading plus 3 lines. This keep is of the form ".KP v + v". See the discussion of the .KP [Keep] control word for information about which forms of keep may cancel or supersede this form.

- These control words all cause breaks.

- If a head level control word calls for an entry in the table of contents, the text goes into the table of contents as entered. You control how the table of contents entry is capitalized by how you enter the associated head level control word text.

- See Figure 31 on page 314 for information about the default and EasySCRIPT default head-level definitions.

- The head-level functions are actually provided by internally generated macros. The names of the macros are DSMSTDH0 - DSMSTDH6, or, when EasySCRIPT is in effect, DSMEZSH0 - DSMEZSH6. You can display the current definition of any of these macros with the .IT [Input Trace] control word. For example,

  ```
  .it snap dsmstdh3
  ```

  causes the definition of the macro for .H3 to be displayed. The .DH [Define Head Level] control word operates by changing these macros. Note that the macro for a particular head level does not exist until it has been invoked for the first time with the associated .Hn control word or defined with the .DH control word.

- If you wanted to define a head level, such as .H3, to include function not within the scope of the .DH [Define Head Level] control word, there are two different methods you could use:

1. You could define a .H3 macro that would provide all the function you wanted for .H3. Your macro would then operate whenever .H3 was encountered in the input file, assuming macro substitution was ON.

2. You could augment the function of the existing .H3 by adding or deleting lines from the DSMSTDH3 macro, which ordinarily provides the function for .H3. The function of the .H3 control word is merely to call the DSMSTDH3 macro, even if macro substitution is OFF. If you do this, make sure the DSMSTDH3 macro has been initially defined by issuing ".DH 3", before attempting to change it.

In either case, see the discussion of the .DM [Define Macro] control word for information about defining macros.

.IF [IF]

The .IF [If] control word allows a SCRIPT/VS input line to be processed conditionally.

.IF	comp1 test comp2 target
	SYSPAGE test { EVEN } target { ODD }
	SYSOUT test { PRINT } target { TERM }

Where:

comp1 is any word or number of eight or fewer characters to be used as the first comparand. This comparand may be the value of a set symbol.

comp2 is any word or number of eight or fewer characters to be used as the second comparand. It too may be the value of a set symbol.

test is a one or two character code that tells SCRIPT/VS how to determine whether the comparison between the two comparands is true. The following codes are recognized by SCRIPT/VS:

Code	Meaning
eq or =	equals
ne or ¬=	is not equal to
gt or >	is greater than
lt or <	is less than
ge or >=	is greater than or equal to
le or <=	is less than or equal to

target is any valid SCRIPT/VS input line. It may be a control word or text. If the condition is true, then the target line is processed next, with the first nonblank character after the second comparand treated as the first position of the subject line. If the condition is not true, the target line is ignored, and processing continues with the input line that follows the .IF control line.

SYSPAGE is a special .IF keyword that tests whether the page that SCRIPT/VS is currently processing is an even- or odd-numbered page.

SYSPAGE may have only one of the two values, EVEN or ODD.

SYSOUT is a special .IF keyword that tests whether SCRIPT/VS output is being directed to the offline printer (if the PRINT option has been specified), or to the terminal (if the TERM option, the default, is in effect).

SYSOUT may have only one of the two values, PRINT or TERM.

The SYSOUT keyword is provided for compatibility with SCRIPT/370 Version 3. In SCRIPT/VS, there is more variety possible in output formatting than can be determined with this keyword. The SCRIPT/VS system symbols '&$LDEV' and '&$PDEV' may be used to determine the actual

logical and physical devices for which formatting is being done.

Notes:

- This control word provides a powerful conditional processing capability to SCRIPT/VS. The .IF and .GO control words can in some cases replace the .CS [Conditional Section] control word.
- The .IF control word itself does not cause a break; the target control word, if it is processed, might.
- Two special sets of comparands are recognized by the IF processing routine. These are SYSPAGE EVEN/ODD and SYSOUT PRINT/TERM. You may use SYSPAGE to determine whether the current page is even- or odd-numbered. The SYSOUT keyword is provided for compatibility with SCRIPT/370 Version 3, as noted above. When you use these two special comparands, you must capitalize the keywords; "SYSPAGE" is recognized, but "syspage" is not. You may use any of the test codes with SYSPAGE and SYSOUT:

  ```
  .if SYSPAGE eq EVEN (do this)
  ```

  is the same as

  ```
  .if SYSPAGE ne ODD (do this)
  ```

- If any value on the .IF control word line exceeds 8 characters after substitution, an error message is issued.

Examples:

1. The subject of an IF may be another IF. Suppose you wanted to imbed a file called ABC if it is after noon and the file is being formatted in printer format. You could use the following:

   ```
   .se H = &SYSHOUR
   .if &H ge 12 .if SYSOUT eq PRINT
                                .im ABC
   ```

   This is the same as saying, "IF the hour is 12 or more, AND IF the output is in printer format, THEN imbed the file; OTHERWISE, go on to the next line."

2. If you want the subject line to contain more than one control word, you should use a special method. Since .IF is a control word, any control word separators on the line are detected before the .IF is processed. Thus, a control word in the form:

   ```
   .if &sval gt 32 .sk 5;.im fig7
   ```

   will process only the .sk 5 conditionally. The .IM control word is treated as a second control line. The following method can be used to get more than one control word to be conditionally processed:

   ```
   .if &sval gt 32 .cw ?;
           .cm ?.sk 5?.im fig7?.cw ;
   ```

   As in the previous example, only the part before the ; is processed conditionally. The remainder of the line is a .CM [Comment] line. If the condition is not true, the .CW control word is not processed, and the remaining line is treated as a comment. If the condition is true, the .CW is processed, and the new control word separator is recognized to allow the remaining line to be broken up into four active control words.

3. If there is a possibility that one of the comparands may be a null symbol, another technique should be used:

   ```
   .if X&answer eq Xyes (do this)
   ```

   Now, if the symbol "answer" is null, the line will become:

   ```
   .if X eq Xyes (do this)
   ```

   Otherwise, if you had not included the Xs, a null symbol could shift the fields over like this:

   ```
   .if eq yes (do this)
   ```

   and "yes" is not a recognized condition. Note that the symbol is null only if so set by the .SE or .RV control words.

4. Although the .IF control word can only compare tokens of eight characters or fewer, the INDEX function of the .SE [Set Symbol] control word can be used in conjunction with .IF to test character strings longer than eight characters. Consider the following macro definition:

   ```
   .dm t() /.su off
   .dm t() /.se *L1 = &L'&*1
   .dm t() /.se *L2 = &L'&*2
   .dm t() /.se *t = INDEX &*1 &*2
   .dm t() /.su on
   .dm t() /.if &*L1 NE &*L2 .se *t=0
   .dm t() /.if &*t EQ 1 .ty TRUE
   .dm t() /.if &*t NE 1 .ty FALSE
   ```

   The local symbol "&*t" will be set to 1 if the two strings are the same length and if the second string is a substring of the first, starting with the first character. You can write such a macro to do whatever you want if the two strings are (or are not) equal.

Note that in this example, you must invoke the macro with the two strings in single quotes to allow an entire string to be treated as a single token to the macro:

```
.t 'string 1' 'string 2'
```

If you don't use quotes, the macro processor will separate the line into tokens using blanks as delimiters.

.IL [INDENT LINE]

Use the .IL [Indent Line] control word to indent the next output line.

.IL	[0 / h / +h / -h]

Where:

h specifies the amount of horizontal space to shift the next line from the current margin. +h specifies that text is shifted to the right, and -h shifts text to the left.

Initial Setting: 0

Default: 0

Notes:

- The .IL control word provides a way to indent only the next output line. The line is shifted to the right or the left of the current margin (which includes any indent or offset values in effect).
- This control word acts as a break.
- The .IL control word and the .UN [Undent] control word are opposites; thus, the control words .UN 5 and .IL -5 are equivalent.
- The .IL control word may be useful for beginning new paragraphs.
- When successive .IL and .UN control words are encountered without intervening text, or when positive or negative increments are specified for .IL control words entered without intervening text, the indent amount is newly set for the next output line, and any unused .IL or .UN is cancelled. Thus the lines

  ```
  .il 4
  .il 6m
  ```

  result in the next line being indented 6 em-spaces.

Example:

```
.il 3m
```

This line is preceded by the control word .il 3m, and it has enough text to show how the first line is indented differently from subsequent lines.

.IM [IMBED]

Use the .IM [Imbed] control word to process the contents of a specified file at this point in the current file. Processing continues as though the material in the imbedded file were part of the current file.

.IM	file-id [token1 ... token14]

Where:

file-id is a one- to eight-character internal SCRIPT/VS name for the file to be imbedded. This name can be associated with a real file or data set name with the .DD [Define Data File-id] control word. If no .DD has been executed for the name, a real name is built by SCRIPT/VS from the given file-id, using established rules for the environment in which it is operating. A description of the file-id in the various environments in which SCRIPT/VS operates is given in "Chapter 2. Using the SCRIPT Command" on page 13.

tokens are positional values with a maximum length of 8 characters to be passed to the file to be imbedded. The first token (word) becomes the value of the symbol &1, the second token becomes the value of the symbol &2, and so forth. The symbol &0 contains the number of tokens that were passed; up to 14 may be specified.

Notes:

- Any SCRIPT/VS control word or text may be in an imbedded file. Files may be imbedded to a maximum nesting level of eight, and no more than 16 files may be active at one time. If you have many files that are open because of the .EF [End of File] control word, the nesting limit of 8 may be reduced. After .EF is processed, that file is left open, but it is not in the list of currently imbedded files. The file is one of the 16 that can be open, but not one of the 8 that can be in the imbed list.
- The .IM and .AP control words perform similar functions, but .IM allows the contents of a second file to be inserted into the processing of an existing file, rather than appended to the end of it. Imbedding may be used to insert standard sets of control words at desired spots in a file, as well as for many other purposes.
- The symbols &0 through &14 are reset whenever an .IM or .AP control word is processed. Whatever tokens are not given on a .IM line are reset. If you want to leave token &1 unset but set token &2, you may use a percent sign (%) in place of the token.

Example:

```
.im common chap4
```

The contents of the SCRIPT file whose file-id is COMMON are inserted into the processing sequence of the current SCRIPT file; when the end of the COMMON file is reached, processing of the current file resumes. The token "CHAP4" is set as the value of the symbol "&1." The file COMMON might have in it another imbed in the form:

```
.im &1
```

and this would be substituted as:

```
.im chap4
```

A different file could contain the control word

```
.im common CHAP5
```

so that &1 in COMMON is substituted with CHAP5 instead.

.IN [INDENT]

Use the .IN [Indent] control word to change the left margin displacement of SCRIPT/VS output.

.IN	[0 / h / +h / -h]

Where:

h specifies the amount of space to be indented. If omitted, 0 is assumed, and indention reverts to the left margin. If you use +h or -h, the current left margin is incremented or decremented accordingly.

Initial Setting: 0

Default: 0

Notes:

- The .IN control word resets the current left margin. This indention remains in effect for all following lines (including new paragraphs and pages), until another .IN control word is encountered. ".IN 0" cancels the indention, and output continues at the original left margin setting.
- The value of h represents the amount of blank space left before text. Thus, ".in .5i" sets a left margin of one half-inch, and the text begins after this blank margin area. The .TB [Tab Setting] and .OF [Offset] control words work in a similar manner.
- This control word acts as a break.
- The .IN control word cancels any .OF setting. Any .OF request cancels the current offset, but leaves the left margin specified by the most recent .IN control word unchanged.
- The value of the system symbol &$IN reflects the composite net indention for the next output line, as controlled by .IN [Indent], .UN [Undent], .IL [Indent Line], and .OF [Offset]. After a .OF control word, &$IN changes only after the offset has been triggered by the next output line being finished or by a .BR control word.
- Although .IN cancels the .OF setting, it does not cancel the effect of .IL or .UN. An attempt to set the indention to the left of the real left margin or to the right of the right margin results in an error message, and all indention is reset to zero.

Examples:

1. .in 10

   All lines processed after this request are indented 10 character spaces from the left. This indention continues until another .IN control word is encountered.

2. .in 0

   The effect of any current .IN and .OF control words is cancelled, and output is formatted flush left. Any .UN [Undent] control word, however, is not cancelled by .IN 0.

3. .un 0;.in 0

   All left indention is reset.

.IR [INDENT RIGHT]

Use the .IR [Indent Right] control word to change the right margin displacement of SCRIPT/VS output.

.IR	[0 / h / +h / -h]

Where:

h specifies the amount of space to be indented. If omitted, 0 is assumed, and indention reverts to the right column boundary. If you use +h or -h, the current right margin is incremented or decremented accordingly.

Initial Setting: 0

Default: 0

Notes:

- The .IR control word resets the current right margin. This indention remains in effect for all following lines (including new paragraphs and pages), until another .IR control word is encountered. ".IR 0" cancels the indention, and output continues at the original right margin setting.
- The value of h represents the amount of blank space just before the right margin.
- This control word acts as a break.

Examples:

1. .ir .5i

   All lines processed after this request are indented one half-inch from the right hand side of the column. This indention continues until another .IR control word is encountered.

2. .ir 0

   The effect of any .IR control word is cancelled, and subsequent lines are formatted to the right hand margin.

.IT [INPUT TRACE]

The .IT [Input Trace] control word allows trace information about input lines to be displayed at your terminal or written to the same file as error messages.

.IT	[ON / OFF / MAC / SUB / ALL / CTL [cw cw...] / STEP / RUN / SNAP [name name...]]

Where:

ON traces macro and symbol substitution, and any control word that has been specified previously with CTL.

OFF terminates tracing.

MAC causes each line coming out of a macro to be traced.

SUB causes each stage of symbol substitution to be traced for lines that contain symbols or GML tags.

ALL causes macro and symbol substitution and all control word lines to be traced.

CTL causes the control words specified to be traced before they are executed. If no control words are given with .IT CTL, then the list of control words to be traced is cleared. If some control words are given, they are added to the list. Nonexistent control words may be added to the list without causing an error, but they will never be traced because they will be detected as invalid control words before tracing would be done. The list remains intact when .IT OFF is executed, and is resumed if .IT ON is subsequently executed.

STEP causes SCRIPT/VS to "single step" through all control words that are being traced. If .IT ALL is in effect, all control words are traced. Otherwise, just those control words specified with .IT CTL are traced.

When .IT STEP is in effect, SCRIPT/VS displays the control word line, and then pauses to read a line from the terminal before executing it. The line you enter at this point can simply allow the control word execution to proceed, or you can enter another input line to be processed before, after, or instead of, the traced control word.

RUN cancels .IT STEP mode, while allowing all tracing to continue. (.IT OFF stops STEP mode, and also stops tracing.)

SNAP displays the current definitions for any symbol and macro that exist by the name or names given. If no names are given, the entire symbol and macro table is displayed. The SNAP is done without changing any other tracing that may be in effect.

Notes:

- All trace information is written out as messages. If the MESSAGE (DELAY) option of the SCRIPT command is in effect, the trace information is written to the same SCRIPT/VS utility file as error messages.
- .IT STEP mode can only take effect if messages (and trace information) are actually being displayed at your interactive terminal. Thus, STEP mode is not available in the batch environment or when the MESSAGE (DELAY) option of the SCRIPT command is in effect. When SCRIPT/VS reads a line from the terminal after tracing a control word line in STEP mode, it may have any of the following formats:

  null line - continue processing

  ? - verifies who is reading from the terminal. If you are stepping through control words and you are also using .TE [Terminal Input], it's easy to lose sight of which kind of read is being done from the terminal. While in .IT STEP mode, the single character ? is recognized by the control trace module and by the terminal input module, and the message "TERMINAL INPUT:" or "CONTROL TRACE:" is displayed, and another read is done. If the read comes from some other source, such as .RV [Read Variable] or .RD [Read Terminal], the ? is taken as ordinary data, just as it would be from terminal input when not in .IT STEP mode.

  STK 'data line' - the data line entered is stacked and processed after the traced control word has been processed

  PRE 'data line' - the data line entered is processed before the traced control word (the tracing is done before the control word is actually processed).

  REP 'data line' - the data line entered replaces the traced control word line, and is processed instead of it.

  'data line' - the data line is treated like a data line entered with the 'PRE' keyword.

  If the new line to be entered is also a control word line that is being traced, it will be traced before being processed, giving another opportunity to enter a line. If the line entered causes the original line to be reprocessed later, it may be traced again.

- The trace function is initially OFF.
- The SNAP parameter provides a selective printout of all currently defined set symbols and their values. It does not affect the current ON/OFF status of the trace control.
- On all trace lines, the first three characters indicate which type of trace it is, as follows:

  S symbol substitution trace

M macro substitution trace

C control word trace

*** symbol or macro SNAP line

If .IT ALL is in effect, control word lines may be traced several times. Each may be traced to show the various stages of symbol substitution, then traced again as a control word line after it has been completely substituted. You can tell which type of tracing a line represents by the first three characters of the line.

.JU [JUSTIFY MODE]

Justification of output lines is one component function of line formatting. Justification is turned ON or OFF by the .FO [Format Mode] control word or the .JU [Justify Mode] control word. In general, the .FO control word is the preferred way to control justification.

.JU	[ON / OFF]

Where:

ON restores right justification of output lines. If neither ON nor OFF is specified, ON is assumed.

OFF cancels justification of output lines. If concatenation is still in effect, .JU OFF results in ragged right output.

Initial Setting: ON

Default: ON

Notes:

- Concatenation and justification are controlled by the .FO [Format Mode] control word. Ragged right output results from concatenation ON and justification OFF. The control word .FO LEFT provides this combination. Full formatting, with concatenation and justification both ON, is provided by .FO ON. "As is" output, with concatenation and justification both OFF is provided by .FO OFF. The only combination not covered by the .FO control word is concatenation OFF and justification ON, and if you need this combination, you can use the .JU control word to control it separately.

.KP [KEEP]

The .KP [Keep] control word allows you to designate blocks of text that must be kept together in the same column. There are several different ways of designating keeps, and each form has different functions and powers. When .KP is encountered inside another keep, it may end the first keep before starting the new one. If the new keep is of a form that can't end the current keep, it is ignored, and the text is kept together by virtue of being part of the larger keep.

.KP	{ ON }
	{ FLOAT }
	{ DELAY }
	{ INLINE }
	{ OFF }
	{ V + V }
	{ V }

Where:

ON starts a regular keep. The text within a regular keep is separate from the text outside of it, and no output line can be built from text part of which came from inside and part from outside the keep. A regular keep is put in this column if

it will fit, and otherwise an immediate column eject is done. The regular keep appears in the output in the same relative location where it was in the input. A regular keep ends any other keep before starting. The ON option causes a break.

FLOAT starts a floating keep. A floating keep is put in this column if it will fit; otherwise it goes at the top of the next column. Text following the floating keep is formatted into the rest of this column. .KP FLOAT ends any other keep before the floating keep is started.

DELAY starts a delayed keep. A delayed keep is always printed at the top of the next column, even if there is room for it in this column, and the current column is filled with text from after the delayed keep. A delayed keep, in effect, acts like a floating keep that did not fit in the current column. .KP DELAY ends any other keep before starting.

INLINE starts an inline keep. An inline keep flows with the preceding and following text. No separation of material inside and outside the keep is done, but formatting continues as though no keep were designated. All lines that contain text from within the keep are then kept together, and if column balancing is done, the entire keep is moved as a block from one column to another. .KP INLINE ends an inline keep or a keep in the form ".KP v + v" or ".KP v" before the inline keep is started, but if a regular, floating, or delayed keep is in process, .KP INLINE is ignored.

OFF marks the end of a regular, floating, delayed, or inline keep. .KP OFF also ends a keep of a designated depth, but is not required. When a keep is started, its maximum depth is set to the maximum that can be kept together, typically, the depth of one full column of text. If the keep is filled before .KP OFF is processed, the keep is ended, and a warning message is issued.

v + v starts a keep of a designated vertical depth. The depth of the keep is determined by adding up all the separate v's given. For example, .KP 3 + 2 would start a keep for 5 lines, and .KP 2i + 3 would start one for 2 inches plus 3 lines. (This is the only control word that allows you to add up different space units to get a single result.) This form of keep is used by the head level control words .H0 - .H6 [Head Level 0 - 6]. A keep for a designated depth need not be explicitly ended with .KP OFF. It will be ended automatically when its depth has been filled. A keep of the form .KP v + v may end another keep of the same form or a keep of the form .KP v. If an inline or higher keep is in process when .KP v + v is encountered, the .KP v + v is ignored. Any head level control word also ends a keep of the "v + v" form.

v starts a keep of a designated depth specified by v. When the designated depth has been filled, a keep of the "v" form is automatically ended. A keep of the "v" form may end another keep of the same form before starting, but if any other form of keep is in process, .KP v is ignored.

Default: None.

Notes:

- Keeps started with .KP ON, .KP FLOAT, and .KP DELAY all operate with a separate environment from ordinary text. No output line may be formed by concatenating text from inside the keep to text from outside of it. When the keep is started, the current indention and certain other values are saved. Offsets and undents are cleared so that the indention at the beginning of the keep is set to the basic indention currently in effect, and the maximum column width is set to the width of the current column. When the keep is ended, the original text values are restored automatically. This means that if you change the indention, formatting mode, hyphenation, double spacing, or certain other things, you need not restore them when the keep is ended. See Figure 32 on page 315 for a list of the active environment values that are saved and restored for these keeps.
- Keeps started with .KP INLINE, .KP v + v, or .KP v are not separated from the surrounding text. Output lines may be formed by concatenating text from inside the keep to text from outside of it. No environment values are saved or changed, and the text within the keep flows with neighboring text.

Formatting continues for these keeps as though no keep had been started, but all output lines encompassed by the keep are kept together in the same column of output.

- Certain control words are not allowed within a keep. If one of the disallowed control words is encountered, the keep is immediately ended, as though .KP OFF had been processed, and then the disallowed control word is executed. A warning message is issued, telling you what control word ended the keep. See Figure 28 on page 312 for a list of the disallowed control words.
- If too much text is processed before .KP OFF is encountered, the keep is ended at its maximum depth. This is the normal way for a keep of a designated depth to end, and no message is issued in these cases. For other keeps, a message is issued, and then processing continues. If a regular keep is ended because it is full, the remaining material is processed as normal text. If a floating or delayed keep becomes full, the remaining material until the .KP OFF is discarded.
- A footnote is a specialized form of keep. A footnote ends any keep before starting. Regular, floating, or delayed keeps can end a footnote, but inline, 'v', and 'v + v' keeps are ignored within footnotes.

.LB [LEADING BLANK]

The .LB [Leading Blank] control word is generated by SCRIPT/VS and executed whenever an input line that starts with a blank is processed.

.LB	

Notes:

- This book states in several places that a leading blank on an input line causes a break. This is actually done by generating and executing a .LB control word whenever a line with a leading blank is processed, and the function of the .LB control word is identical to that of the .BR control word.

  If you wish to have leading blanks perform some other function, you can define a .LB macro with .DM [Define Macro], and, assuming macro substitution is ON, your .LB macro will be executed whenever a leading blank is processed. Note, however, that after the .LB control word or macro is processed, the leading blank is still on the line, and it is processed as part of that text input line. In other words, you cannot use the .LB macro to remove leading blanks from a line.
- No .LB function is performed for lines processed in literal mode (.LI [Literal]).

.LI [LITERAL]

The .LI [Literal] control word allows all input lines, including those that begin with periods, to be processed as text.

.LI	[1 / n / ON / OFF / line]

Where:

n specifies the number of lines to be treated literally. If omitted, 1 is assumed.

ON starts an open-ended literal mode, in which every line read is treated as literal text. After this control word is processed, SCRIPT/VS reads input lines looking only for ".LI OFF" beginning in column 1 on a line by itself.

OFF terminates literal mode if it was ON, or if n was given and has not been exhausted.

line is the line to be treated as literal text.

Notes:

- Ordinarily, any SCRIPT/VS input line that begins with a period is interpreted as a SCRIPT/VS control word. The LITERAL control word causes the following n lines to be processed as normal input lines even if the first character of one of the lines is a period. If .LI ON is encountered, all subsequent lines except .LI OFF (which must be recognized to cancel literal mode) are treated as literals.
- When literal mode is in effect, null lines, lines with leading blanks, and lines with leading tabs do not cause a break. Null lines, however, do cancel continuation if the previous line ended with a continuation character.

Example:

If a text line must begin with a period:

```
Study the following control words:
.fo off
.in 5
.li on
.LB [Leading Blank]
.LT [Leading Tab]
.NL [Null Line]
.LI [Literal]
.li off
.fo on
.in
```

These lines are formatted as:

```
Study the following control words:
     .LB [Leading Blank]
     .LT [Leading Tab]
     .NL [Null Line]
     .LI [Literal]
```

If formatting mode had not been turned OFF with .fo off, the same lines would be processed as:

```
Study the following control words:
     .LB [Leading Blank] .LT [Leading
     Tab] .NL [Null Line] .LI [Literal]
```

.LL [LINE LENGTH]

The .LL [Line Length] control word specifies the width of running titles, running headings, and running footings. It also changes the column width, which governs the width of text lines, if the latter has never been set explicitly with .CL [Column Width].

.LL	[h +h -h]

Where:

h specifies an output line length not greater than the output device capability. If no value is specified for h, the default value established for the device being used will be taken.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

- The .LL control word sets the total length for output lines from the left margin to the right margin. The .LL value governs the length of title lines. Text lines are always governed by the .CL [Column Width] control word, but if the column width has never been explicitly set, it has the same value as the line length. See the discussion of the .CL control word.
- This control word takes effect on the page after it is encountered. If it also performs a .CL function, however, the new column width takes effect immediately.
- This control word causes a break.

.LS [LINE SPACING]

Use the .LS [Line Spacing] control word to specify multiple-spacing of output text lines. This control word is a generalization of the .SS [Single Space Mode] and .DS [Double Space Mode] control words.

.LS	n

Where:

n specifies the number of blank lines to be inserted after each standard text line.

Initial Setting: 0

Default: None.

Notes:

- Contrast the function of the two control words .LS [Line Spacing] and .SL [Set Line Space]. The .SL [Set Line Space] control word defines the actual depth of each single output line. For example, .SL .5i sets the vertical depth of each output line to one-half inch. The .LS [Line Spacing] control word, on the other hand, defines how many output lines should be generated for each line of text or space request. If .SL .5i has been executed, each single-spaced line of formatted output occupies one-half inch vertically, and if double-spacing is in effect, another half-inch blank line is inserted after each text line.
- This control word does not cause a break.
- The .SS control word is identical to .LS 0, and the .DS control word is identical to .LS 1. When line spacing is in effect, the amount of vertical space requested by the .SK [Skip] and .SP [Space] control words may be multiplied by a line-spacing multiplier if the request is in lines. For example, if .LS 1 (double spacing) is in effect, the line space multiplier is set to 2. A .SK or .SP request for a particular number of lines, such as .sk 3, is multiplied by 2, and the actual amount spaced is 6 lines. (Each of the 6 lines is of

the depth defined with .SL [Set Line Space].) If the skip or space request is for an absolute amount, however, it is not multiplied. A request to skip 3/4 of an inch (.sk .75i) is an absolute request, and the amount skipped will be as close to 3/4 of an inch as the logical device allows, regardless of the current line spacing.

- This control word overrides previous .LS, .SS, and .DS control words.

Example:

```
.ls 2
```

Subsequent output is "triple-spaced" such that each text line is separated by two blank lines.

.LT [LEADING TAB]

The .LT [Leading Tab] control word is generated by SCRIPT/VS and executed whenever an input line that starts with a tab is processed.

.LT	

Notes:

- This book states in several places that a leading tab on an input line causes a break. This is actually done by generating and executing a .LT control word whenever a line with a leading tab is processed, and the function of the .LT control word is identical to that of the .BR control word.

  If you wish to have leading tabs perform some other function, you can define a .LT macro with .DM [Define Macro], and, assuming macro substitution is ON, your .LT macro will be executed whenever a leading tab is processed. Note, however, that after the .LT control word or macro is processed, the leading tab is still on the line, and it is processed as part of that text input line. In other words, you cannot use the .LT macro to remove leading tabs from a line.

- No .LT function is performed for lines processed in literal mode (.LI [Literal]).

.LY [LIBRARY]

Use the .LY [Library] control word to cause symbol and macro definitions to be retrieved from a library defined with the LIB option on the SCRIPT command.

.LY	[ON / SYM / MAC / OFF]

Where:

ON specifies that both symbol and macro definitions may be retrieved from a library. This is the default.

SYM causes unresolved symbol values to be retrieved from the library. If SYM is specified, the library will not be used to resolve undefined macros from the library (unless MAC or ON is also specified).

MAC causes unresolved macro definitions to be resolved from the library. If MAC is specified, no undefined symbol values will be resolved from the library (unless SYM or ON is also specified).

OFF indicates that use of the library for symbol values and macro definitions is to stop. This is the initial setting.

Initial Setting: OFF

Default: ON

Notes:

- Use of the library to resolve symbol values and macro definition is expensive in processing time. This is especially true for forward referencing of symbol values where there are normally many potentially unresolved symbols. For this reason, the .LY control word is provided to control library look-up. The .LY control word allows you to tell SCRIPT/VS that, if unresolved symbols or macros are used in a document, to attempt to resolve these from a library.
- Symbol values or macro definitions may be explicitly set from the library, regardless of the setting of the .LY control word, using the LIB option of the .SE and .DM control words.

.MC [MULTICOLUMN MODE]

The .MC [Multicolumn Mode] control word restores multiple column processing after it has been temporarily suspended by .SC [Single Column Mode].

.MC	

Notes:

- The .MC control word cancels a temporary single column mode that was put into effect by the .SC [Single Column Mode] control word. If there was no .SC control word preceding this control word, it has no effect, other than to cause a break.
- This control word is not allowed in a keep.
- The .SC control word saves the current column definition, and starts a temporary single-column processing mode. The column definition that was in effect when .SC saved it might actually have been a multiple-column definition, or it might have been a single column definition. The .MC control word is, perhaps, misnamed. What .MC actually does is to restore the column definition that was saved by .SC, however many columns that definition called for. The column definition saved by .SC and restored by .MC includes the number of columns and their positions and the column width. If two .SCs are processed without an intervening .MC, then it takes two .MCs to restore the original column definition that existed before the first .SC. The first .MC restores the single column definition that existed, by virtue of the first .SC, when the second .SC was processed.
- The .CD [Column Definition] control word starts an entirely new column definition, and cancels any .SCs and .MCs that may be in effect.

.MG [MESSAGE]

The .MG [Message] control word is used to write out a message. It may be used to provide diagnostic messages from macros.

.MG	/[mid]/[message text]/

Where:

/ is any delimiter character. The first nonblank character will be taken as the delimiter character.

mid is the message id. This string must not be longer than 16 characters and the last character must be R, I, W, E, S, or T. This final letter is used to establish the severity of the message, and the same meanings apply as for regular SCRIPT/VS messages. If a null message id is specified, the message is considered to be of type I, an information message. If the message is type R (Response), you must provide the terminal read using .RV or .TE; the Message control word will not do this for you. The message id is not printed unless the MESSAGE (ID) command option is in effect.

message text is the text of the message. It may be any string of characters.

Notes:

- Messages generated by .MG may cause SCRIPT/VS processing to terminate. Type S (severe) or type T (terminating) messages always terminate processing, and type E (error) messages terminate processing if the CONTINUE option of the SCRIPT command is not in effect.
- The delimiter character between the strings may be any unique character which does not occur within the strings themselves.
- When a message is displayed, a prefix of "+++" appears before the id or text to indicate the message was generated by .MG. If the net message is null, the prefix only is displayed. This can happen if you do not specify any message id or text, or if you specify a message id and no text, but the MESSAGE (ID) option is not in effect.
- If the .MG line has no data at all, it is ignored.
- If the message header is longer than 16 characters or if it does not end with one of the valid type-codes, it is considered an invalid control word parameter, and an error message is issued.

Example:

The control word:

```
.mg /msg001e/this is a message/
```

is displayed as:

```
+++MSG001E this is a message
```

if MESSAGE(ID) is in effect, or:

```
+++ this is a message
```

if MESSAGE(ID) is not in effect.

.MS [MACRO SUBSTITUTION]

Use the .MS [Macro Substitution] control word to initiate or cancel automatic macro calls during SCRIPT/VS processing.

.MS	{ ON } { OFF }

Where:

ON causes SCRIPT/VS to begin searching for macro names when it encounters unrecognized control words.

OFF causes SCRIPT/VS to stop searching for macro names during processing.

Initial Setting: OFF

Default: None

Notes:

- SCRIPT/VS macros can be defined with the .DM control word. However, SCRIPT/VS does not ordinarily recognize and process macros unless the .MS control word has been used. When macro substitution is OFF (the initial setting for SCRIPT/VS processing), macros that have been defined via the .DM [Define Macro] control word are treated as invalid control words.
- Even when macro substitution is OFF, a macro can be explicitly invoked via the .EM [Execute Macro] control word.

.NL [NULL LINE]

The .NL [Null Line] control word is generated by SCRIPT/VS and executed whenever a null line is processed.

.NL	

Notes:

- Whenever SCRIPT/VS encounters a null input line, that is, a line whose length is zero, it generates and executes a .NL control word. The .NL control word does nothing, except to reset line continuation, in case the previous line ended with a continuation character.

  If you wish to have null lines perform some other function, you can define a .NL macro with .DM [Define Macro], and, assuming macro substitution is ON, your .NL macro will be executed whenever a null line is processed.
- No .NL function is performed for lines processed in literal mode (.LI [Literal]). A null text line, however, does reset continuation if the previous text line ended with a continuation character.
- A null line may originate from a number of sources. Because of this, you should define a .NL macro only when a specific use in a certain part of a document requires it. Null lines may originate from:
  - A source input file (not all systems in which SCRIPT/VS operates allow this).
  - From terminal input (.TE).
  - A non-null line that becomes null as a result of substitution.
  - A macro line that is null.

.OB [ODD PAGE BOTTOM TITLE]

The .OB [Odd Page Bottom Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the bottom of the current page, if it is odd-numbered, and each subsequent odd-numbered output page.

.OB	[n] /part1/part2/part3/

Where:

n is the number of the bottom title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even bottom title 1" sets the same storage buffer as "even top title 6." See the discussion of the .FS [Footing Space] control word for information on how to allocate space on your output page for bottom titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any character that does not appear in part1, part2, or part3.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RT [Running Title]. See the discussion of the .RT control word for further information about running titles, including those for the bottom of odd pages.

.OC [OUTPUT COMMENT]

Use the .OC [Output Comment] control word to place comments in the output data stream. Such comments are not examined by the formatter, and will be placed in the output where found in the input. This means that if parts of the page are still in storage the output comment may not appear in the output data stream in the same place that it was, relative to the input data stream. This control word is designed for the systems programming user of SCRIPT/VS and must be used with caution.

.OC	line

Where:

line may be anything, since it is not used in formatting the output. However, since this is a control word, the input line is scanned for control word separators.

Notes:

- The .OC control word allows comments to be placed in the output data stream. They are not examined by the formatter and thus, unless they are correctly interpreted by the output device, the output will be disrupted.

  The .OC may be used, for example, to control a printer in a certain way, such as to transmit codes recognized by certain printers.

- The position of the line written out as an output comment is not synchronized with the formatted output. The output comment may appear in the output data stream before the text that precedes it in the input file, because the text may still be filling a column. If you force a "section break" with .CD [Column Definition] or .SK P, all columns up to that point will be balanced and written out, and the .OC will be in the same relative position in the output as it was in the input. If the output is in a single column, the section break will not be noticeable.

.OF [OFFSET]

Use the .OF [Offset] control word to indent all but the first line of a block of text.

.OF	[0 / h / +h / -h]

Where:

h specifies the horizontal size of the offset. If you specify +h or -h, the old offset value is incremented or decremented the specified amount to establish the new offset size. If "h" is omitted, the new offset size is 0.

The next output line to be formatted after the .OF control word has been processed is formatted at the left margin established by the .IN [Indent] control word, with no added offset. For all subsequent lines, the left margin is established by adding the offset (.OF) to the size of the indent (.IN).

Initial Setting: 0

Default: 0

Notes:

- A .OF control word does not take effect until after the next line is formatted. The offset remains in effect until a .IN [Indent] control word or another .OF control word is encountered.

  The .OF control may be used within a section which is also indented with the .IN control. Note that .IN settings take precedence over .OF, however, and any .IN request clears all offsets.

  If you want to start a new section with the same offset as the previous section, you need only repeat the .OF h request.

- This control word acts as a break.
- The .IL [Indent Line] and the .UN [Undent] control words can be used to shift only the next line to the left or right of the current margin.
- Tabs should be used whenever possible to format numbered or bulleted lists, to ensure that the first text word on the line is even with subsequent offset lines. The items in this "Notes" section are created using offsets and tabs.

Examples:

1. Starting an offset:

```
.of 7
The line immediately following the
      .OF control word is printed
      at the current left margin.
      All lines thereafter (until
      the next indent or offset
      request) are indented seven
      character spaces from the
      current margin setting.
      These two examples were
      processed with .OF control
      words in the positions
      shown.
```

2. Ending an offset:

```
.of
The effect of any previous .OF
request is cancelled, and all out-
put after the next line continues
at the current left margin setting.
```

.OP [ODD PAGE EJECT]

Use the .OP [Odd Page Eject] control word to cause either one or two page ejects, such that the new page is odd-numbered, regardless of whether the current page is even- or odd-numbered.

.OP	[ON / OFF]

Where:

ON specifies that subsequent text is to be printed only on odd-numbered pages. Even-numbered pages are left blank, except for top and bottom titles, if any.

OFF resumes processing so that text appears on odd- and even-numbered pages.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .PA ODD [ON|OFF].

.OT [ODD PAGE TOP TITLE]

The .OT [Odd Page Top Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the top of the current page, if it is odd-numbered, and each subsequent odd-numbered output page.

.OT	[n] /part1/part2/part3/

Where:

n is the number of the top title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "odd bottom title 1" sets the same storage buffer as "odd top title 6." See the discussion of the .HS [Heading Space] control word for information on how to allocate space on your output page for top titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

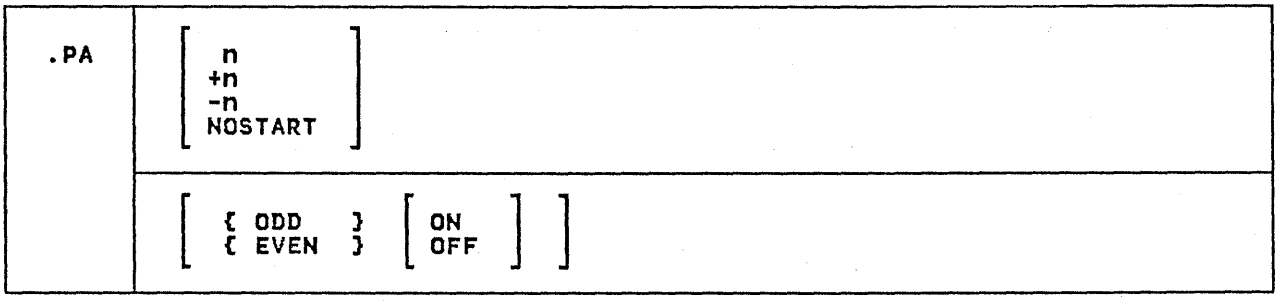
/ is any character that does not appear in part1, part2, or part3.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RT [Running Title]. See the discussion of the .RT control word for further information about running titles, including those for the top of odd pages.

.PA [PAGE EJECT]

Use the .PA [Page Eject] control word to force subsequent text onto a new page of output, even if the current page has not been filled.



Where:

n specifies the page number of the next page. If n is not specified, sequential page numbering is assumed, and the next page number is one greater than the current page number. n must be an Arabic number with no decimal point.

+n specifies that the next page should have a number that is equal to the normal next sequential page number plus n. n must be a non-decimal Arabic integer.

-n specifies that the next page should have a page number that is equal to the next sequential page number minus n. If subtracting n from the next page number yields a negative number, an error message is issued, and the control word is ignored.

The maximum allowed page number is 9999.

NOSTART causes the current page to be ended, but the next page will not be started until some data causes it to be started or a control word that requires the page to be started is processed. After .PA NOSTART, the page definition (including running headings and footings), may be changed until the page is started.

ODD causes one or two page ejects, such that the new page is odd numbered.

EVEN causes one or two page ejects, such that the new page is even numbered.

ON defines the start of odd or even page eject mode. This mode is ended by specifying OFF or the start of another .PA even or odd mode, or n. In odd or even page eject mode, output is formatted on odd pages only, or even pages only, whichever the case may be, and the other pages are left blank, except for running titles and headings and footings.

OFF defines the end of odd or even page eject mode.

Notes:

- The minimum page number is 1, and the maximum is 9999. If a .PA control word attempts to set the page number outside this range, a message is issued, and the control word is ignored.
- Whenever a .PA control word is encountered, the rest of the current page is skipped after printing any text lines accumulated thus far. The next page is started, unless .PA NOSTART was specified. Starting a page includes formatting running headings, running footings, and running titles for the page, and establishing the page dimensions for the page. These things are then fixed for the duration of the page, and may not change until the next page is started.
- If you use the STOP option of the SCRIPT command, SCRIPT/VS waits for you to enter a null line (with the Return or Enter key) before starting the new page.
- This control word acts as a break. It is not allowed in a keep.
- If you want to change any page dimensions or define new running titles or running headings and footings for a new page, the appropriate control words must be processed before the .PA control word (except when NOSTART is

specified). These control words are .BM, .FM, .FS, .HM, .HS, .LL, .PL, .PN, .RH, .RH, .RF, .RT and .TM . Note that at the beginning of SCRIPT/VS processing, the first page has not yet been started.

- If .PA n (or +n or -n) is specified after .PN FRAC is specified, the page eject will occur, but the page number will not be reset. This is because the page number change to fractional pagination is pending.
- The following control words require a page to be started, and will cause one to start if one is not already started: .BX, .CB, .CC, .CD, .CP, .PT, .RD, .SK, .SP, and .SX.

Examples:

1. To start the next sequential page:

   ```
   .pa
   ```

   The rest of the current page is skipped. The top titles and page number are put in the top margin of the next page, and output resumes.

2. To repeat a page number:

   ```
   .pa -1
   ```

   The new page will have the same page number as the preceding page. The calculation is done after establishing the next sequential page number.

.PF [PREVIOUS FONT]

Use the .PF [Previous Font] control word to resume the use of the font whose id was last saved using the .SF [Save Font] control word.

.PF	

Notes:

- If the .PF control word is used when there is no previously saved font, the default font for the output device will become effective.
- This control word is ignored when formatting for logical devices that do not support multiple fonts.

.PL [PAGE LENGTH]

The .PL [Page Length] control word specifies the vertical length (depth) of output pages. The value specified overrides the standard page length which is established for each logical device.

.PL	[v | +v | -v]

Where:

v specifies the vertical length, or depth, of output pages. If no value is specified for v, the default value for the device will be used. This number should be the same as the physical size of the paper being used. However, when formatting for a printer logical device, it may be different, as explained below. The minimum value for the page length is the sum of the top margin (.TM) and the bottom margin (.BM) plus one line. The maximum value that may be specified for the page length is as established for each logical device. If +v or -v is specified, the current page length is incremented or decremented accordingly.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

- The .PL control word allows varying paper sizes to be used for output. (The logical device specified in the DEV option of the SCRIPT command implies a default page length,

but this can be overridden with .PL.) Page length may be changed anywhere in a file, with the change effective on the page after the control word is encountered.

- This control word does not cause a break. It is not allowed in a keep.
- If the output is in printer format, the page length value need not be the same as the actual number of print lines on the real paper, because SCRIPT/VS will cause the printer paper to be ejected to the top of the next real page whenever a new SCRIPT/VS page is started. Thus, a SCRIPT/VS page may occupy less than a real page or more than one real page, and the output will be newly aligned to the paper each time a SCRIPT/VS page is started.
- The previous rule notwithstanding, if you define a top margin (.TM) and a heading space (.HS) and heading margin (.HM) such that SCRIPT/VS needs to print data within the first three lines on a page, no printer page ejects can be done. Instead, SCRIPT/VS uses the page length value to find the top of the next page. It is, therefore, good practice to keep the .PL value accurate, so that it reflects the true depth of the page under SCRIPT/VS control.
- The maximum value of the page length that may be set is governed by the value established as the maximum for the logical device for which formatting is being performed.
- If the running headings and footings that are defined for a page fill up the page so that no room is left for text, SCRIPT/VS terminates with an error message. The depth of running headings and footings cannot be predicted at the time they are defined, because they are formatted to the current line length (.LL) when a page is started. The same running heading can occupy differing amounts of vertical space on different pages if the line length changes.

Example:

```
.pl 84
```

Page length is set to 84 lines. This is the correct size for 14 inch printer paper when printing at six lines per inch.

.PN [PAGE NUMBERING MODE]

The .PN [Page Numbering Mode] control word allows you to control various aspects of page numbering, including the format of the page number, and whether it is to be shown in running titles or running headings and footings that call for it.

.PN	{ OFF }
	{ OFFNO }
	{ ON }
	{ ARabic }
	{ ROman }
	{ ALph }
	{ FRAC }
	{ NORM }
	{ PREF string }
	{ n }

Where:

OFF suppresses the display of page numbers in running titles and running headings and footings, although pages are still sequentially numbered internally. Symbols set with .SE [Set Symbol] to the current page number will contain the correct number of the page on which they were processed.

OFFNO suppresses both page number display and internal page numbering. The current page number set with .SE remains the same for all pages until .PN OFFNO is ended with .PN ON.

ON cancels .PN OFF or .PN OFFNO, so that internal numbering of pages is resumed, and the current page number can be displayed in running titles and running headings and footings.

ARabic causes the following page numbers to be represented as standard Arabic numerals. The ARABIC keyword may be abbreviated as AR.

ROman causes page numbers to be represented as lowercase Roman numerals. Page numbers greater than 3999 are not supported with the ROMAN option. The ROMAN keyword may be abbreviated as RO.

ALph causes alphabetic page numbering to be started. In this mode, the number 1 is converted to a, 2 to b, 26 to z, and 27 to aa. The number 1978 is represented as bxb. The ALPH keyword may be abbreviated as AL.

FRAC causes fractional pagination to begin. The next time a page eject occurs that would normally increment from an even to an odd number, the even number (for example, 20) is saved, and numbering starts with a fractional sequence, in this case, 20.1, 20.2, 20.3, and so forth.

NORM causes an immediate page eject to occur, and normal pagination to be resumed. In the previous example, the new page would be numbered 21. If .PN FRAC is not in effect, .PN NORM is ignored, and does nothing.

PREF string specifies a 1- to 8-character string to be used as a prefix in front of all page numbers printed in titles, in tables of contents, or in front of set symbols set with the value of the current page number (&). The string may not contain embedded blanks. To cause the prefix to be omitted from the page number, specify ".PN pref", with no string. This clears the previously defined prefix string.

n specifies the number of the next page. When the next page eject occurs, either naturally because of the page becoming full, or as a result of .PA, the new page will have the page number specified in n, as though this page eject had been caused by .PA n. If the next page really is started with .PA n, the number given on the .PA control word supersedes the number previously specified with .PN n.

Initial Setting: ON, Arabic

Notes:

- The .PN control word can be used to control SCRIPT/VS's page numbering. If the OFF operand is specified, page numbering is discontinued on output, although the page numbers continue to be incremented internally. The OFFNO operand discontinues page numbering on output and stops the internal incrementing of page numbers. When the ON operand is specified, page numbering resumes from the last internal page number.

  The actual page numbers may appear in either Arabic numerals, which is the default, or Roman numerals, depending upon whether .PN ARABIC or .PN ROMAN was most recent. Changes in the page numbering will take effect on the page after the .PN control was encountered.

- The .PN OFF and .PN OFFNO control words suppress the default running top title "PAGE &." If you use the .RT [Running Title] control word and include an &, only the page number and not the text is suppressed.

- If FRAC is specified while the page numbers are represented in ROMAN or ALPHA numerals, the page number that is printed is in lowercase Roman or alpha numerals, but the fractional part is in Arabic.

- Table of contents entries generated by .H0 - .H6 [Head Level 0 - 6] or the .PT [Put Table of Contents] control words show the page numbers in the same format they appear on the page, that is, if a prefix is used, it is shown in the table of contents; if Roman numbers are in effect, the contents entry has a Roman numeral, and so on.

- Whenever the page number symbol is substituted, its prefix will also be included. Care must be taken therefore when using the page number symbol as a part of an arithmetic operation on the right hand side of a .SE statement.

- The .PN control word will take effect on the page after it is encountered.

Examples:

1. .pn off

   The internal page count continues to be incremented for each page printed.

2. .pn offno

   No page numbers appear on SCRIPT/VS output, and the internal page count remains at its current setting without further incrementing.

3. .pn on

   Page numbering on SCRIPT/VS output resumes using the current internal page count; this count is incremented for each page printed.

4. .pn roman

   The page number in the title at the bottom of the page after this one appears as a Roman numeral.

   The control word

   .pn arabic

   restores Arabic numbering on the next page.

.PP [PARAGRAPH START]

Use the .PP [Paragraph Start] control word to start a new paragraph.

.PP	[line]

Where:

line is the text that begins a new paragraph. If line is omitted, the text from the next input line after the .PP control word begins the new paragraph.

Notes:

- When the .PP control word is encountered, a break occurs, a skip is generated, and the next line of text is indented three character spaces to the right of the current margin. The .PP control word is equivalent to the control words:

  .sk
  .il +3

  If these values are not satisfactory for your paragraph formatting, you can redefine the .PP control word as a SCRIPT/VS macro.

Example:

The input lines:

```
.pp This line begins with
a .PP control word.
Here is some more text to show
the formatting.
```

Are formatted as:

   This line begins with a .PP control word. Here is some more text to show the formatting.

.PS [PAGE NUMBER SYMBOL]

The .PS [Page Number Symbol] control word allows you to change the page number symbol used in running top and bottom titles and running headings and footings. The default page number symbol is the ampersand (&) character. The .DC PS [Define Character] control word can also be used to alter the page number symbol.

.PS	[c]

Where:

c specifies the character to be used as the page number symbol. It may be any character other than a blank. If it is omitted, no character is assigned as the page number symbol.

Initial Setting: Ampersand (&)

Default: Nothing. (No page number symbol.)

Notes:

- Every occurrence of the page number symbol is replaced with the current page number in running titles, running headings, and running footings, unless .PN OFF or .PN OFFNO is in effect.
- The .PS control word allows you to change the page number symbol currently in effect. The initial page

number symbol is the ampersand (&) character. It may be necessary to change the page number symbol if the & character is not a valid character on your terminal keyboard or the & character is needed as a regular character in your title text.

- This control word affects all running top and bottom titles and all running headings and footings, including those that have been previously defined. Thus, if a title has been set by the control word:

  ```
  .rt t ///Page &/
  ```

  and later the control word:

  ```
  .ps ?
  ```

  is encountered, the top-title must be reset to:

  ```
  .rt t ///Page ?/
  ```

  Otherwise, the current page number will not be substituted into the title.

- Do not confuse the page number symbol with the ampersand used on the right-hand side of a .SE [Set Symbol] control word. A single ampersand in a .SE control word always means that the symbol is to be set to the current page number.

  ```
  .se currpage = &
  ```

  sets the symbol 'currpage' to the current page number, regardless of what character, if any, is defined as the page number symbol.

.PT [PUT TABLE OF CONTENTS]

Use the .PT [Put Table of Contents] control word to add lines or control words to the file which is used to generate the automatic table of contents.

.PT	{ line } { line }

Where:

line is any text line or control word line that you want in the table of contents. This line may be preceded with one or more extra leading blanks (other than the blank that delimits the control word name), and these extra blanks will be removed before the line is written into the table of contents file.

If 'line' is text, it is written to the file DSMUTTOC as part of a .SX [Split Text] control word, which causes it to be formatted as a table of contents line when DSMUTTOC is processed. (See the discussion of the .SX control word.)

If 'line' is a control word, it is written into the DSMUTTOC file directly, and it is executed when the DSMUTTOC file is processed.

If 'line' is specified with extra leading blanks, it is taken as a line of text, even if the first nonblank character is a period. The extra leading blanks are removed, and a .SX control word is built for the DSMUTTOC file, using the first nonblank character as the beginning of the data.

Notes:

- For text lines, the .PT control word generates a .SX control word to be written into the table of contents utility file in the form:

  ```
  .SX F /text line/ ./33/
  ```

  where the page number used is the actual page number when the .PT is processed, and the delimiter used is actually hexadecimal 00. The .PT control word does not accept lines that begin with hexadecimal 00 as valid lines; such lines result in an error message.

- This control word is especially useful for defining heading levels with the .DM [Define Macro] control word. The internal macros that process the head level control words .H0 - .H6 [Head Level 0 - 6] use .PT to write the required information into the table of contents file.

- The .PT control word is ignored while a table of contents is actually being formatted.

Examples:

1. ```
   .pt .pa
   ```

   This line places the .PA control word in the table of contents so
   ```

that when the table of contents file is being processed, a page eject occurs at this point. You may do this if you want separate content sections to appear on different pages.

2. .pt .pa

   Since the line given has extra leading blanks, it is a text line, not a control word. The leading blanks are removed, and a .SX control word is built, using the characters ".pa" as the data:

   ```
   .sx f /.pa/ ./33/
   ```

   (The head level control word macros insert a leading blank in front of a line to be written to the table of contents with .PT when it is known to be text.)

3. .pt .h3 this is a head level 3

   In this case, the control word .h3 is written into the table of contents file because the period appears in the first available position with no extra leading blanks. Any head level that is written into the table of contents file in this way is processed as a heading when the table of contents is actually formatted. A normal head level 3 is generated at that point in the table of contents, but no attempt is made to write any more information into the table of contents utility file. In other words, the .PT function of the macro for .H3 is ignored while the table of contents is actually being formatted.

.QQ [QUICK QUIT]

The .QQ [Quick Quit] control word causes SCRIPT/VS processing to terminate immediately, without the usual final page eject.

.QQ	

Notes:

- Since SCRIPT/VS does not perform a final page eject after encountering the .QQ control word, some output that has been formatted may never be displayed.
- The .QQ control word is useful when you are using the .TE [Terminal Input] control word to enter lines from the terminal, and you want to terminate processing quickly.

.QU [QUIT]

The .QU [Quit] control word causes processing to terminate with a final page eject.

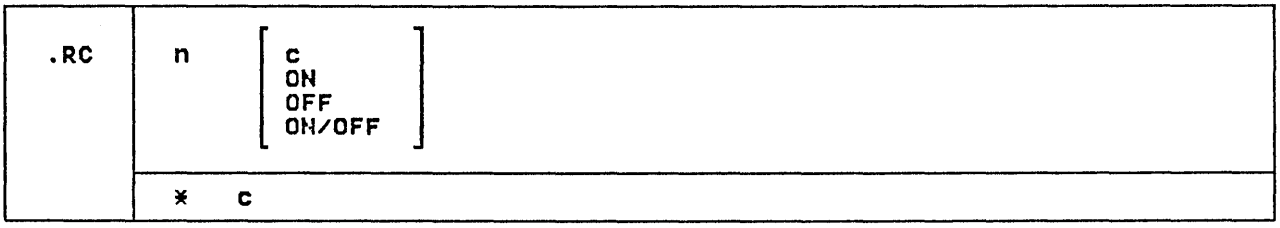
.QU	

Notes:

- The .QU control word causes a final page eject so that the last partial page of formatted text may be printed.
- The .QU control word will cause termination no matter where or when it is encountered, including within imbedded files (see the .IM control word). All open SCRIPT files are closed before processing terminates.

.RC [REVISION CODE]

The .RC [Revision Code] control word allows you to designate a revision code marker to be printed to the left of the column.



Where:

n specifies the revision code number from 1 to 9.

c specifies the revision code character to be printed along the left margin. It may be any single character, including the blank. If not specified, a blank character is assumed.

ON signifies the beginning of text to be marked with the code character associated with RC n.

OFF signifies the end of text to be marked with the code associated with RC n.

ON/OFF signifies that the next output line should be marked with the RC n code on output.

***** marks the next output line with the specified revision code (like the ON/OFF option). Unlike the ON/OFF option, the * option allows you to specify any character for this one occasion without associating it with a revision code number.

Notes:

- The .RC control word has two functions:

  1. to define a revision code symbol and
  2. to activate the revision code.

  You may have up to 9 revision codes defined at any time, and each revision code may be assigned a different character. The operands ON and OFF activate and deactivate the actual revision code marking. The operand ON/OFF has the effect of turning ON revision code n for one line only; the line that is next printed after the .RC n ON/OFF is processed.

- By assigning different symbols to different revision code numbers, including the blank, it is possible to selectively print specific revision code markers or differentiate between various levels of revision.

- Since the .RC control word does not cause an automatic break, revision code markings may be turned on and off within a paragraph or even a sentence without disrupting normal SCRIPT/VS formatting. An explicit .BR control word may be necessary under certain circumstances to cause the last unrevised line to be finished before formatting begins on the revised material.

- The revision code for the leftmost column is placed in the binding that is specified with the BIND command option. The revision code for other columns is placed in the intercolumn gutter. If there is not at least two character spaces of binding or gutter, the revision code is omitted.

- Revision codes may be nested to a depth of 9. This is useful in circumstances where revisions are made to sections that have already been revised. If a revision code is turned ON while another is ON, the first is stacked. It is neither ON nor OFF. When the inner RC is turned OFF, the stacked RC is turned ON again. Only one RC is ON at a time.

- If you attempt to redefine a revision code character while that revision code is ON, an error message is issued.

- The revision code status is subject to the .SA [Save Status] and .RE [Restore Status] control words. If you have .RC 3 ON, then .SA, then .RC 3 OFF, then .RE, the status is restored as it was before the .SA (that is, the revision code is turned back on).

Example:

```
.rc 1 1
.rc 2 *
(input)
.rc 1 on
"This writeup applies to version 5."
.rc 1 off
```

The marker for revision code 1 is defined to be a number one (1) and the marker for revision code 2 is defined to be an asterisk (*). All other revision code markers are defined to be blank by default. The line or lines of printout that contain the sentence

```
1 "This writeup applies to version 5."
```

will be noted by a number 1 printed along the left margin.

.RD [READ TERMINAL]

The .RD [Read Terminal] control word allows you to enter a line from the terminal during SCRIPT/VS processing. SCRIPT/VS does not process this line in any way.

.RD	[1 / n]

Where:

n specifies the number of lines to be read at the terminal. If omitted, 1 is assumed.

Notes:

- The .RD control word is meaningful only when the formatted output is actually being typed at your terminal in interactive environments. The line or lines typed are not processed by SCRIPT/VS, but they appear in the output exactly as they are typed.
- The .RD control word causes a break and a section break. All lines read by .RD are read while SCRIPT/VS is in a single column mode. After the .RD is finished, the previous column definition is resumed.
- If the output is not being typed at a terminal, the .RD control word causes a break and a section break, and then spaces down as many lines as the "n" value specified, but in a single column mode. In this case, .RD acts very much like ".SP n P".
- As SCRIPT/VS reads lines from the terminal, it accounts for the space they occupy on the page. If the end of a page is reached before the number of lines to be read is exhausted, SCRIPT/VS takes control, performs a page eject to start a new page, and then continues reading the remaining lines.

.RE [RESTORE STATUS]

The .RE [Restore Status] control word restores the status of the SCRIPT/VS variables that were previously saved by the .SA [Save Status] control word.

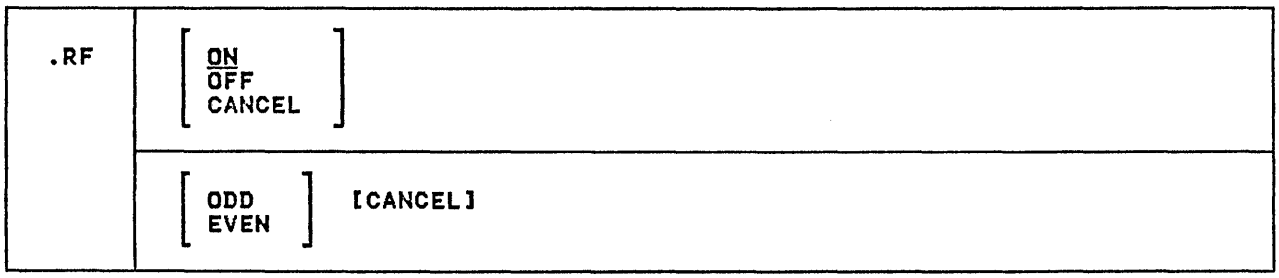
.RE	

Notes:

- The .RE control word restores the status of certain SCRIPT/VS variables from the last-in-first-out stack created by the .SA control word. The .RE control word restores the SCRIPT/VS variables to values that were in effect at the time of the corresponding .SA control word. See the description of the .SA control word for additional information.
- If there is no currently active .SA control word, the .RE control word restores the initial values. Each .RE control word effectively cancels a corresponding preceding .SA control word.

.RF [RUNNING FOOTING]

Use the .RF [Running Footing] control word to specify that the following lines of text are to be saved as a running footing for subsequent pages.



Where:

ON identifies the following lines as a running footing to be saved and placed on every subsequent page. ON is the default.

OFF indicates that the specification of the running footing is ended.

ODD specifies that the following lines are to be saved as the running footing for odd-numbered pages only.

EVEN specifies that the following lines are to be saved as the running footing for even-numbered pages only.

CANCEL may be used with the ODD and EVEN parameters, or by itself to cancel running footings defined with the ODD, EVEN, or ON parameters.

Default: ON

Notes:

- The running footing will be placed on the page immediately above the space which is defined by the .BM [Bottom Margin] control word. It is formatted to the current line length, and all page number symbols are replaced by the current page number.
- The running footing defined with the .RF control word will take effect on the page after the .RF control word is encountered.
- Implicit termination of the footing text is caused by starting a new running heading or footing, or by any control word that is not allowed in a running footing. These control words are the same as those that are not allowed in a keep, and are described in Figure 27 on page 312.
- Some control words are processed once when the running footing is encountered, and others are processed every page as the footing is being formatted to be put on the page. A list of the control words that are processed immediately is given in Figure 28 on page 312.

.RH [RUNNING HEADING]

Use the .RH [Running Heading] control word to specify that the following lines of text are to be saved as a running heading for subsequent pages.

```
.RH  [ ON     ]
     [ OFF    ]
     [ CANCEL ]

     [ ODD  ]  [CANCEL]
     [ EVEN ]
```

Where:

ON identifies the following lines as a running heading to be saved and placed on every subsequent page. ON is the default.

OFF indicates that the specification of the running heading is ended.

ODD specifies that the following lines are to be saved as the running heading for odd-numbered pages only.

EVEN specifies that the following lines are to be saved as the running heading for even-numbered pages only.

CANCEL may be used with the ODD and EVEN options, or by itself to cancel running headings defined with the ODD, EVEN, or ON options.

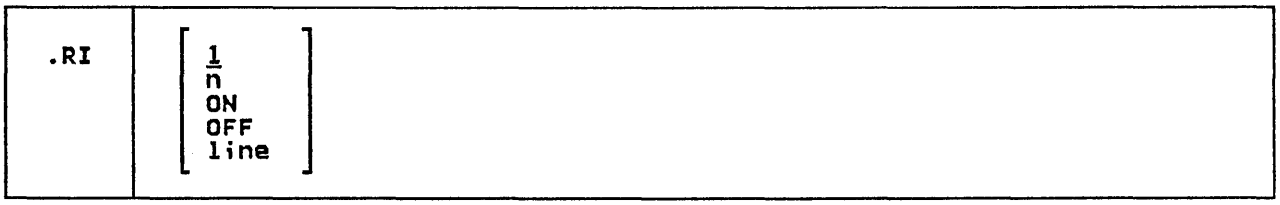
Default: ON

Notes:

- The running heading will be placed on the page immediately below the space which is defined by the .TM [Top Margin] control word. It is formatted to the line length, and all page number symbols are replaced by the current page number.
- The running heading defined with the .RH control word will take effect on the page after the .RH control word is encountered.
- Implicit termination of the heading text is caused by starting a new running heading or footing, or by any control word that is not allowed in a running heading. These control words are the same as those that are not allowed in a keep, and are described in Figure 27 on page 312.
- Some control words are processed once when the running heading is encountered, and others are processed every page as the heading is being formatted to be put on the page. A list of the control words that are processed immediately is given in Figure 28 on page 312.
- The function of the SCRIPT/370 Version 3 control word .HN [Headnote] is provided by the .RH control word. Thus, you cannot have a SCRIPT/370 headnote and a SCRIPT/VS running heading at the same time.

.RI [RIGHT ADJUST]

Use the .RI [Right Adjust] control word to position an output line flush with the right margin.



Where:

n specifies the number of lines to be right adjusted. If omitted, 1 is assumed. If .RI n is specified when .RI ON is in effect, right adjusting is turned off when n lines have been right adjusted, or when .RI OFF is encountered.

ON specifies that subsequent text lines are to be right adjusted.

OFF terminates right adjust mode if it was ON, or if n has been specified and has not been exhausted.

line is a line of text to be right adjusted. The line is considered to start with the first nonblank character after the .RI control word.

Default: 1

Notes:

- The keywords ON and OFF, and a number of lines to be right adjusted (n), must be the only parameter on the control word line. A string of words that happens to start with one of these is interpreted as a single line to be right adjusted. For example, the control word lines:

  ```
  .ri on top of old smokey
  .ri 555 Bailey Ave.
  ```

  are taken to be of the ".RI line" form, not requests for large numbers of lines to be right adjusted.

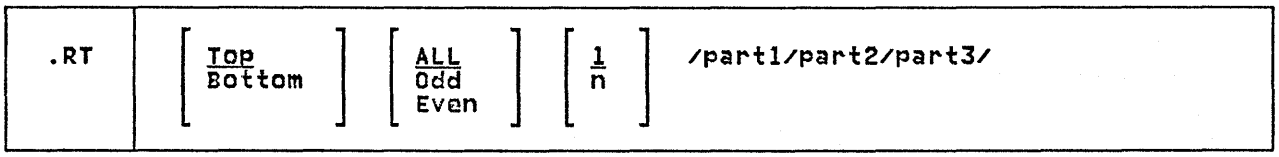
- When right adjusting is in effect, no formatting is done on the line. That is, the line is right adjusted as it stands, and it is not filled from other input lines or justified. If a tab character appears in the line to be right adjusted, the tab is resolved before the line is right adjusted.
- This control word acts as a break.
- If the line to be right adjusted is longer than the current column length, it is truncated, and the excess is used on a second line.
- The .CE [Center] control word is a variant of .RI. If either of these control words is processed, the other is cancelled.
- Contrast this control word with .FO RIGHT. The latter allows lines to be formatted by concatenating words until the line is nearly full, but then the filled line is right adjusted instead of being justified, as would be the case with .FO ON.

Example:

```
.ri 3
                    These three lines are
                          right-adjusted,
                          as you can see.
```

.RT [RUNNING TITLE]

The .RT [Running Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the top or bottom of the next page and each subsequent output page.



Where:

Top specifies that this control word refers to top titles. The TOP keyword may be abbreviated as T. This is the default.

Bottom specifies that this control word refers to bottom titles. The BOTTOM keyword may be abbreviated as B.

Odd specifies that the title being defined is to be printed on odd numbered pages only. The ODD keyword may be abbreviated as O.

Even specifies that the title which is being defined is to be printed on even-numbered pages only. If neither ODD nor EVEN is specified, the title being defined will be printed on both even- and odd-numbered pages. The EVEN keyword may be abbreviated as E.

ALL specifies that the title is to be printed on both odd- and even-numbered pages. ALL is the default.

n is the number of the title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "top title 1" sets the same storage buffer as "bottom title 6." See the discussion of the .HS [Heading Space] and .FS [Footing Space] control words for information on how to allocate space on your output page for top and bottom titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any delimiter character that does not appear in part1, part2, or part3.

Default: TOP ALL 1

Notes:

- Every occurrence of the page number symbol in part1, part2, and part3 is replaced with the current page number on each page where a title appears, unless .PN OFF or .PN OFFNO is in effect. The character designated as the page number symbol may be changed with the .PS [Page Number Symbol] control word or the .DC PS [Define Character] control word.
- Symbol substitution and character translations set up by the .TR [Translate Character] control word are done on part1, part2, and part3 when the .RT control word is processed, not on every page.
- The three parts of the title are used to form the actual title that is to be saved for future use. This title may be printed at the top or bottom of each subsequent output page, if space has been allocated for it using the .HS or .FS control words.
- The specific location of the top titles on the page is controlled by the .TM [Top Margin] and .HM [Heading Margin] control words; the number of top titles to be used on each page is controlled by the .HS [Heading Space] control word.
- The specific location of the bottom titles on the page is controlled by the .BM [Bottom Margin] and .FM [Footing Margin] control words; the number of bottom titles to be used on each page is controlled by the .FS [Footing Space] control word.
- Any title may be changed by including another .RT control word later in the file.

- The default top title, printed on each page of output after page one, is

  PAGE &

  which is right-justified at the top of the page. This title may be suppressed with the .PN OFF control word.

- This control word will take effect on the page after it is encountered.

- The length of the title will be that of the line length as set by the .LL control word.

- The parameters may be specified in any order, and if contradictory options are specified, only the latest one will be used. The first character that is not recognized as an option will be taken as a delimiter.

Example:

```
.rt t 'heading''PAGE &'
```

The heading and the current page number will be printed at the top of all subsequent pages, unless the heading space has been set to zero.

.RV [READ VARIABLE]

The .RV [Read Variable] control word is similar to the .SE [Set Symbol] control word, except that the value of the symbol is read from the terminal.

.RV	symname [= ']

Where:

symname is the name of the symbol to be set. It may be any name that would be allowable on the left-hand side of the equal sign in a .SE [Set Symbol] control word.

= ' indicates that the value set into the named symbol is to be treated as a quoted string. If you do not specify the equal sign and the single quote, SCRIPT/VS provides the equal sign automatically, and processes whatever string is entered according to the rules for the value on the right-hand side of the equal sign in .SE [Set Symbol] control words. In this case, any value that requires single quotes must have the quotes explicitly supplied as part of the value entered from the terminal.

Notes:

- When the .RV control word is encountered, a line is read from your terminal. This line is used as the right-hand side of the equal sign to set the value of the symbol named in the .RV control word. Any expression that would be allowable as the value in a .SE control word is allowable here. If no name is given on the .RV control word, it is ignored, and no line is read from the terminal.

- The .RV control word does not cause an automatic break.

- No message is displayed before the terminal is unlocked to accept the input line. You may use the .TY [Type on Terminal] control word to issue a prompting message before the .RV control word issues its terminal read.

Example:

A symbol called "name" could be set with the following control word:

```
.se name = 'John Doe'
```

The same symbol could also be set this way:

```
.rv name = '
```

At this point, SCRIPT/VS issues a read to your terminal, and you may enter the material to be used as the value of the symbol. In this example, you would enter:

```
John Doe
```

You must use single quotes in the same circumstances where they would be required in a .SE control word, unless the .rv name =' form is used.

.SA [SAVE STATUS]

The .SA [Save Status] control word saves the SCRIPT/VS formatting environment, which consists of the values and dimensions of certain control words. If any of these control words is processed, the environment is changed accordingly. The .RE [Restore Status] control word restores all the environment values to the settings that were in effect before the .SA control word was issued.

.SA	

Notes:

- The .SA control word saves environments in a stack. The .RE [Restore Status] control word restores the SCRIPT/VS environment to the values that were in effect at the time of the most recent .SA control word.
- The .SA control word can save up to five different environments before a .RE [Restore Status] control word is required. If a sixth .SA is encountered, a message is issued, indicating that the save stack has overflowed, and the control word is ignored.
- The .SA control word only saves a copy of the values of these SCRIPT/VS variables, it does not change any of these variables.
- Since .SA does not change any of the SCRIPT/VS variable settings, all variables should be explicitly set to the values appropriate unless the current settings are known. For example, you can explicitly set indention to 0, and then restore it to whatever it was previously.
- The control word values in the saved environment are listed in Figure 32 on page 315.
- The environment saved by .SA is divided into three parts, the "active environment," "page control," and translate tables. The active environment is automatically saved and restored for some keeps (see the discussion of the .KP [Keep] control word) and for footnotes. It is not necessary to use .SA and .RE within keeps and footnotes unless you want to save and restore values that are not in the active environment, such as .TR [Translate Character] specifications.

.SC [SINGLE COLUMN MODE]

The .SC [Single Column Mode] control word saves the current column definition and starts a temporary single column format. The .MC [Multicolumn Mode] control word restores the column definition that was saved by .SC.

.SC	

Notes:

- The .SC [Single Column Mode] control word temporarily starts formatting in a single column that is the same width as the current .LL [Line Length] specification. The .MC [Multicolumn Mode] control word restores the column definition that was in effect before the .SC was processed.
- More than one .SC control word may be processed without an intervening .MC. Each .MC clears one .SC, and, until the first .SC in the list is cleared, the column definition restored by each .MC is a single column definition that was set up by an earlier .SC.
- The .CD [Column Definition] control word starts an entirely new column definition, and clears all .SC's and .MC's that may be in effect.
- This control word is not allowed in a keep.
- The .SC control word starts a new section. Therefore, skips inserted by the .SK [Skip] control word are discarded, since they would appear at the top of a column.

.SE [SET SYMBOL]

The .SE [Set Symbol] control word allows you to define and assign values to symbols or arrays of symbols. Using the .SE control word, you can give a symbolic name to a page number, a word, or even a string of SCRIPT/VS control words. The .SE control word itself, or any of its parameters, can be a symbol.

.SE	symname [([n])] =	[symvalue & SUBSTR string [start [length]] INDEX string1 [string2]]
	symname [([n])]	{ OFF } { LIB }

Where:

symname is the name to which you want to assign a symbolic value to be substituted during SCRIPT/VS processing. It may contain a maximum of 10 nonblank characters which may be upper- and lowercase alphabetic, numeric and the characters @, #, and $. You may specify a line number in parentheses for array symbols, except when you use the LIB parameter. An array line number is also called an element number or a subscript.

symvalue assigns a value to the symbol name; it may be a character string or arithmetic expression.

& assigns the symbol name a value equal to the current page number string.

SUBSTR obtains the specified characters (substring) from a given string and assigns them to the symbol provided.

string is the string from which the substring is to be extracted.

start is a positive integer that defines the position of the beginning character in the string which is to be assigned to the symbol. If both start and length values are omitted, the symbol will be assigned the entire value of the string.

length specifies the number of characters to be extracted from the string, in other words, the length of the substring. If length is omitted, the remainder of the string, from the specified start to the end, will be assigned to the symbol.

INDEX searches the string "string1" to see if it contains the string "string2". If it does, the symbol value is set to the position of the starting character of "string2" within "string1".

string1 is a string that is to be searched to see if it contains the string "string2".

string2 is a string that is to be searched for in the string "string1". If string2 is omitted, or has a null value, the symbol will be set to 0.

LIB causes the symbol to be set by retrieving its value from a library. The name of this library may be defined using the LIB option on the SCRIPT command. If the LIB option is used to set a symbol, the value retrieved from the library will replace the current value. If no entry with the symbol name given exists in the library, the symbol will be undefined. Since symbol names in the library are in uppercase only, the same member of the library will be used to define all symbols of the same name that differ only in the case of the characters used. Subscripted symbol names may not be used with the LIB option. The LIB option may be used regardless of the most recent specification of the .LY control word.

OFF unsets the named symbol so that, to SCRIPT/VS, it was never set. An entire array symbol will be set off if no subscript is provided. However, only the specified element will be set off if a subscript is provided.

Symbol Names:

The character '*' has a special meaning as the first character of a symbol name because it denotes that the symbol is a local symbol. This means that it will only have the value that is set at the current level of macro nesting, and every macro that is called has its own set of local symbols for the duration of that macro's execution.

During SCRIPT/VS processing, a symbol name is recognized when it is preceded by an ampersand (&) and followed by a blank or a period:

```
&symname
```

If the symbol name appears in any of the following forms:

```
symname()
symname(n)
symname(&symbol)
```

it is an array symbol.

SCRIPT/VS also recognizes symbol names that are preceded by the GML delimiter, initially and by default, the colon (:). The name of a symbol defined with .SE as a GML tag must be all in uppercase characters, and it cannot be an array symbol. The GML delimiter causes SCRIPT/VS to search for a symbol name that is all uppercase, regardless of the case of the name in the input line.

Symbol Values:

If a symbol value is set to a character string that contains any embedded blanks or any special characters, it must be enclosed in single quotes. For example,

```
.se dog = cat
.se end = '.qu'
.se sentence = 'This is a sentence.'
```

are all valid character strings. If you want a character string to contain a single quote ('), you must enter two of them, for example

```
.se title = 'Mrs. O''Grady''s Cat'
```

If you want to use the INDEX or SUBSTR parameter of .SE to operate on a portion of the string "Mrs. O'Grady's Cat," which is the value of the symbol &title, you should turn substitution OFF with the .SU [Substitute Symbol] control word before issuing the .SE control word. If substitution is ON, the control word line

```
.se syma = index &title Cat
```

does not work properly because substitution is performed on the line before the .SE control word processes it. The substituted line already has the string "Mrs. O'Grady's Cat" on the right-hand side of the equal sign, and the .SE control word misinterprets the internal blanks as delimiters between parameters. If substitution is OFF, the .SE control word receives the line in its unsubstituted form, and the parameter on the right-hand side of the equal sign is the character string "&title". Even though substitution is OFF, the .SE control word can retrieve symbol values when it recognizes symbol names on the right-hand side of the equal sign. In this case, the .SE control word knows that the entire value of the symbol &title constitutes the "string1" parameter for this .SE control word.

If the symbol value is an arithmetic expression, it must be in the form:

```
[op1] n op2 n op2 n op2 n...
```

where:

op1 is a unary + or - sign.

op2 is an arithmetic operator:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)

n is a valid integer of length less than 9 digits. Lengths greater than this may produce unpredictable results. The integers may have been assigned their values as a result of a symbol substitution (including the page number symbol).

For example,

```
.se nextpage = & + 1
.se current = -100
.se addit = &current + 25
.se answer = 15 - 42
```

are all valid arithmetic expressions.

Notes:

- In symbol names, uppercase and lowercase letters are considered to be different, thus the symbols symbol1, Symbol1, and SYMBOL1 are three distinct symbols. Symbols whose names start with the dollar sign ($) are system symbols, and they exist only in an uppercase form. The reserved system symbols which may not be set by the user, such as &$RET, are in this category. Although you can set a symbol whose name starts with $ if that name is not in use as a read-only system symbol, this is discouraged; confusion can occur due to the name folding. Symbols preceded by the GML delimiter can be recog-

nized only if a symbol (GML tag) has been defined with the specified name all in uppercase. During substitution, a symbol name that begins with $ or a symbol preceded by the GML delimiter is folded to uppercase before being resolved.

- An iterative substitution, as described in the .SU [Substitute Symbol] control word discussion, is automatically performed on all character string symbol values.

- If the symbol value is omitted, the symbol's value is set to a null character string (length zero).

- The symbol for the current page number, &, remains the same even if the page number symbol that is used in running titles and running headings and footings is changed with the .PS [Page Number Symbol] or .DC PS [Define Character] control word.

- If you set a symbol name equal to the current page number (.SE refer = &) within a keep, the symbol is actually set twice. The page where the keep will finally be located is not known until the keep is ended and measured. When a .SE control word sets a symbol to the current page number, the symbol is set immediately, and then it is set again when the page number of the keep is known. If you refer to this symbol before the second setting, the number may be inaccurate.

- Arithmetic expressions in set statements are evaluated strictly from left to right, and no operator takes precedence over another. For example, the expression:

  ```
  .se x = 1 + 2 * 4 + 6
  ```

  will set the symbol x to the value 18.

- See the discussion of the .SU [Substitute Symbol] control word for more information about symbol substitution.

- The LIB option of the .SE control word allows a symbol to be explicitly retrieved from the library. The .LY [Library] control word allows a symbol value to be retrieved from the library when it is used in a document and when a value for it does not currently exist. When a symbol value is once retrieved from the library, it is stored in the symbol table for future use.

- You should be careful when using local symbols and page number symbols in arithmetic set statements with the multiplication operand (*). The expression .se a = &*3+1 will be taken as a request to add 1 to the value of the local symbol &*3, not as a request to multiply the page number by 3 and then add 1. To achieve the latter effect, the page number symbol must be delimited (.se a = &.*3+1).

- When symbol substitution is ON, a .SE control word line is completely substituted before it is processed. When substitution is OFF, the .SE control word can still perform individual substitutions on symbolic values in the control word. See the next two notes for examples.

- Be careful of the effects of substitution on arithmetic set statements when symbols that contain negative numbers are used. For example,

  ```
  .se a = -3
  .se b = 5+&a
  ```

  will result in an invalid expression if substitution is on, as the line will be substituted as:

  ```
  .se b = 5+-3
  ```

  which is invalid. However, if substitution is OFF, the .SE control word processor can see that you want to add 5 to -3, and can do it correctly.

- Substitution also has an effect on the .SE control word if strings are to be set which are longer than 16 characters. SCRIPT/VS will treat a single character string without special characters or blanks as a character string even if it is not enclosed in quotes, if it is not more than 16 characters long. If the string is longer than this, an error will result. For example:

  ```
  .se a = '12345678901234567890'
  .se b = index &a 1
  ```

  will result in an error if substitution is on because the symbol &a is not enclosed in quotation marks. The error will not happen if substitution is off.

.SF [SAVE FONT]

Use the .SF [Save Font] control word to save the current font identification.

.SF	

Notes:

- The font-id saved with .SF is restored by the .PF [Previous Font] control word.
- You may save up to 16 font-ids with .SF. If you issue more than 16 .SF control words without an intervening .PF, the oldest saved font-id is lost.

.SK [SKIP]

Use the .SK [Skip] control word to generate blank vertical space before the next text output line, except at the top of a column or page.

.SK	[1 / v] [C] [A] [P]

Where:

v is the amount of space to be inserted in the output. If no number is given, 1 line is assumed. If the size in "v" is not qualified as any of the other space units (inches, picas, ciceros, or millimeters), it is a request to skip a number of lines. In this case, unless A is specified, the size of the request is multiplied by the appropriate factor if double spacing or multiple spacing is in effect.

C indicates conditional skips. These skips depend upon what follows them in the output column. If conditional skips are followed by a line of text, they appear in the column as requested. If they are followed by another skip or space request, the two skip or space requests are compared, and only the larger of the two remains in the column.

A indicates absolute skips. If the vertical size of the skip given in "v" is expressed in inches, picas, ciceros, or millimeters, it is already an absolute number, and the actual requested depth will be skipped, to the closest approximation possible on the current logical device. In this case, A need not be specified.

P indicates page skips. These skips will generate skip space across the full width of the page, even when formatting in multiple columns. Since this type of skip causes a section break, it is not allowed in a keep.

Notes:

- No blank space is generated if it would be the first to be printed at the top of a column of output. The top of a column may be at the top of the page or after a section break. If the blank space would not fall at the top of a column, the .SK control word is identical to the .SP [Space] control word. If the column is partially filled, and a .SK control word is encountered requesting more space than remains in the column, only enough space to fill the column is generated, and then all the rest (at the top of the next column) is ignored.
- Page skips (the P parameter) are ignored if they fall at the top of a page, but not if they fall elsewhere on the page.
- If double spacing is in effect, the number of skips generated is multiplied by the line spacing amount, unless absolute spacing is specified.
- This control word acts as a break.
- If the skip request is in lines (unqualified space units), the size of each line is as defined with the .SL [Set Line Space] control word.

.SL [SET LINE SPACE]

This control defines the vertical distance from the baseline of the current line to the baseline of the following line.

.SL	[vsize]

Where:

vsize is the the vertical size of all following formatted output lines until redefined with another .SL.

Initial Setting: One logical device print line.

Default: One print line.

Notes:

- The vertical size of formatted output lines is set by .SL to the nearest approximation of the requested size that is possible on the current logical device.
- The .SL value is used for formatted lines and for requests in lines for the following control words:

  .CC [Conditional Column Begin]
  .CP [Conditional Page Eject]
  .SK [Skip]
  .SP [Space]

  In all other control words that can have a vertical dimension expressed in lines, such as .PL [Page Length] and .TM [Top Margin], the size of the request is based on the size of a print line on the current logical device. For example, if the logical device were an 8 line per inch printer, the control word .TM 4 would set the top margin to 4 lines, or one-half inch, regardless of the .SL value.

.SP [SPACE]

Use the .SP [Space] control word to generate blank vertical space before the next text output line.

.SP	[1 / v] [C] [A] [P]

Where:

v is the amount of space to be inserted in the output. If no number is given, 1 line is assumed. If the size in "v" is not qualified as any of the other space units (inches, picas, ciceros, or millimeters), it is a request to space a number of lines. In this case, the size of the request is multiplied by the appropriate factor if double spacing or multiple spacing is in effect, unless A is specified.

C indicates conditional spaces. These spaces depend upon what follows them in the output column. If conditional spaces are followed by a line of text, they appear in the column as requested. If they are followed by another skip or space request, the two skip or space requests are compared, and only the larger of the two remains in the column.

A indicates absolute spaces. If the vertical size of the space given in "v" is expressed in inches, picas, ciceros, or millimeters, it is already an absolute number, and the actual requested depth will be spaced, to the closest approximation possible on the current logical device. In this case, A need not be specified.

P indicates page spaces. These spaces will generate space across the full width of the page, even when formatting in multiple columns. Since this type of space causes a section break, it is not allowed in a keep.

Notes:

- If double spacing is in effect, the number of spaces generated is multiplied by the line spacing amount, unless absolute spacing is specified.

- This control word acts as a break.
- If the space request is in lines (unqualified space units), the size of each line is as defined with the .SL [Set Line Space] control word.
- If a page eject occurs while SCRIPT/VS is processing a .SP control word, remaining blank lines are inserted after the top titles and running heading on the following page. If you do not want spaces to appear at the top of the page, use the .SK [Skip] control word.
- Spacing via .SP for greater than the number of lines left in the column may produce undesirable results if column balancing is in effect. This is because the space will be balanced across all columns that were not completely filled by the space.

.SS [SINGLE SPACE MODE]

Use the .SS [Single Space Mode] control word to cancel a previous .DS [Double Space Mode] or .LS [Line Spacing] control word, and to resume single-spacing of output.

.SS	

Notes:

- This control word does not cause a break.
- Output following the .SS [Single Space Mode] control word is single spaced. Since this is the normal output format, .SS is needed only to cancel a previous .DS [Double Space Mode] or .LS [Line Spacing] control word.

.SU [SUBSTITUTE SYMBOL]

Use the .SU [Substitute Symbol] control word to cause SCRIPT/VS to stop substitution of defined set symbols or to restore substitution.

.SU	[1 / n / ON / OFF / line]

Where:

n specifies the number of lines to be scanned for set symbols to be substituted. If omitted, 1 is assumed.

ON turns on an open-ended substitution mode. ON is the initial setting.

OFF turns off substitution mode if it was ON, or if n was given and is not yet exhausted.

line is a line containing symbols that you want SCRIPT/VS to substitute with values previously set. Symbols may be set via the .SE, .RV, .IM, or .AP control words, or by a macro call.

Initial Setting: ON

Default: 1

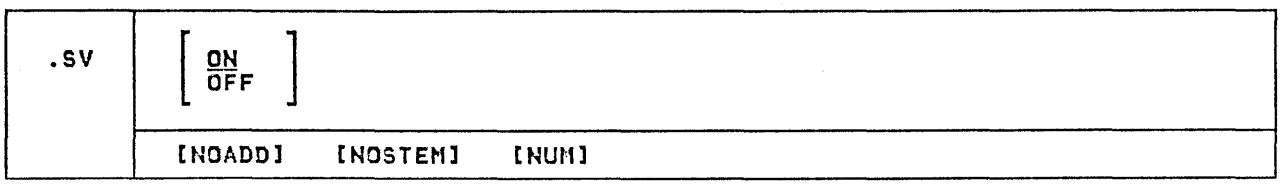
Notes:

- The .SU control word causes a specified number of the following input lines, control words as well as text, to be scanned for defined set symbols. If the argument ON is in effect, every line up to a subsequent .SU OFF will be scanned. Substitution ON is the initial mode of operation, but it is reset to OFF with .SU OFF; with .SU n, after n lines have been read; or with ".SU line" after the line is scanned.

- When an input line is substituted, each complex symbol may go through several stages of substitution until no further substitution can be done. Any "symbol name" for which no definition exists is left in the input line as text.
- The substitution of set symbols may increase or decrease the length of the text line. If the line's length reduces to zero, it becomes a "null line."
- The TWOPASS option of the SCRIPT command may result in defining symbols during the first pass that can be substituted during the second, even though these symbols are defined physically later in the SCRIPT file. If the length of the symbol value and the length of the symbol name are grossly different, the formatting may come out slightly differently in the two passes.

.SV [SPELLING VERIFICATION]

Use the .SV [Spelling Verification] control word to cause spelling checking to start and stop. This control word must be enabled by the SPELLCHK option of the SCRIPT command. If the SPELLCHK option is not in effect, the .SV control word is ignored.



Where:

ON specifies that spelling verification is to be started. This is the default.

NOADD The addenda dictionary will be searched, stem processing will be performed, and words that contain numeric characters will not be checked. Turns on verification if it was off and inhibits use of the addenda dictionary for spelling checking. This dictionary is created using the .DU [Dictionary Update] control word. If NOADD is specified, only the main dictionary will be used for word verification.

NOSTEM turns on verification if it was off and stops the spelling checking function from performing stem processing on words to be verified. Stem processing is described in more detail in "Chapter 15. Automatic Hyphenation and Spelling Verification" on page 157.

NUM turns on verification if it was off and indicates that spelling verification is to be started for words which contain numeric as well as alphabetic characters. This option allows text that contains numbers to be verified. If ON instead of NUM is specified, words that contain alphabetic characters only will be checked.

OFF stops spelling checking.

Initial Setting: OFF

Default: ON

Notes:

- Each time the .SV control word is used, the settings that control spelling verification are all reset. For example:

  ```
  .sv noadd
  ```

  will stop spelling checking against the addenda dictionary. If this is followed later in the document by:

  ```
  .sv num
  ```

  spelling verification will now start for numbers, and will be resumed from the addenda dictionary.

.SX [SPLIT TEXT]

The .SX [Split Text] control word is used to split a string of text between the left and right column margins, with a filler in between the two.

.SX	[F] /lpart/fill/rpart

Where:

F allows the left part of a split line to be folded if it will not fit in the column. The folding is done according to rules appropriate for creating table of contents entries. The fill string and the right part are never folded; they must fit in the column.

/ is any delimiter character. The first nonblank character will be taken as the delimiter character.

lpart is the string to be placed against the current left margin.

fill is a string of up to eight characters to be used to fill the space between lpart and rpart. If this is not specified, blanks are used. If the fill string is shorter than the space between the left part and the right part, it is repeated. If the fill string is longer than the space between the strings, it is not used. The fill string may not contain tabs or backspaces.

rpart is the string to be placed against the current right margin.

Notes:

- The delimiter character between the strings may be any unique character that does not occur within the strings themselves.
- Any of the three parts of the line may be null.
- The final split line is printed in the font that is in effect when the .SX control word is encountered.
- This control word causes a break.
- The .PT [Put Table of Contents] control word writes .SX control words into the the table of contents file to be processed when the table of contents is formatted. The delimiter used for these internally generated .SX control words is hexadecimal 00.

Examples:

1. Split text with null fill string:

```
.sx /left part//right part/
```

```
left part                                  right part
```

2. A foldable split text, as used in tables of contents:

```
.of 1
.sx f /An example.../ ./282/
```

```
An example of a folded split text
 to demonstrate it  . . . . . .  282
```

3. Split text with null left and right parts:

```
.sx //-+//
```

```
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

.SY [SYSTEM COMMAND]

The .SY [System Command] control word is only supported in the interactive environments of CMS and TSO. In CMS, SCRIPT/VS passes a line to CMS for processing as a CMS or CP command line. For TSO, the line is held until the end of SCRIPT/VS processing.

.SY	line

Where:

line is a CMS, CP, or TSO command line. In CMS, if line is omitted, CMS subset is entered.

Notes:

- Use the .SY control word if you want to perform some CMS or CP command when your SCRIPT file is processed, or, in TSO, if you want some command to be performed after formatting is complete.
- The .SY control word does not cause a break.
- No CMS command or user program is allowed that requires the use of the same area of storage that is being used by SCRIPT/VS. CMS commands that are valid in CMS SUBSET are valid on the .SY command. An invalid SUBSET command results in a return code of -2.
- To test whether a command executed successfully in a SCRIPT file, you can use the .IF control word to test the value of the reserved symbol &$RET. For example:

  ```
  .if &$ret ne 0 .qu
  ```

  causes SCRIPT/VS to terminate processing if the return code from the last executed CMS command is not zero.
- If the command does not exist or was not executed at all, &$RET is set to a negative value. This would be the case for nonexistent commands in CMS, and for all commands in environments other than CMS.
- In the TSO environment, if the .SY control word is used more than once, the commands will be executed in the order in which they were encountered.

Example:

The .IM [Imbed] control word issues an error message if the designated file is not found. The CONTINUE option of the SCRIPT command allows SCRIPT/VS to continue processing after this error. The following .CIM [conditional imbed] macro would allow SCRIPT/VS to test for the existence of a file in CMS before attempting to imbed it, and only imbed it if it is available:

```
.dm cim() /.sy state &*1 * *
.dm cim() /.if &$ret eq 0 .im sg10@sy
```

The macro would be invoked as follows:

```
.cim filename
```

.TB [TAB SETTING]

Use the .TB [Tab Setting] control word to define how tab characters (hexadecimal 05) are to be resolved. They may be changed into a number of blanks or to a string of another character.

.TB	[[f/]h [f/]h ... [f/]h]

Where:

h specifies the horizontal displacements of the tab stops. SCRIPT/VS displaces to the next stop by padding with blanks or other fill character. The sequence must consist of increasing positive values separated by one or more blanks.

If no parameters are specified, the default tab settings (5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, and 80) are restored.

f/ specifies the fill character to be used in displacing through position h. If the fill character is to be the blank, it need not be specified.

Notes:

- This control word acts as a break.
- The tab settings must be increasing. Tab settings that are not so ordered result in an error message.
- Tab characters that are found beyond (to the right) of the last defined tab stop are converted to a single blank.
- 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 (minimum of 24 pels on the 3800), the tab stop after the next is used.
- Fill characters are only supported with monospaced fonts on the 3800. If you use fill characters with proportionally spaced fonts, vertical misalignment may result.
- Backspaces after a tab have the effect of reducing the tab position for non-3800 logical devices, but the distance to be tabbed is never reduced to less than one character space.
- No more than 16 tab stops may be specified with the .TB control word.

Examples:

1. .tb 10 20 */30 40

   Tab positions are interpreted as character positions 10, 20, 30, and 40. If a tab character is processed between positions 20 and 30 of a line, the positions from the current position up through and including position 30 are filled with asterisks (*) instead of blanks. The next character goes in position 31. For example, using the system symbol &$tab to generate tab characters, the line,

   ```
   &$tab.text&$tab.text&$tab.text
   ```

   results in:

   ```
   text     text******text
   ```

2. .tb

   Tab positions revert to default values of 5, 10, 15, etc.

.TC [TABLE OF CONTENTS]

The .TC [Table of Contents] control word causes the automatically generated table of contents to be imbedded and printed. Entries may be placed in the table of contents by head level control words .H0 - .H6 [Head Level 0 - 6] and by the .PT [Put Table of Contents] control word.



Where:

n is the number of page numbers to be reserved for the table of contents. If omitted, 1 is assumed. This operand is meaningful when the table of contents is at the front of the document, and the TWOPASS option is used to process it. On the first pass, the table of contents is empty, but on the second pass, it may occupy several pages. If the page numbers in the table of contents are to be accurate, every entry in the table of contents must have the same page number on both passes. After .TC, if .PA n, or .PN n, explicitly sets the page number before .PT sets anything into the table of contents, then all the pages from the table of contents to the explicit .PA n, or .PN n will be sequentially numbered. If not, the n value on the .TC control word will be used to determine the number of the page after the table of contents.

name is an optional line to be used as the title of the table of contents. If no name is given, the word CONTENTS is used. A head-level 1 (.H1) is generated at the top of the table of contents using the name given or the word CONTENTS.

control is a control word to be processed at the top of the table of contents in lieu of the .H1. If this parameter begins with a period, it is assumed to be a control word, and not a name.

/ signals SCRIPT/VS not to generate any head level 1 for the table of contents. Use this when you want no name on the table of contents, you have no control word to be executed, and you don't want the default name CONTENTS to be generated.

Notes:

- When .TC is encountered, a head level 1 is processed. A page eject is done if not already at the top of a page, but no entry is placed in the table of contents for the head. All table of contents entries that have been saved in the utility file DSMUTTOC are then formatted and printed. The entries come from the head level control words whose definitions call for table of contents entries (by default, the control words .H0 through .H3 cause these entries) plus any explicit .PT control words in the source file.
- The table of contents is formatted according to the line and page dimensions in effect at the time the .TC control word is encountered, not those in effect when the head level was processed. Each line in the table has the revision code and the page number that were in effect when the head level was processed.
- When the table of contents is completely formatted, the utility file is erased. Another page eject is done, and the new page is numbered as though sequential page numbering had occurred and the table of contents had occupied exactly n pages. If the table takes other than n pages, there will be either a gap or an overlap in pagination. If TWOPASS is in effect, the pagination may be allowed to run sequentially if the page number is explicitly set before a .PT tries to write an entry in the table of contents file DSMUTTOC. In this case, it doesn't matter what was specified for n.
- This control word acts as a break. It is not allowed in a keep.
- If the .TC control word is used at the beginning of a document you must be careful that the resolution of symbols during the second pass does not cause the document to

expand or contract in such a way that the page numbers established during the first pass are caused to be invalid.

Example:

See the table of contents of this document for an example of an automatically generated table of contents.

.TE [TERMINAL INPUT]

Use the .TE [Terminal Input] control word when you want to enter text or control lines during the processing of the input file.

.TE	[1 / n / ON / OFF / line]

Where:

n specifies the number of lines that will be accepted from the terminal. If omitted, 1 is assumed.

ON starts an open-ended terminal input mode.

OFF turns off terminal input mode if it was ON, or if n was given and has not yet been exhausted.

line is an input line to be processed. The "line" form is available with .TE because it is a Type 1 control word, but it actually does not read anything from the terminal. The control word:

```
.te read this line
```

causes the line "read this line" to be processed as an ordinary input line, but SCRIPT/VS obviously does not read it from the terminal, because it already has the line.

Notes:

- When the .TE control word is encountered, your terminal keyboard is unlocked to accept input lines. The input lines may be text or control words and are processed as if they had been read from an imbedded file (see the .IM [Imbed] control word). The only exceptions to this are the .GO [Goto] and ... [Set Label] control words, which are not allowed during terminal input. If a numeric operand was specified, terminal input is ended after reading n lines. If no operand was specified, only one line is read from the terminal. If ON was specified, input is accepted from the terminal until ended with .TE OFF. When terminal input is ended, processing reverts back to the line following the .TE control word in the file. If the TWOPASS option of the SCRIPT command is in effect, .TE control words in the input file will be processed on both passes.
- If you use .TE while the formatted output is being displayed at your terminal, the input and output may be interspersed. This can be useful for testing or experimentation, but is not usually appropriate for final output.
- The .RD [Read Terminal] control word merely unlocks the keyboard to allow you to type lines in the midst of the normal terminal output. It does not process what you type. The .TE control word, on the other hand, may be used to enter control words or cause text input to be formatted and to appear in the output when the output is written to a device other than the terminal. The .TE control is in effect an imbed, where the "file" imbedded is your keyboard.
- Use the .TY [Type on Terminal] control word immediately before the .TE control word to display prompting messages.
- If .TE ON was specified, the number of lines to be read is open ended. It can be ended by .TE OFF, but since your keyboard is a simulated imbed file, .EF, .QQ, or .QU will also end it.
- The .TE control word may be used to enter control words to specify a particular processing of the input file, such as revision codes or conditional sections.

- Terminal input may be read from a disk file if the terminal input file name DSMTERMI has been associated with the file or data set name with the .DD [Define Data File-id] control word. See the discussion of .DD for more information.

.TI [TRANSLATE INPUT]

Use the .TI [Translate Input] control word to translate the input text from one input representation to another. **This control word should be used with caution, since this translation will occur before any other processing is done.**

.TI	[s t] ...

Where:

s is the source character to be translated. It may be a single character, or a 2-character hexadecimal code.

t is the desired output representation of the source character. It may be a single character, or a 2-character hexadecimal code.

Default: Restores the initial input translate table.

Notes:

- Multiple pairs of translate characters may be specified with a single .TI control word.
- Translate-character specifications remain in effect until explicitly respecified.
- A .TI control word with no operands causes the translation table to be reinitialized and all previously specified translations to be reset.
- The .TI control word does not cause a break.

.TM [TOP MARGIN]

The .TM [Top Margin] control word specifies the amount of vertical space to be skipped above the text and running heading on output pages, overriding the initial value established for the device.

.TM	[v] [+v] [-v]

Where:

v specifies the amount of vertical space to be skipped at the top of output pages. If no value is specified for v, the default value for the logical device will be used. v must be large enough to accommodate the heading margin and the heading space, both of which are allocated from the top margin area. The top and bottom margins may not fill the page so that there is no room left for formatted text. If the value specified for the top margin is so large that there would not be at least one line available for text, the old top margin is left unchanged, and an error message is issued.

+v or -v increases or decreases the existing top margin by the amount given. The calculated top margin value must fall within the allowed range, or an error message will be issued.

Initial Setting: Dependent upon the logical device for which the document is being formatted.

Default: Restores the initial setting.

Notes:

- When the .TM control word is processed, a new top margin is set for future pages, but it is too late for the new value to take effect on the current page. If you

want to increase the heading margin (.HM) or the heading space (.HS) beyond what the current top margin will accommodate, you must change the top margin value first.

- This control word does not cause a break, and it takes effect on the page after the control word is encountered.
- If you specify .TM 0, the heading margin and the heading space are also made zero automatically. Any other top margin specification that is smaller than the current size of the heading margin plus the heading space cannot be satisfied, and results in an error message.
- An error message is also issued if you try to set the top and bottom margins so that they fill the entire page, without at least one line left for text.

.TR [TRANSLATE CHARACTER]

The .TR [Translate Character] control word allows you to specify the output representation of each character in the source text.

.TR	[s t] ...

Where:

s is the source character to be translated. It may be a single character, or a 2-character hexadecimal code.

t is the desired output representation of the source character.

More than one pair of source and intended output codes may be specified with a single .TR control word.

Default: Restores the initial output translate table.

Notes:

- The .TR control word is primarily of use when the final output device uses a different character set than was used to create the source SCRIPT file.
- The text associated with running title lines (.RT) is translated under control of the translations in effect at the time that the .RT control word was processed. If you change the translations after the running title has been saved for future use, it is too late to affect that running title.
- Since control words are only processed internally, they are never translated by the .TR control word. However, text data associated with a control word (as in running titles and typed messages) can be translated.
- Translate-character specifications remain in effect until explicitly respecified.
- A .TR control word with no operands causes the translation table to be reinitialized and all previously specified translations to be reset.
- The UPCASE option of the SCRIPT command has the same effect as the 26 TRANSLATE CHARACTER control words: ".tr a A;.tr b B; ... ;.tr z Z".
- By using the .IF, .CS, or .TE control words, you may specify different output character sets for different runs with different output devices.
- The .TR control word does not cause a break.
- The hexadecimal codes for each printable character for the various character sets and fonts used by SCRIPT/VS are shown in "Appendix A. SCRIPT/VS Summary" on page 297.
- During the time a translation is in effect, every occurrence of the character is translated to the designated output character in formatted text. You should therefore take care not to translate characters that will be needed during that range. The actual translation is done at various times in the formatting process, depending on the requirements of the logical device for which the document is being formatted. The latest time when a translation can be done is when a line is finished, and is placed in a column. You should assume, therefore, that a trans-

lation will be needed until the next break is done, whether this happens naturally because a line is full, or is forced by a control word that causes a break.

Examples:

1. .tr 0 b0 1 b1 ... 9 b9

   This causes the characters 0, 1, ..., 9 to print as their corresponding superscript characters if they are available in the current font. For example, the formula:

   X2+Y2=Z3

   prints as:

   $X^2+Y^2=Z^3$

2. .tr 40 ?

   This causes all blanks in the file to be translated to question marks (?) on output.

.TT [TOP TITLE]

The .TT [Top Title] control word saves a specified title line in a storage buffer for possible future use. This title may be used at the top of the current page, and each subsequent output page.

.TT	[n] /part1/part2/part3/

Where:

n is the number of the top title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even bottom title 1" sets the same storage buffer as "even top title 6." See the discussion of the .HS [Heading Space] control word for information on how to allocate space on your output page for top titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any character that does not appear in part1, part2, or part3.

Notes:

- This control word is provided for compatibility with SCRIPT/370 Version 3. The same function is provided by the SCRIPT/VS control word .RT [Running Title]. See the discussion of the .RT control word for further information about running titles, including those for the top of pages.

.TY [TYPE ON TERMINAL]

The .TY [Type on Terminal] control word causes one line of information to be displayed at your terminal, or written into the file DSMTERMO, no matter where the SCRIPT/VS formatted output is going.

.TY	text

Where:

text is the line to be typed. It is used only for this message. It does not become part of your document unless the document output is also being typed at your terminal.

Notes:

- When the .TY control word is processed, the text line given is typed at the terminal. This line is not part of the document. SCRIPT/VS does not process the line for output; the line is not justified, or formatted in any way. However, the line is scanned for control word separators and symbols are substituted. The text to be typed is translated according to the .TR [Translate Character] translations currently in effect.
- You may use the .TY control word to issue a prompting message before a .TE [Terminal Input] or .RV [Read Variable] control word.
- The information line printed is not counted as part of the normal output. Thus, if the formatted output is being typed on the terminal, the paper positioning may become incorrect and require manual adjustment. In general, the .TY control word should be used for document-driven messages when the formatted output is going to a printer or to a disk file.
- Contrast this control word with .MG [Message]. The .MG control word allows you to issue a true SCRIPT/VS message. A true message may have any of several different degrees of severity, it may terminate SCRIPT/VS processing, and its destination and final form are controlled by the MESSAGE command option. The .TY control word merely types out a line without any of the function of a true message.

Example:

```
.ty Do you want 2 column output?
.rv answer
.if x&answer ne xyes .go by2col
.cd 2 0 46
.cl 43
...by2col
```

.UC [UNDERSCORE AND CAPITALIZE]

The .UC [Underscore and Capitalize] control word automatically underscores and capitalizes one or more input lines.

.UC	[1 / n / ON / OFF / line]

Where:

n specifies the number of lines to be underscored and capitalized. If omitted, 1 is assumed. If .UC n is specified when .UC ON is in effect, .UC is turned off when n lines have been underscored and capitalized, or when .UC OFF is encountered.

ON specifies that subsequent text lines are to be underscored and capitalized.

OFF terminates underscore and capitalization mode if it was ON, or if n has been specified and has not been exhausted.

line is a single text line to be capitalized and underscored.

Initial Setting: OFF

Default: 1

Notes:

- Use the .UC control word whenever you have a line of data that is to be formatted in capital letters and underscored. This control word provides the combined function of .US [Underscore] and .UP [Uppercase].
- The .UC control word does not cause an automatic break; single words in a sentence may be underscored and capitalized.

Examples:

1. Underscoring and capitalizing a single word:

```
 This sentence has
 .uc one
 word processed by .UC.
```

   results in

   This sentence has ONE word processed by .UC.

2. The .UC control word is a Type 1 control word. The SCRIPT/370 Version 3 .UC control word was not Type 1, but it accepted a single line of text only, like the 'line' form of a Type 1 control word. The SCRIPT/370 control word ".uc 80" would process the string "80", but the same control word in SCRIPT/VS starts .UC mode for 80 LINES. If you have any SCRIPT/370 documents that you want to process with SCRIPT/VS, the following macro may be defined to make .UC operate as the SCRIPT/370 control word:

```
.dm uc() /.'up off
.dm uc() /.'us off
.dm uc() /.'uc 1
.dm uc() /.li &*
```

.UD [UNDERSCORE DEFINITION]

Use the .UD [Underscore Definition] control word to specify which characters should be underscored whenever automatic underscoring is done.

.UD	[{ ON } c c ...] { OFF }

Where:

ON specifies that the following characters are to be underscored.

OFF specifies that the following characters are not to be underscored.

c is either a single character or a 2-character hexadecimal code representing a character that is defined for underscoring (ON) or not underscoring (OFF).

Initial Setting: See Notes.

Default: Restores initial settings.

Notes:

- When a line is automatically underscored, each character is subject to underscoring or not, depending on the current .UD specification. For example, with .UD, you can specify that no capital H should be underscored. If you do not give any parameters with the control word, but put ".UD" alone, the initial definitions are restored.
- The control words that cause underscoring are .US [Underscore], .UC [Underscore and Capitalize] and any .H0 - .H6 [Head Level 0 - 6] control word whose current definition calls for it.
- This control word does not cause a break.
- You may specify as many characters on a .UD control word line as you wish. If you want to change some characters to ON, and others to OFF, or if you want to change more characters than it is practical to specify on a single input line, you may use more than one .UD control

word. Each .UD control word changes only the characters specified, and leaves the rest of the characters unchanged.

- The initial .UD setting calls for all characters to be underscored except for blanks, tabs, punctuation characters, and certain other special characters. The characters that are not automatically underscored are shown in Figure 36 on page 319. Blanks or fill characters generated by a tab character are not underscored.

.UN [UNDENT]

Use the .UN [Undent] control word to cause the next line to be shifted. The current indention is changed for the next line only, then restored to its previous value for subsequent lines.

.UN	[0 / h / +h / -h]

Where:

h specifies the amount of horizontal space by which the indention is to be altered for the next line only. A SCRIPT/VS "undent" is a negative indent. If -h is specified, the .UN control word is effectively the same as the .IL [Indent Line] control word. If omitted, 0 is assumed, and the indention is not changed.

Initial Setting: 0

Default: 0

Notes:

- The .UN control word provides the same function as the .OF [Offset] control word. The choice between using .UN and .OF is usually a matter of personal preference. They may also be used at the same time to control margins that shift both right and left.
- This control word acts as a break.
- The value specified in a .UN control word is subtracted from the current indention (indent value plus offset value) to determine where to format the next line. If the .UN amount exceeds the current indention amount, an error message results.
- If successive .UN or .IL control words with positive or negative specification for h are encountered without intervening text lines, the .UN value is reset to the latest specified value each time.

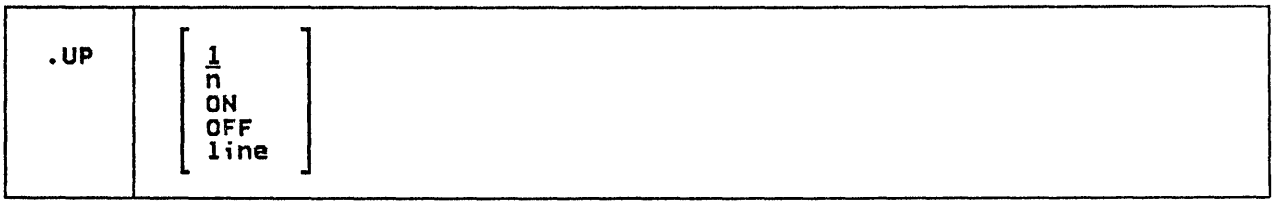
Example:

```
.in 3p
.un 3p
```

If an indention of 3 picas is in effect (as in these lines), the next line is undented to the left margin; all following lines have the normal indention of 3 picas from the left margin.

.UP [UPPERCASE]

The .UP [Uppercase] control word automatically capitalizes one or more input lines.



Where:

n specifies the number of lines to be capitalized. If omitted, 1 is assumed. If .UP n is specified when .UP ON is in effect, capitalization is turned off when n lines have been capitalized, or when .UP OFF is encountered.

ON specifies that subsequent text lines are to be capitalized.

OFF terminates capitalization mode if it was ON, or if n has been specified and has not been exhausted.

line is the line to be capitalized.

Notes:

- Use the .UP control word whenever you have a line of data that is to be formatted in capital letters. If your entire document is to be in capital letters, use the UPCASE option of the SCRIPT/VS command line.
- The .UP control word does not cause an automatic break. Single words in a sentence may be capitalized.
- Another method of capitalizing a single word is to use the uppercase attribute symbol &u' that is recognized by the symbol processor.

Examples:

1. Capitalizing a single word:

```
This sentence has
.up one
capitalized word.
```

   results in:

   This sentence has ONE capitalized word.

2. Capitalizing a single word using the symbol processor's uppercase attribute:

   This sentence has &u'one capitalized word.

   results in:

   This sentence has ONE capitalized word.

3. The .UP control word is a Type 1 control word. The SCRIPT/370 Version 3 .UP control word was not Type 1, but it accepted a single line of text only, like the 'line' form of a Type 1 control word. The SCRIPT/370 control word ".up 80" would process the string "80", but the same control word in SCRIPT/VS capitalizes 80 LINES. If you have any SCRIPT/370 documents that you want to process with SCRIPT/VS, the following macro may be defined to make .UP operate as the SCRIPT/370 control word:

```
.dm up() /.'uc off
.dm up() /.'us off
.dm up() /.'up 1
.dm up() /.li &*
```

.US [UNDERSCORE]

The .US [Underscore] control word automatically underscores one or more input lines.

.US	[1 / n / ON / OFF / line]

Where:

n specifies the number of lines to be underscored. If omitted, 1 is assumed. If .US n is specified when .US ON is in effect, .US is turned off when n lines have been underscored, or when .US OFF is encountered.

ON specifies that subsequent text lines are to be underscored.

OFF terminates underscoring if it was ON, or if n has been specified and has not been exhausted.

line is a single text line to be underscored.

Initial Setting: OFF

Default: 1

Notes:

- Use the .US control word whenever you have a line of data that is to be underscored.
- The .US control word does not cause an automatic break; single words in a sentence may be underscored.

Examples:

1. Underscoring a single word:

```
This sentence has
.us one
underscored word.
```

   results in:

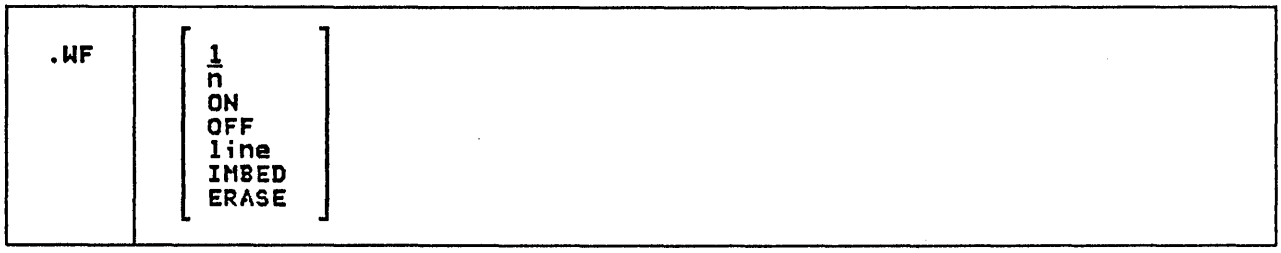
   This sentence has one underscored word.

2. The .US control word is a Type 1 control word. The SCRIPT/370 Version 3 .US control word was not Type 1, but it accepted a single line of text only, like the 'line' form of a Type 1 control word. The SCRIPT/370 control word ".us 80" would underscore the string "80", but the same control word in SCRIPT/VS underscores 80 LINES. If you have any SCRIPT/370 documents that you want to process with SCRIPT/VS, the following macro may be defined to make .US operate as the SCRIPT/370 control word:

```
.dm us() /.'up off
.dm us() /.'uc off
.dm us() /.'us 1
.dm us() /.li &*
```

.WF [WRITE TO FILE]

Use the .WF [Write To File] control word to cause lines of text or control words to be written to an output file with the id DSMUTWTF.



Where:

n specifies that the next n lines are to be written into the DSMUTWTF file.

ON specifies that the following text and control words are to be written into the DSMUTWTF file until .WF OFF is encountered.

OFF stops writing text and control words to the DSMUTWTF file, whether ON was specified, or a number of lines in 'n' that has not yet been exhausted. The .WF OFF control word must occur on a line by itself.

line is a line of text or control words to be written to the file.

IMBED causes the DSMUTWTF file to be imbedded.

ERASE causes the DSMUTWTF file to be erased.

Notes:

- All the text and control words between the .WF ON and OFF control words will be written into the DSMUTWTF file. No .WF control word is written to the file. Any .WF other than .WF OFF is ignored when .WF is writing lines to the file.
- If symbol substitution is ON, the lines that are written to the file will have been substituted. If substitution is ON when the file is imbedded, the lines will be substituted again if any unresolved symbols remained from the first substitution.
- The file-id DSMUTWTF may be associated with different file or data set names using the .DD [Define Data File-id] control word. See the discussion of .DD for more information. Different groups of information can be written to different actual files when .DD is used.
- The .WF control word cannot write into a file that is currently in use for .AP [Append] or .IM [Imbed]. If an imbedded or appended file is ended with the .EF [End of File] control word it is still "in use" unless the CLOSE option was specified.
- The data written to a file will be added after any existing data in the file.

.ZZ [DIAGNOSTIC]

The .ZZ [Diagnostic] control word provides the system programmer with the ability to dump selected internal SCRIPT/VS control blocks. This control word is ignored unless enabled by the DUMP option of the SCRIPT command.

.ZZ	{ ON } { OFF } { DUMP nn [nn ...] }

Where:

ON allows dump data that is specified in message definitions to be printed. This is the initial setting if the DUMP option of the SCRIPT command is specified.

OFF prevents dump data that is specified in message definitions from being printed.

DUMP causes immediate dumping of the data areas indicated by the code numbers given in "nn". The valid range of values is 9 through 64. Any number of values may be specified. The valid code numbers are as follows:

9 Active save areas - DSMSAVD

10 Global area - DSMSGLB

11 Language processor common area - DSMSECT

12 PF work area - DSMNWRK

13 MF work area - DSMMCOM

16 Logical Device table - DSMSLDT

17 TRS work area - DSMTRSC

18 Font work area - DSMSFWA

19 Font tables for logical device - DSMSFTB

20 Trace table

Notes:

- The .ZZ control word may be used several times within the source text to provide selective dump information.
- The information is dumped to the same destination to which error messages are written.
- The control will only be active if the DUMP option was specified on the SCRIPT command.

Figure	Page	Description
21	298	A picture of the SCRIPT/VS page layout.
22	299	List of the utility files that SCRIPT/VS creates or uses.
23	299	Summary of the options of the SCRIPT command.
24	301	Summary of the SCRIPT/VS control words and their parameters.
25	311	List of the control words that result in a break between input lines of text.
26	311	List of the control words that always take effect on the next output page.
27	312	List of the control words that are not allowed within a keep, running heading or footing, or footnote.
28	312	List of the control words that are processed only once within a running heading or footing.
29	313	List of the control words whose initial values are based on the logical default device.
30	313	List of SCRIPT/VS Logical Device Characteristics.
31	314	Summary of the default characteristics of each head-level control word.
32	315	Summary of the parameters that are saved as a result of the .SA [Save Status] control word.
33	316	Summary of SCRIPT/VS system symbols.
34	318	List of attributes of a symbol's value.
35	319	List of the characters that mark the beginning or ending of a word.
36	319	List of the characters that are underscored by default.
37	320	IBM 1403 Printer's TN print train character set.
38	320	List of the fonts provided with SCRIPT/VS for use with the 3800 Printer
39	321	List of the fonts provided with the 3800 Printer.

Figure 20. Index to SCRIPT/VS Summary

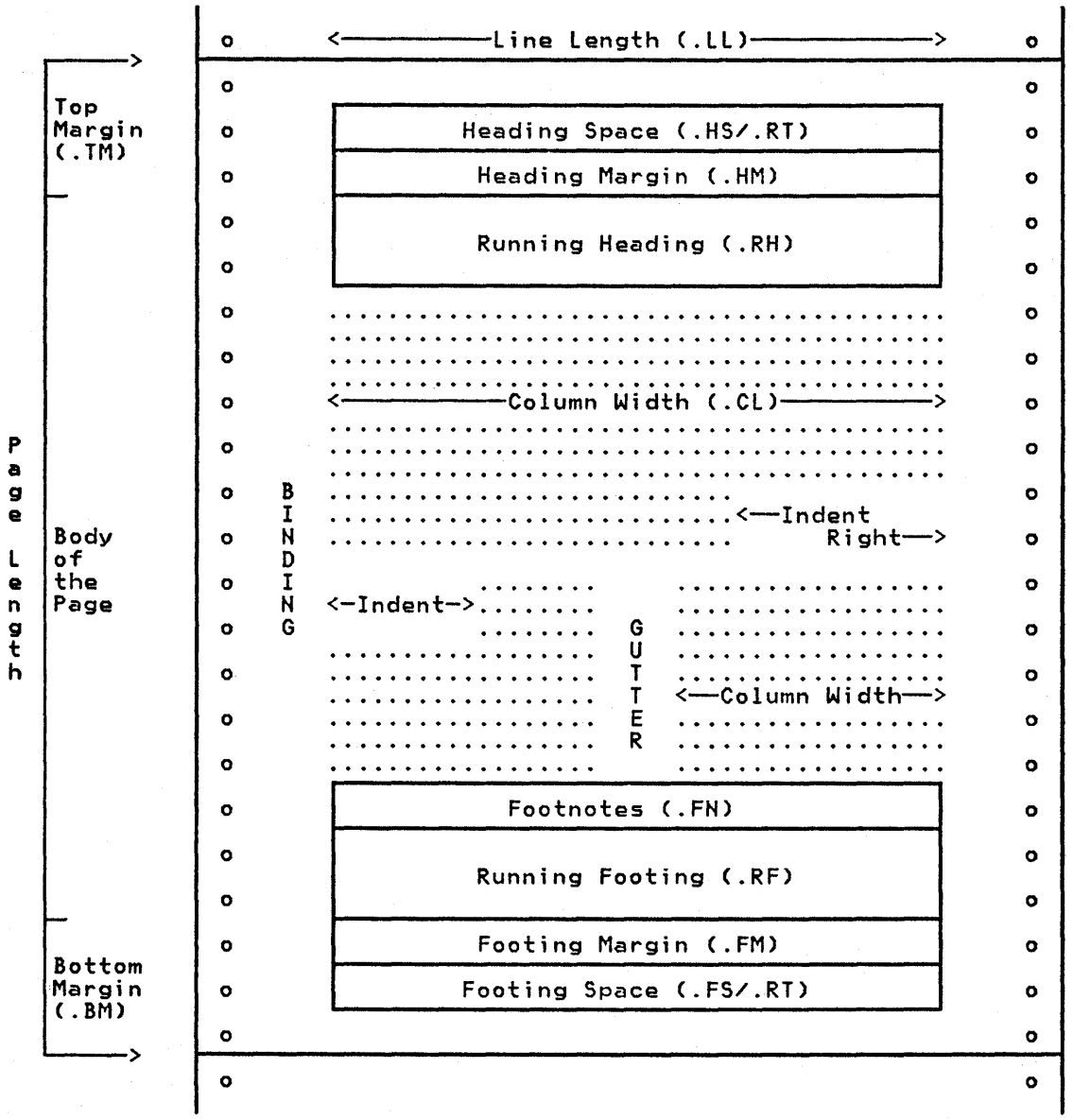


Figure 21. SCRIPT/VS Terms for Parts of the Page: Note that Top Margin and Bottom Margin include all the space on the paper that is accessible to SCRIPT/VS. For terminals and 1403-type printers, this includes the entire page. For 3800-type devices, Top and Bottom Margin do not include 1/2 inch on each side of the interpage perforation. This space is reserved by the 3800 Printer for accelerating and decelerating the paper when it is necessary to halt the paper path.

SCRIPT/VS Utility Files			
File-id	Description	Control Word	Option
DSMTERMI DSMTERMO DSMUTDIM DSMUTMSG DSMUTTOC DSMUTWTF	Terminal input file Terminal output file Delay imbed file Error messages file Table of Contents file Write to file file	.TE .TY .DI .MG .TC, .PT .WF	TERM MESSAGE(DELAY)

Figure 22. File-id's of SCRIPT/VS Utility files: SCRIPT/VS uses or creates these files as a result of the control words or options indicated. Any of these files may be redefined with the .DD [Define Data File-id] control word.

Option	Parameters	Description
BIND	(bind) (obind ebind)	Shift the page image to the right.
CHARS	(font1 ... font4)	Specify up to four fonts.
CONTINUE		Continue processing after a nonsevere error occurs.
DEST	(station id)	Specify a remote output station. (Valid only for TSO.)
DEVICE	(devtype)	Specify a logical output device.
DUMP		Enables the .ZZ [Diagnostic] control word.
FILE	[(fileid)]	Specify a disk file for output.
LIB	(libname ...)	Specify symbol and macro libraries. (Only one for TSO; up to eight for CMS.)
MESSAGE	([DELAY] [ID] [TRACE])	Control message printing.
NOPROF		Suppress the profile.
NOSPIE		Prevent entering SPIE exit routines. (Valid only for CMS and TSO.)
NOWAIT		Prevent prompting for paper adjustment. (Valid only for typewriter terminal output.)
NUMBER		Print file name and line number.
OPTIONS	[(fileid)]	Specify a file that contains SCRIPT options (Valid only for CMS.)

Figure 23. Summary of SCRIPT Options (Part 1 of 2)

Option	Parameters	Description
PAGE	[FROM] p [TO] q [FROM] p FOR n [FROM] p ONLY PROMPT	Selectively print pages.
PRINT	[(copies,class, fcb,ucs)]	Produce printer output. (Sub-options valid only for TSO.)
PROFILE	[(fileid)]	Specify a profile. (A file to be imbedded before the primary input file is processed.)
QUIET		Suppress the formatter's identifier message.
SEARCH	(libname)	Specify a library. (Not valid in a CMS environment.)
SPELLCHK		Enable the .SV [Spelling Verification] control word.
STOP		Print separate pages at the terminal. (Valid only for typewriter-terminal output.)
SYSVAR	(n value ...)	Set symbol values for &SYSVARn.
TERM		Display the output at a user's terminal. (Valid only in CMS and TSO.)
TWOPASS		Prepare with two formatting passes, and produce output on the second pass.
UNFORMAT		Print all input lines without formatting.
UPCASE		Fold lowercase letters to uppercase before printing.

Figure 23. Summary of SCRIPT Options (Part 2 of 2)

Control Word	Parameters	Description
...	label [input line]	Set Label: Inserts a line that can be used as the target of a .GO control word.
.AP	file-id [token1 ... token14]	Append: Allows an additional file to to be appended to the file just processed
.BC	[ON|OFF]	Balance Columns: Causes SCRIPT/VS to attempt to balance the columns when a page eject occurs or when the column definition is changed.
.BF	font	Begin Font: Causes SCRIPT/VS to use a new font. Based on logical device.
.BM	[v|+v|-v]	Bottom Margin: Specifies the amount of space in the bottom margin area. Causes break. SCRIPT/VS special symbol: &$BM Not allowed within a keep, running heading, or running footing. Default: Based on logical device type
.BR		Break: Prevents the concatenation of the following text line with preceding text.
.BT	[n] /left/center/right/	Bottom Title: Sets a bottom title line. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] control word instead.) Not allowed within a keep, running heading, or running footing.
.BX	[NEW|OFF|CAN] [d1 [/] d2 ...] [CHAR name]	Box: Draws horizontal and vertical lines around subsequent output text. Causes a break.
.CB		Column Begin: Causes an eject to the next column (or next page.) Causes a break. Not allowed within a keep, running heading, or running footing.
.CC	[v]	Conditional Column Begin: Causes a column eject if less than a specified amount of space remains in the column. Causes a break. Not allowed within a keep, running heading, or running footing. .CC causes a column eject unless there is no data in the current column.
.CD	n [p1 p2 ... p9]	Column Definition: Specifies the number of columns on the page and position of each column. Causes a break. Not allowed within a keep, running heading, or running footing. Initial value: One column at position 0.
.CE	[1|n|ON|OFF| input-line]	Center: Centers text lines between the current left and right margins. Causes a break.

Figure 24. SCRIPT/VS Control Word Summary (Part 1 of 10)

Control Word	Parameters	Description
.CL	[h|+h|-h]	Column Width: Specifies the width of each column (all columns are the same width). Causes a break. SCRIPT/VS special symbol: &$CL Default: Line length
.CM		Comment: Identifies a comment line.
.CO	[ON|OFF]	Concatenate Mode: Causes output lines to be formed by concatenating input lines. Causes a break.
.CP	[v]	Conditional Page Eject: Causes a page eject if less than a specified amount of space remains on the page. Causes a break. Not allowed within a keep, running heading, or running footing. Default: .CP causes a page eject unless there is no data on the current page.
.CS	n [ON|OFF] n [INCLUDE|IGNORE]	Conditional Section: Allows conditional inclusion of input in the formatted output. Initial value: All conditional sections included.
.CW	[character]	Control Word Separator: Defines the control word separator character. SCRIPT/VS special character: &$CW Initial value: ; (semi-colon) Default: no control word separator.
.DC	[option char ... | OFF]	Define Character: Defines the characters for special functions. Options: ASEP: Array element separator characters (up to 4) CONT: Line continuation character (&$CONT) CW: Control word separator (&$CW) GML: GML tag delimiter (&$GML) STOP: End-of-sentence characters PUNC: Punctuation characters PS: Page number symbol (&$PS) RB: Required blank (&$RB) WORD: [See Figure 35 on page 319] Defaults: ASEP: , 40 GML: : PS: & CONT: none STOP: . ! ? RB: 41 CW: ; PUNC: - '
.DD	file-id [LIB|DD|DSN file-id] [PROC name] [SEQ col length]	Define Data File-id: Specifies the file-id of a file to be used with the .IM [Imbed], .AP [Append] or .WF [Write To File] control words. Default: .DD file-id LIB file-id
.DH	n [options]	Define Head Level: Defines the format and characteristics of the section headings produced by the .Hn control words. Default: Restores initial settings.
.DI	[1|n|ON|OFF| input-line]	Delay Imbed: Delays the processing of input lines until the next page eject occurs. Causes a break. Not allowed within a keep, running heading, or running footing.

Figure 24. SCRIPT/VS Control Word Summary (Part 2 of 10)

Control Word	Parameters	Description
.DM	name (n) / [x|OFF|input-line] name [x|OFF|LIB|	Define Macro: Defines a macro using text, SCRIPT/VS control words, and special symbols.
.DS		Doublespace Mode: Causes subsequent output lines to be doublespaced.
.DU	ADD|DEL word ...	Dictionary Update: Adds words to or deletes words from the addenda dictionary, which is used to supplement the SCRIPT/VS main dictionary for spelling verification and hyphenation.
.EB	[n] /left/center/right/	Even-Page Bottom Title: Sets bottom title lines for even-numbered pages. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] control word instead.) Not allowed within a keep, running heading, or running footing.
.EC	input line	Execute Control: Execute the input line as a control word even if there is a macro of the same name.
.EF	[CLOSE]	End of File: Simulates an end of file condition.
.EM	input line	Execute Macro: Execute the input line as a macro even if macro substitution is off.
.EP	[ON|OFF]	Even-Page Eject: Causes a page eject to the next even-numbered page. (Provided for compatibility with SCRIPT/370; use the .PA [Page Eject] control word instead.) Causes a break. Not allowed within a keep, running heading, or running footing.
.ET	[n] /left/center/right/	Even-Page Top Title: Sets a top title line on even-numbered pages only. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] instead.) Not allowed within a keep, running heading, or running footing.
.EZ	ON|OFF|tag	EasySCRIPT: Enables or disables the EasySCRIPT processing functions.
.FM	[v|+v|-v]	Footing Margin: Specifies the amount of space between the last line of text in the page's body and the first bottom title line. SCRIPT/VS special symbol: &$FM Not allowed within a keep, running heading, or running footing. Default: Based on logical device type.
.FN	ON|OFF|LEADER	Footnote: Saves formatted text and prints it at the bottom of the page in single-column format. Not allowed within a keep, running heading, or running footing.

Figure 24. SCRIPT/VS Control Word Summary (Part 3 of 10)

Control Word	Parameters	Description
.FO	[ON|OFF| LEFT|RIGHT|CENTER] [EXTEND|TRUNC|FOLD]	Format Mode: Controls concatenation and justification of input lines. Default (for .FO OFF): EXTEND
.FS	[n|+n|-n]	Footing Space: Specifies the number of lines in the bottom margin area that can contain bottom title lines. SCRIPT/VS special symbol: &$FS Not allowed within a keep, running heading, or running footing. Default: Based on logical device type.
.FT	line	Footing: Specifies a bottom title. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] control word instead.) Not allowed within a keep, running heading, or running footing.
.GO	label	Go To: Causes SCRIPT/VS to locate the input line identified with "label" and resume processing with that input line.
.HE	line	Heading: Specifies a top title. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] control word instead.) Not allowed within a keep, running heading, or running footing.
.HM	[v|+v|-v]	Heading Margin: Specifies the amount of space between the top title lines and the first line of text (or running heading) on the body of the page. SCRIPT/VS special symbol: &$HM Not allowed within a keep, running heading, or running footing. Default: Based on logical device type.
.HN	ON|OFF|CANCEL	Headnote: Specifies a headnote similiar to a running heading. (Provided for compatibility with SCRIPT/370; use the .RH [Running Heading] control word for text at the top of a page instead.)
.HS	[n|+n|-n]	Heading Space: Specifies the number of lines in the top margin area that contain top title lines. SCRIPT/VS special symbol: &$HS Not allowed within a keep, running heading, or running footing. Default: Based on logical device type.
.HW	text-word	Hyphenate Word: Specifies hyphenation points for a word that might need to be hyphenated during formatting.
.HY	[ON|OFF|SUP|NOADD] [SET MINPT n] [SET THRESH n]	Hyphenate: Controls the SCRIPT/VS automatic hyphenation function. Initial setting: OFF, MINPT=4, THRESH=7
.Hn	[text-line]	Head Level n: Formats a section heading according to default characteristics supplied for the heading.

Figure 24. SCRIPT/VS Control Word Summary (Part 4 of 10)

Control Word	Parameters	Description
.IF	x test y input-line "test" can be: lt le eq ne gt ge < <= = ¬= > >=	If: Tests the relationship between "x" and "y". When the test is satisfied, SCRIPT/VS processes the "input-line." Otherwise, SCRIPT/VS ignores the "input-line." "x test y" can be: SYSPAGE eq|ne EVEN|ODD SYSOUT eq|ne PRINT|TERM
.IL	[0|h|+h|-h]	Indent Line: Indents the next output line the specified amount of horizontal space. Causes a break.
.IM	file-id [token1 ... token14]	Imbed: Processes the named file at this point.
.IN	[0|h|+h|-h]	Indent: Specifies the amount of space subsequent output lines are to be indented from the current left margin. Causes a break. SCRIPT/VS special symbol: &$IN
.IR	[0|h|+h|-h]	Indent Right: Specifies the amount of space subsequent input lines are to be indented from the current right margin. Causes a break. SCRIPT/VS special symbol: &$IR
.IT	[ON|OFF|ALL|MAC|SUB CTL|SNAP|STEP|RUN] Response to STEP: (null) PRE input-line REP input-line STK input-line	Input Substitution Trace: Provides a trace of processing for each SCRIPT/VS control word and macro, as well as symbol substitution. When .IT STEP is in effect, the user responds interactively. Initial value: No input tracing.
.JU	[ON|OFF]	Justify Mode: Causes left and right justification of output lines as needed to make the end of each line even with the current right margin. Causes a break.
.KP	[ON|FLOAT|DELAY| INLINE|v|v + v|OFF]	Keep: Ensures that a group of output lines are kept together in the same column. SCRIPT/VS special symbol: &$KP
.LB		Leading Blank: Is processed whenever an input line with a blank as the first character is encountered. Causes a break.
.LI	[1|n|ON|OFF| input-line]	Literal: Ensures that input lines are treated as text lines by SCRIPT/VS (used when a text input line begins with a period).
.LL	[h|+h|-h]	Line Length: Specifies the length of each subsequent output line. SCRIPT/VS special symbol: &$LL Not allowed within a keep, running heading, or running footing. Default: Based on logical device type.

Figure 24. SCRIPT/VS Control Word Summary (Part 5 of 10)

Control Word	Parameters	Description
.LS	n	Line Spacing: Specifies the number of blank lines between each subsequent output line. (Provided for compatibility with SCRIPT/370; use the .SL [Set Line Space] control word instead.)
.LT		Leading Tab: Is processed whenever an input line with a tab as the first character is encountered. Causes a break.
.LY	[ON|OFF|SYM|MAC]	Library: Specifies whether a library is to be used to resolve symbol and macro definitions. Use the LIB option to identify the libraries.
.MC		Multicolumn Mode: Restores column definition saved by a previous .SC [Single Column Mode] control word. Causes a break. Not allowed within a keep, running heading, or running footing.
.MG	/[id]/text/	Message: Produces a message similar in format to the SCRIPT/VS error messages.
.MS	ON|OFF	Macro Substitution: Causes SCRIPT/VS to recognize and process macros. Initial value: OFF
.NL		Null Line: Is processed whenever an input line that contains no characters is encountered.
.OB	[n] /left/center/right/	Odd Page Bottom Title: Sets bottom title lines for odd-numbered pages. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] control word instead.) Not allowed within a keep, running heading, or running footing.
.OC	input-line	Output Comment: Specifies a line that is to be inserted into the output document as it is, as an output comment.
.OF	[0|h|+h|-h]	Offset: Causes a hanging indention (a paragraph in which the indention of the first line is unchanged and subsequent lines are indented to the offset value.) Causes a break. SCRIPT/VS special symbol: &$OF
.OP	[ON|OFF]	Odd-Page Eject: Causes a page eject to the next odd-numbered page. (Provided for compatibility with SCRIPT/370; use the .PA [Page Eject] control word instead.) Not allowed within a keep, running heading, or running footing.

Figure 24. SCRIPT/VS Control Word Summary (Part 6 of 10)

Control Word	Parameters	Description
.OT	[n] /left/center/right/	Odd-Page Top Title: Sets a top title line for subsequent odd-numbered pages. (Provided for compatibility with SCRIPT/370; use the .RT [Running Title] control word instead.) Not allowed within a keep, running heading, or running footing.
.PA	[ODD|EVEN] [ON|OFF] [+0|n|+n|-n] [NOSTART]	Page Eject: Causes a page eject, and can set the page number of the new page. Not allowed within a keep, running heading, or running footing.
.PF		Previous Font: Causes the last stacked font to become the current font.
.PL	[v|+v|-v]	Page Length: Specifies the amount of space, including top and bottom margins, for each output page. SCRIPT/VS special symbol: &$PL Not allowed within a keep, running heading, or running footing. Default: Based on logical output device
.PN	[n|ON|OFF|OFFNO| ARABIC|ROMAN|ALPHA| NORM|FRAC| PREF string]	Page Numbering Mode: Controls external and internal page numbering. Not allowed within a keep, running heading, or running footing. Initial value: Arabic numbers from 1.
.PP	[input-line]	Paragraph Start: Begins formatting the output line as the start of a paragraph after a skip.
.PS	character	Page Number Symbol: Sets a page number symbol. SCRIPT/VS special symbol: &$PS Initial value: & (ampersand)
.PT	input-line	Put Table of Contents: Places the input line (which may be a control word, macro, GML tag, symbol, or line of text) into the file used to accumulate table of contents entries (DSMUTTOC).
.QQ		Quick Quit: Causes SCRIPT/VS processing to terminate immediately without completing the current page.
.QU		Quit: Causes SCRIPT/VS processing to terminate after completing the current page.
.RC	n s n [ON|OFF|ON/OFF] * s	Revision Code: Specifies a revision code symbol that is to be printed to the left of the output line that contains updated material.
.RD	[1|n]	Read Terminal: Allows user to type in one or more text lines while a file is being formatted. Causes a break. Not allowed within a keep, running heading, or running footing.

Figure 24. SCRIPT/VS Control Word Summary (Part 7 of 10)

Control Word	Parameters	Description
.RE		Restore Status: Restores environment that has been previously saved with the .SA [Save Status] control word.
.RF	[ON|OFF|CANCEL] [ODD|EVEN]	Running Footing: Specifies input lines that are to be saved as a running footing and processed at the bottom of each appropriate page. Initial value: No running footing.
.RH	[ON|OFF|CANCEL] [ODD|EVEN]	Running Heading: Specifies input lines that are to be saved as a running heading and processed at the top of each appropriate page. Initial value: No running heading.
.RI	[1|n|ON|OFF| input-line]	Right Adjust: Produces output lines that are unconcatenated input lines aligned with the right-hand margin. Causes a break.
.RT	[TOP|BOTTOM] [ALL|ODD|EVEN] [1|n] /left/center/right/	Running Title: Defines running title lines for the top and bottom of even, odd, or all output pages. Not allowed within a keep, running heading, or running footing. Initial value: .RT TOP ALL 1 ///PAGE &/ Default: .RT TOP ALL 1 /left///
.RV	symbolname [']	Read Variable: Allows user to assign a value to a symbolname by entering it at the terminal in response to an interactive request made while SCRIPT/VS is processing the input file.
.SA		Save Status: Saves the current values and parameters of the formatting environment.
.SC		Single-Column Mode: Causes SCRIPT/VS to save the current column definition and format subsequent input lines in a single column. Causes a break. Not allowed within a keep, running heading, or running footing.
.SE	symname[(n)] [LIB|OFF] [= value] [= SUBSTR str n1 n2] [= INDEX str1 str2]	Set Symbol: Defines a symbol name and assigns a value to it.
.SF		Save Font: Saves the current font-id.
.SK	[1|v] [A] [C] [P]	Skip Lines: Specifies the amount of space to insert before the next text output line. No lines are inserted if the .SK occurs at the top of a page or column. Causes a break.
.SL	[v]	Set Line Spacing: Specifies the vertical distance between baselines of output lines. Default: Based on logical output device.

Figure 24. SCRIPT/VS Control Word Summary (Part 8 of 10)

Control Word	Parameters	Description
.SP	[1|v] [A] [C] [P]	Space Lines: Specifies the amount of space to insert before the next text output line. The specified number of lines are inserted even when the .SP occurs at the top of a page or column. Causes a break.
.SS		Single-Space Mode: Causes subsequent output lines to be single-spaced.
.SU	[1|n|ON|OFF| input-line]	Substitute Symbol: Controls the substitution of symbols with their previously assigned values. SCRIPT/VS special symbol: &$SU Initial value: ON
.SV	[ON|OFF] [NOADD] [NOSTEM] [NUM]	Spelling Verification: Defines the start and functions of the SCRIPT/VS spelling verification function. Enabled with the SPELLCHK option. Initial value: OFF
.SX	[F] /left/fill/right/	Split Text: Produces an output line of three parts: "left" is aligned with the current left margin; "right" is aligned with the current right margin; "fill" is characters that fill the remaining space between the two strings. Causes a break.
.SY	input-line	System Command: SCRIPT/VS passes the input line to the host system for processing. SCRIPT/VS special symbol: &$RET (return code)
.TB	[h h h ...] [f/h f/h ...]	Tab Setting: Specifies the tab settings to be used when the input file is formatted. Causes a break. Default: 5 10 15 20 ... 75
.TC	n [name|/]	Table of Contents: Imbeds the table contents file (DSMUTTOC), which consists of table of contents entries automatically generated by the .Hn control words, and entries inserted by using the .PT [Put Table of Contents] control word. Use the TWOPASS option if the table of contents is not at the back of the document. Causes a break. Not allowed within a keep, running heading, or running footing.
.TE	[1|n|ON|OFF]	Terminal Input: Allows user to enter lines interactively from the terminal when the file is formatted.
.TI	[s t ...]	Translate Input: Specifies character translations to be performed on input lines before SCRIPT/VS processing begins. Default: Identity

Figure 24. SCRIPT/VS Control Word Summary (Part 9 of 10)

Control Word	Parameters	Description
.TM	[v|+v|-v]	Top Margin: Specifies the amount of space in the top margin area. SCRIPT/VS special symbol: &$TM Not allowed within a keep, running heading, or running footing. Default: Based on logical device type.
.TR	[s t ...]	Translate Character: Specifies character translations to be performed on output. Default: Identity
.TT	[n] /left/center/right/	Top Title: Sets a top title line for subsequent pages. (Provided for compatibility with SCRIPT/370; use the .RD [Read Terminal] control word instead.) Not allowed within a keep, running heading, or running footing.
.TY	input-line	Type On Terminal: Types the input line on the user's terminal during formatting.
.UC	[1|n|ON|OFF| input-line]	Underscore and Capitalize: UNDERSCORES AND CAPITALIZES one or more subsequent input lines.
.UD	ON|OFF c c ...	Underscore Definition: Defines the characters to be underscored when the .UC and .US control words are used.
.UN	[0|h|+h|-h]	Undent: Causes the next output line's indention to change: it is moved to the left of the current left margin. Causes a break.
.UP	[1|n|ON|OFF| input-line]	Uppercase: Prints one or more subsequent input lines in UPPERCASE characters.
.US	[1|n|ON|OFF| input-line]	Underscore: Prints one or more subsequent input lines with underscored characters.
.WF	[1|n|ON|OFF| IMBED|ERASE| input-line]	Writes one or more input lines to the output file DSMUTWTF. IMBED: imbeds file DSMUTWTF. ERASE: erases file DSMUTWTF.
.ZZ	[ON|OFF] nn ...	Diagnostic: Turns on or off the diagnostic trace function, and selects the type of data to be traced. Enabled with the DUMP option. Initial value: OFF

Figure 24. SCRIPT/VS Control Word Summary (Part 10 of 10)

.BR [Break]
.BX [Box]
.CB [Column Begin]
.CC [Conditional Column Begin] [1]
.CD [Column Definition]
.CE [Center]
.CL [Column Width]
.CO [Concatenate Mode]
.CP [Conditional Page Eject] [1]
.EP [Even Page Eject]
.FI [Fill Mode]
.FO [Format Mode]
.HN [Headnote]
.H1 [Head Level 1]
.H2 [Head Level 2]
.H3 [Head Level 3]
.H4 [Head Level 4]
.H5 [Head Level 5]
.H6 [Head Level 6]
.IL [Indent Line]
.IN [Indent]
.IR [Indent Right]
.JU [Justify Mode]
.LB [Leading Blank]
.LL [Line Length]
.LT [Leading Tab]
.MC [Multicolumn Mode] [1]
.NB [No Balancing]
.NC [No Concatenation]
.NF [No Formatting]
.NJ [No Justification]
.OF [Offset]
.OP [Odd Page Eject]
.PA [Page Eject]
.PP [Paragraph Start]
.QU [Quit]
.RD [Read Terminal]
.RF [Running Footing]
.RH [Running Heading]
.RI [Right Adjust]
.SC [Single Column Mode]
.SK [Skip]
.SP [Space]
.SX [Split Text]
.TB [Tab Setting]
.TC [Table of Contents]
.UN [Undent]

[1] The break occurs only if the control word performs its function. These control words may do nothing if the function is not needed.

Figure 25. Control Words That Cause a Break: When Concatenation is on (see the .CO [Concatenate Mode] and .FO [Format Mode] control words), words from input lines are re-arranged on output lines to make each column line as full as possible. This process is inhibited for the current line if any of these control words is encountered.

.BM [Bottom Margin]
.FM [Footing Margin]
.FS [Footing Space]
.H1 [Head Level 1] [1]
.HM [Heading Margin]
.HS [Heading Space]
.LL [Line Length]
.PL [Page Length]
.PN [Page Numbering Mode]
.RF [Running Footing]
.RH [Running Heading]
.RT [Running Title]
.TM [Top Margin]

[1] .H1 causes a page eject by default. The .DH [Define Head Level] control word allows you to redefine the meaning of the .H1 control word.

Figure 26. Control Words That Take Effect On the Next Page: These control words take effect on the next output page to be started. If no data has yet been placed on the first page of the document, or the previous page was ended with a .PA NOSTART control word, the first, or next, page has not yet been started, and these control words can take effect on this page.

.BM [Bottom Margin]
.BT [Bottom Title]
.CB [Column Begin]
.CC [Conditional Column Begin]
.CD [Column Definition]
.CP [Conditional Page Eject]
.DI [Delay Imbed]
.EB [Even Page Bottom Title]
.EP [Even Page Eject]
.ET [Even Page Top Title]
.FM [Footing Margin]
.FS [Footing Space]
.FT [Footing]
.HE [Heading]
.HM [Heading Margin]
.HN [Headnote]
.HS [Heading Space]
.LL [Line Length]
.MC [Multicolumn Mode]
.OB [Odd Page Bottom Title]
.OP [Odd Page Eject]
.OT [Odd Page Top Title]
.PA [Page Eject]
.PL [Page Length]
.PN [Page Numbering Mode]
.RD [Read Terminal]
.RT [Running Title]
.SC [Single Column Mode]
.TC [Table of Contents]
.TM [Top Margin]
.TT [Top Title]

Figure 27. Control Words That End a Keep, Running Heading or Footing, or Footnote: If found, a message is issued and the Keep, Heading or Footing, or Footnote is terminated before the control word is processed.

Note: .RF and .RH are disallowed in keeps and footnotes. .KP and .FN are disallowed in running headings and footings.

.AP [Append]
.CM [Comment]
.CS [Conditional Section]
.CW [Control Word Separator]
.DC [Define Character]
.DD [Define Data File-id]
.DM [Define Macro]
.DU [Dictionary Update]
.EF [End of File]
.GO [Goto]
.Hn [Head Level n]
.IF [If]
.IM [Imbed]
.IT [Input Trace]
.LI [Literal]
.LY [Library]
.MS [Macro Substitution]
.OC [Output Comment]
.PN [Page Numbering Mode]
.PP [Paragraph Start]
.PT [Put Table of Contents]
.QQ [Quick Quit]
.QU [Quit]
.RF [Running Footing]
.RH [Running Heading]
.RV [Read Variable]
.SE [Set Symbol]
.SU [Substitute Symbol]
.SV [Spelling Verification]
.SY [System Command]
.TE [Terminal Input]
.TY [Type on Terminal]
.UC [Underscore and Capitalize]
.UD [Underscore Definition]
.UP [Uppercase]
.US [Underscore]
.WF [Write To File]

Figure 28. Control Words Within a Running Heading or Footing: These control words are processed only once, during a Running Heading or Footing definition. All other control words are saved as part of the Heading or Footing definition, and processed each time a new page is formatted.

.BF [Begin Font] [1]
.BM [Bottom Margin]
.FM [Footing Margin]
.FS [Footing Space]
.HM [Heading Margin]
.HS [Heading Space]
.LL [Line Length]
.PL [Page Length] [2]
.TM [Top Margin]

Device Class	Initial Values						
	.TM	.HS	.HM	.BM	.FS	.FM	.LL
Terminal	6	1	2	6	1	2	6i
1403	6	1	2	6	1	2	6i
3800	3	1	2	3	1	2	6i

[1] .BF is applicable only to 3800-type logical devices, and is determined by the CHARS option if specified; otherwise, by the DEVICE option.

[2] For 3800-type devices, Page Length does not include 1/2 inch at the top and bottom of each page. This area is inaccessible to the formatter.

Figure 29. Control Word Values Based On the Logical Device: The initial and default values for these control words vary, depending upon the specified or implied logical output device.

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

[1] Line lengths are given as "default/maximum" in 10-pitch characters. For the 3800 Printer, 12-pitch and 15-pitch fonts have values 20% and 50% greater, respectively.

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

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

Figure 30. SCRIPT/VS Logical Device Characteristics

.Hn Control Word							
keys	H0	H1	H2	H3	H4	H5	H6
SKBF	0	0	3	3	3	1	1
SPAF	0	5	2	2	2	0	0
TCIN	0	0	0	2	4	6	8
TO	X						
TC	X	X	X	X			
TS		X					
US		X	X		X	X	X
UP		X	X	X		X	
OJ		X					
PA		X					
BR		X	X	X	X		

EasySCRIPT Head Levels							
keys	H0	H1	H2	H3	H4	H5	H6
SKBF	0	0	3	3	3	3	3
SPAF	0	5	3	3	3	0	0
TCIN	0	0	0	2	4	6	8
TO	X						
TC	X	X	X	X			
TS		X					
US		X	X		X		X
UP		X	X	X		X	
OJ		X					
PA		X					
BR		X	X	X	X		

where the "keys" are:

SKBF: number of line skips before the head.
SPAF: number of line spaces after the head.
TCIN: amount of indentation for table of contents entry.
TO: table of contents entry only; no heading in text.
TC: table of contents entry.
TS: line space before table of contents entry.
US: head is underscored.
UP: head is capitalized.
OJ: head is outjustified.
PA: page eject before head.
BR: break after head.

Figure 31. Summary of Head Level Characteristics: This table lists the default characteristics of the .Hn [Head Level n] control words and EasySCRIPT &Hn tags. The .DH [Define Head Level] control word allows you to redefine any of these Head Levels to suit your needs.

Active Environment			
Parameter	Control Word	Initial Setting	Symbol
Column balancing	.BC	ON	
Continuation Character	.DC CONT	(null)	&$CONT
Control Word Separator	.CW, .DC CW	";"	&$CW
Current font	.BF, .PF	(1)	
Column definition	.CD	Single column	
Centering [2]	.CE	OFF	
Column width	.CL	Line length	&$CL
Concatenation	.CO, .FO	ON	
Conditional sections	.CS	INCLUDE	
Line spacing	.SS, .DS, .SL	Single spacing	
Format mode	.FO	ON	
GML tag delimiter	.DC GML	":"	&$GML
Indention [3]		0	&$IN
Justification	.JU, .FO	ON	
Page number symbol	.DC PS, .PS	"&"	
Revision code	.RC	OFF	
Right adjustment	.RI	OFF	
Spelling verification	.SV	OFF	
Tab setting	.TB	5 10 15 ... 80	&$TAB
Terminal input [2]	.TE	OFF	

Page Environment			
Parameter	Control Word	Initial Setting	Symbol
Bottom margin	.BM	(1)	&$BM
Footing margin	.FM	(1)	&$FM
Footing space	.FS	(1)	&$FS
Heading margin	.HM	(1)	&$HM
Heading space	.HS	(1)	&$HS
Hyphenation	.HY	OFF	
Page length	.PL	(1)	&$PL
Page numbering mode	.PN	Arabic	
Macro substitution [2]	.MS	OFF	
Symbol substitution [2]	.SU	ON	&$SU
Top margin	.TM	(1)	&$TM

Translate Tables			
Parameter	Control Word	Initial Setting	Symbol
Input translation	.TI	Identity	
Output translation	.TR	Identity	

[1] These parameters' initial settings are based upon the logical output device.

[2] The number of lines remaining, or ON or OFF, is saved.

[3] The composite current indention is determined from the .IN, .IR, .IL, .UN and .OF control word values. These values are individually saved.

Figure 32. The SCRIPT/VS Formatting Environment: The .SA [Save Status] and .KP [Keep] control words preserve these parameters.

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	2741, 1403, 3800

SCRIPT Command Options		
Symbol	Description	Value
&$BE	Even bind [3] [4]	0-
&$BO	Odd bind [3] [4]	0-
&$CHAR(n)	Fonts [5]	1-4 characters
&$LIB	Macro library available	0, 1
&$PARM	Command options [6]	8-256 characters
&$TWO	TWOPASS option in effect	0, 1 (0 is no, 1 is yes)
&$UNF	Unformatted output	0, 1 (0 is no, 1 is yes)

[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.

[4] The system symbol values are represented in character spaces, regardless of the space units used in setting them. The maximum value depends upon the logical output device.

[5] Set by the CHARS option of the SCRIPT command. This is a symbol array; element 0 contains the number of fonts specified and elements 1, 2, ... contain the names of the fonts specified.

[6] This is the SCRIPT command options list, tokenized into eight character tokens. Note that this list will be truncated at 32 tokens (256 characters).

Figure 33. SCRIPT/VS System Symbol Names (Part 1 of 2)

Page Characteristics		
Symbol	Description	Value
&$BM	Bottom margin (.BM) [7]	0-
&$CL	Column width (.CL) [8]	0-
&$FM	Footing margin (.FM) [7]	0-
&$FS	Footing space (.FS)	0-6
&$HM	Heading margin (.HM) [7]	0-
&$HS	Heading space (.HS)	0-6
&$IN	Left indention [8]	0-
&$LC	Lines left in column [7] [9]	0-
&$LL	Line length (.LL) [8]	0-
&$OF	Offset [8]	0-
&$PL	Page length (.PL) [7]	0-
&$TM	Top margin (.TM) [7]	0-

SCRIPT/VS Formatter Parameters		
Symbol	Description	Value
&$BS	Backspace Character	hexadecimal 16
&$CONT	Continuation character [10]	one character
&$CW	Control word separator [10]	(default: ";")
&$C256	Identity vector	256 characters
&$FNAM	Current input file name	eight characters
&$GML	GML tag delimiter [10]	(default: ":")
&$KP	Keep in effect	ON, OFF
&$LNUM	Last line number read	0-
&$PS	Page number symbol [10]	(default: "&")
&$RB	Required Blank [10]	(default: hexadecimal 41)
&$RET	Return code from .SY [11]	0-
&$SU	Symbol substitution enabled	ON, OFF
&$TAB	Tab Character	hexadecimal 05

[7] 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.

[8] 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.

[9] Note that the value of &$LC does **not** include &$BM, as it does in SCRIPT/370.

[10] Set by the .DC [Define Character] control word.

[11] In CMS, any possible return code value. In TSO, "0" to indicate the command was stacked for execution after SCRIPT/VS terminates. In batch, "-3" to indicate that the .SY [System Command] control word is not supported.

Figure 33. SCRIPT/VS System Symbol Names (Part 2 of 2)

Attribute	Function
&a'	Converts a numeric character string[1] to a "base-26" lowercase alphabetic "number."
&A'	Converts a numeric character string[1] to a "base-26" uppercase alphabetic "number."
&E'	Verifies the existence of a symbol; the value is 1 if the symbol has been set; 0 if not.
&L'	Yields the length of a character string[1].
&r'	Converts a numeric character string[1] into a lowercase Roman numeral.
&R'	Converts a numeric character string[1] into an uppercase Roman numeral.
&U'	Converts a lowercase character string to uppercase.
&V'	Yields the current value of a symbol.

[1] The character string may be the value of a symbol.

Figure 34. Attributes of a Symbol's Value

Hexadecimal Code	Character	Hexadecimal Code	Character	Hexadecimal Code	Character
05	Tab	4E	+	6C	%
11	Special Blank [1]	4F	|	6D	_
12	Special Blank [1]	5A	:	6F	?
13	Special Blank [1]	5B	$	7A	:
16	Backspace	5C	*	7E	=
40	Blank	5D	)	7F	"
41	Required Blank [2]	5E	;	8B	{
4B	. (Period)	5F		9B	}
4C	<	61	/	AD	[
4D	(	6B	,	BD	]

[1] Special Blanks are used for justification in documents formatted for the 3800 Printer.

[2] The required blank is a blank which cannot have space added to it during justification. Its value may be changed with the .DC [Define Character] control word.

Figure 35. Characters that Delimit Words for Spelling Verification: These default characters can be changed with the .DC [Define Character] control word, which accepts either single characters or two-digit hexadecimal character codes.

Hexadecimal Code	Character	Hexadecimal Code	Character	Hexadecimal Code	Character
05	Tab	5A	:	7A	:
16	Backspace	5D	)	7F	"
40	Blank	5E	;	8B	{
41	Required Blank [1]	6B	,	9B	}
4B	. (Period)	6D	_	AD	[
4D	(	6F	?	BD	]

[1] The required blank is a blank which cannot have space added to it during justification. Its value can be changed with the .DC [Define Character] control word.

Figure 36. Characters Not Underscored By Default: The .UD [Underscore Definition] control word can be used to change these defaults, and accepts either single characters or two-digit hexadecimal character codes.

Hexadecimal character code in the form X'ab':

	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	aA	aB	aC	aD	aE	aF	
0b																	0b
1b																	1b
2b																	2b
3b																	3b
4b											¢	.	<	(	+	|	4b
5b	&										!	$	*	)	;	¬	5b
6b	-	/										,	%	_	>	?	6b
7b											:	#	@	'	=	"	7b
8b		a	b	c	d	e	f	g	h	i		{	≤	(	+	±	8b
9b		j	k	l	m	n	o	p	q	r		}	■	)	±	▪	9b
Ab	~	°	s	t	u	v	w	x	y	z		↳	↱	[	≥	•	Ab
Bb	0	1	2	3	4	5	6	7	8	9		⌟	¬	]	≠	—	Bb
Cb		A	B	C	D	E	F	G	H	I							Cb
Db		J	K	L	M	N	O	P	Q	R							Db
Eb			S	T	U	V	W	X	Y	Z							Eb
Fb	0	1	2	3	4	5	6	7	8	9							Fb
	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9	aA	aB	aC	aD	aE	aF	

Tab = X'05'
Backspace = X'16'

Figure 37. TN Translate Table For the 1403 Printer

Text Fonts
GT10 Gothic (10-pitch) GT12 Gothic (12-pitch) GT15 Gothic (15-pitch) ST10 Serif (10-pitch) ST12 Serif (12-pitch) ST15 Serif (15-pitch)

Highlight Fonts
GB10 Gothic Bold GB12 Gothic Bold GI12 Gothic Italic SI10 Serif Italic SI12 Serif Italic SB12 Serif Bold

Special Fonts
GR10 Gothic Reverse GP12 Proportional RT10 Roman Text SO12 Serif Overstruck

Figure 38. Fonts Provided With SCRIPT/VS: Each font is a complete set of upper- and lower-case characters. Any two of these fonts may be specified with the CHARS option of the SCRIPT command.

10-pitch Fonts	
GS10	Gothic
GF10	Gothic Folded
GU10	Gothic Underscored
TU10	Text Underscored [1]

12-pitch Fonts	
GS12	Gothic
GF12	Gothic Folded
GU12	Gothic Underscored

15-pitch Fonts	
GS15	Gothic
GSC	Gothic Condensed
GF13	Gothic Folded
GFC	Gothic Folded Condensed
GU15	Gothic Underscored
GUC	Gothic Underscored Condensed
DUMP	Condensed DUMP [2]

Format Fonts	
FM10	Format 10-pitch
FM12	Format 12-pitch
FM15	Format 15-pitch

10-pitch Katakana Fonts	
2773	Gothic and Katakana
2774	Gothic and Katakana [2]
KN1	Gothic and Katakana [2]

3211 Print Trains	
A11	Gothic 10-pitch
G11	Gothic 10-pitch
H11	Gothic 10-pitch
P11	Gothic 10-pitch
T11	Text 10-pitch [1]

1403 Print Trains	
AN	Gothic 10-pitch
GN	Gothic 10-pitch
HN	Gothic 10-pitch
PCAN	Gothic 10-pitch
PCHN	Gothic 10-pitch
PN	Gothic 10-pitch
QN	Gothic 10-pitch
QNC	Gothic 10-pitch
RN	Gothic 10-pitch
XN	Gothic 10-pitch
YN	Gothic 10-pitch
SN	Text 10-pitch [1]
TN	Text 10-pitch [1]

10-pitch OCR Fonts	
AOA	Gothic and OCR-A
AOD	Gothic and OCR-A
AON	Gothic and OCR-A
OAA	Gothic and OCR-A
ODA	Gothic and OCR-A
ONA	Gothic and OCR-A
BOA	Gothic and OCR-B
BON	Gothic and OCR-B
OAB	OCR-B
ONB	Gothic and OCR-B

[1] This is an upper- and lowercase font which closely resembles the ST10 SCRIPT/VS font. It counts as two fonts when combined with other fonts in the CHARS option of the SCRIPT command.

[2] This font contains more than 64 characters. It counts as two fonts when combined with other fonts in the CHARS option of the SCRIPT command.

Figure 39. Fonts Supplied With the 3800 Printer: These are all uppercase-only fonts, unless otherwise marked. Any four fonts (except those otherwise marked) of identical pitch may be specified with the CHARS option of the SCRIPT command.

SCRIPT/VS bases its formatting of a document on the characteristics of a specified (or implied) logical output device. SCRIPT/VS takes into account the characteristics of the physical device, as well as dynamically changing characteristics: font, lines per inch, and form size.

The combination of these fixed (physical) characteristics and changeable characteristics is called the logical output device, which corresponds to the "setup" of a physical output device.

The module DSMLPLDT contains one logical device table (LDT) for each logical output device. Each LDT is created by a DSMSLDD macro and is mapped by the DSMSLDT DSECT.

UPDATING A LOGICAL DEVICE TABLE (LDT)

Use the following procedure to add or change an LDT entry:

1. Obtain a listing of the DSMSLDD macro and of the DSMLPLDT module from your library.
2. Obtain the source code for the DSMLPLDT module and add a new DSMSLDD macro that describes the new logical output device

   or

   Modify an existing DSMSLDD specification macro that describes the logical output device whose characteristics you want to change. The macro's field names are described in detail in "LDT Field Descriptions."
3. Assemble the DSMLPLDT module to include the changes you've made.
4. Link the newly-assembled version of the DSMLPLDT module with the rest of SCRIPT/VS, as described in the Program Directory.

LDT FIELD DESCRIPTIONS

Each logical device table (LDT) macro is specified as:

DSMSLDD	LD=name,	Logical device name
	PD=device name,	Physical device identifier
	[DF=font name]	Default font identifier
	[,LPI=6|8|12]	Lines per inch
	[,MPL=device units|66]	Maximum page length
	[,DPL=device units|66]	Default page length
	[,MLL=device units|60]	Maximum line length
	[,DLL=device units|60]	Default line length
	[,P=10|12|15]	Pitch
	[,HS=device units|1]	Horizontal space unit
	[,VS=device units|1]	Vertical space unit
	[,DOTxx[14]=value]	Page margin values

Page dimensions (MPL, DPL, MLL, DLL) and space units (HS and VS) are represented as device units: the printer's physical unit of resolution in the relevant direction. Vertically, this is one line for all printers. Horizontally, this is one character space

[14] xx = TM (Top Margin), HS (Heading Space), HM (Heading Margin), BM (Bottom Margin), FS (Footing Space), or FM (Footing Margin). DOTxx can be specified repeatedly (for example: ... DOTTM=3, DOTBM=3).

for all printers except the 3800 Printer. The 3800 Printer's unit of horizontal resolution is the pel, a space of 1/180th of an inch. The HS space unit for the 3800 Printer, therefore, could be 18 (for a 10-pitch horizontal space), 15 (for a 12-pitch space), or 12 (for a 15-pitch space).

The content and meaning of each LDT field that you can specify with the DSMSLDD are shown below.

LDT Field	Description
LD	Logical device name: 1 to 8 alphameric characters.
PD	Physical device name: 4 to 8 alphameric characters. The values recognized are 1403, 2741, 3270, and 3800.
DF	Name of the default font, 1 to 4 alphanumeric characters (required only for logical devices that use the 3800 Printer; not allowed for other output device types). The default font is used when the CHARS option of the SCRIPT command is not specified.
LPI	Lines per inch. A decimal number: 6, 8, or 12. Default = 6.
MPL	The maximum number of vertical device units in the printable portion of the page. Default = 66.
MLL	The maximum number of horizontal device units in the printable portion of the page. Default = 60. MPL and MLL establish the form size for the logical output device (that is, the size of its physical page.)
DPL	The default page length in device units. This value is used until reset with the .PL [Page Length] control word. DPL cannot be greater than MPL. Default = 66.
DLL	The default line width in device units. This value is used until reset with the .LL [Line Length] control word. DLL cannot be greater than MLL. Default = 60. DPL and DLL establish the default page size, which cannot be larger than the form size.
P	Pitch, or number of equal-width characters per inch. For monospaced devices only. A decimal number: 10, 12, or 15. Default = 10.
HS	The number of device units in one "horizontal space unit" (the width of a character in the default font.) This value is used to resolve a control word's parameter that specifies a horizontal space or displacement, but does not specify a unit of measurement (for example, .IN 2).
VS	The number of device units in one "vertical space unit" (that is, the vertical space of a print line). This value is used to resolve a control word's parameter that specifies a vertical space or displacement, but does not specify a unit of measurement (for example, .SP 2). The blank-character codes are used for line justification when the output is in a font that includes pseudo-blanks. All fonts to be used in a mixed-pitch environment must include the proportional-spaced blank character codes in their character arrangement tables.

DOTxx These keywords can be used to override the default page margin parameters:

Keyword	Overrides Default
DOTBM	Bottom margin
DOTTM	Top margin
DOTHM	Heading margin
DOTFM	Footing margin
DOTHS	Heading space
DOTFS	Footing space

Note: For 3800-type logical devices, DOTBM and DOTTM are ignored. Instead,

Bottom margin = Footing margin + Footing space

Top margin = Heading margin + Heading space

DEFAULT VALUES FOR LOGICAL OUTPUT DEVICES

When you specify values for the various fields of the LDT, SCRIPT/VS will use those values to derive the following defaults:

Parameter	Default derived from
Top margin	6 times VS (6 vertical line spaces) or for the 3800 Printer: 3 times VS or 0, if page length is less than 15.
Heading Space	VS (1 vertical line space) or 0, if page length is less than 15.
Heading Margin	2 times VS (2 vertical line spaces) or 0, if page length is less than 15.
Bottom Margin	6 times VS (6 vertical line spaces) or for the 3800 Printer, 3 times VS or 0, if page length is less than 15.
Footing Space	VS (1 vertical line space) or 0, if page length is less than 15.
Footing Margin	2 times VS (2 vertical line spaces) or 0, if page length is less than 15.

FONT TABLE MAINTENANCE

When formatting documents for the 3800 Printer, SCRIPT/VS makes use of font tables for each of the fonts named in the CHARS option of the SCRIPT command. Each font table describes the font in terms of its name, pitch, and the width of each character.

The module DSMFT381 contains one font table (FTB) for each known font. The FTBs are created by the DSMSFTD macro and are mapped by the DSMSFTB DSECT.

UPDATING THE FONT TABLE (FTB)

Use the following procedure to add or change a font table:

1. Obtain a listing of the DSMSFTD macro and of the DSMFT381 module from your library.

2. Code a DSMSFTD macro that describes the new font.

   or

   Modify the DSMSFTD macro that describes the font whose characteristics you want to change. (The macro's field names are described in detail in "FTB Field Descriptions," which follows.)

```
*                 0 1 2 3 4 5 6 7 8 9 A B C D E F       0123456789ABCDEF
GP12W     DC    X'0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F' 00  ----------------
          DC    X'0F120F0C0F0F0F0F0F0F0F0F0F0F0F0F' 10  ----------------
          DC    X'0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F' 20  ----------------
          DC    X'0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F' 30  ----------------
          DC    X'0F0F0F0F0F0F0F0F0F0F0C0F0C0F0C0C' 40   ---------¢.<(+|
          DC    X'120F0F0F0F0F0F0F0F0F0C0F0F0C0C0F' 50  &---------:$*);-
          DC    X'0F120F0F0F0F0F0F0F0F0C12120F0F' 60    -/---------,%_>?
          DC    X'0F0F0F0F0F0F0F0F0F0F0C12120C0F0F' 70  ----------:#@'="
          DC    X'0F0F0F0F0F0F0C0F0F0C0F0C0C0C0F' 80    -abcdefghi------
          DC    X'0F0C0F0C120F0F0F0F0F0C0F0C0F0C' 90    -jklmnopqr------
          DC    X'0C0F0F0C0F0F120F0F0F0F12120C0F0C' A0  --stuvwxyz------
          DC    X'0C0C0C0C0C0C0C0C0C0C0F12120C0F12' B0  ----------------
          DC    X'0F0F0F0F0F0F0F0F0F0F0C0F0F0F0F0F' C0  -ABCDEFGHI------
          DC    X'0F0C0F0C120F0F0F0F0F0F0F0F120F0F' D0  -JKLMNOPQR------
          DC    X'120F0F0F0F0F120F0F0F0F0F0F0F0F0F' E0  --STUVWXYZ------
          DC    X'0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F' F0  0123456789------
*                 0 1 2 3 4 5 6 7 8 9 A B C D E F       0123456789ABCDEF
```

Figure 40. Example of a Font Width Table: GP12 is a 12-pitch proportional-spaced font with psuedo-blanks.

---

3. Assemble the DSMFT381 module to include the changes you've made.

4. Link the newly-assembled version of the DSMFT381 module with the rest of SCRIPT/VS. (See the Program Directory for details.)

FTB FIELD DESCRIPTIONS

Each font table (FTB) macro is specified as:

```
DSMSFTD   parm1,          Font name
          parm2,          Character width table
          parm3,          Box character vector
          parm4,          Pitch
          parm5           Font type code
```

The content and meaning of each font table (FTB) field that you can specify with the DSMSFTD macro are shown below:

FTB Field	Description
parm1	The font name: 1 to 4 characters.
parm2	The address of the font's width table. The width table contains a one-byte entry for each of the 256 character codes. Each entry specifies the width of the character as a binary number of device units. See Figure 40 for an example of the width table.
parm3	The address of the box character set vector.
parm4	The font's pitch (characters per horizontal inch); either 10, 12, or 15.

parm5 A one- or two-character code to specify the font type:

M Monospace font

MB Monospace font with special blanks

PB Proportional-spaced font with special blanks

Special blanks are described in "Appendix D. Formatting Considerations for the 3800 Printer" on page 337.

FONTS PROVIDED WITH SCRIPT/VS

The fonts provided with SCRIPT/VS for use with the 3800 Printer are listed and illustrated in "Appendix C. Fonts Supplied with SCRIPT/VS" on page 329.

3800 PRINTER FONTS SUPPORTED BY SCRIPT/VS

The fonts provided with the 3800 Printer are listed in Figure 39 on page 321. For details on creating a character arrangement table and its corresponding character set, see IBM 3800 Printing Subsystem Programmer's Guide.

The fonts illustrated in this appendix are provided with SCRIPT/VS for use with the 3800 Printer. One or two font names can be specified with the CHARS option of the SCRIPT command (see "Chapter 2. Using the SCRIPT Command" on page 13 for details). The SCRIPT/VS fonts cannot, in general, be combined in the CHARS option with the IBM 3800 fonts listed in Figure 39 on page 321.

Figure 41 lists the fonts provided by SCRIPT/VS for use with the 3800 Printer. Each is a full uppercase and lowercase font.

Text Fonts
GT10 Gothic (10-pitch) GT12 Gothic (12-pitch) GT15 Gothic (15-pitch) ST10 Serif (10-pitch) ST12 Serif (12-pitch) ST15 Serif (15-pitch)

Highlight Fonts
GB10 Gothic Bold GB12 Gothic Bold GI12 Gothic Italic SI10 Serif Italic SI12 Serif Italic SB12 Serif Bold

Special Fonts
GR10 Gothic Reverse GP12 Proportional RT10 Roman Text SO12 Serif Overstruck

Figure 41. Fonts Provided With SCRIPT/VS: Each font is a complete set of upper- and lower-case characters. Any two of these fonts may be specified with the CHARS option of the SCRIPT command.

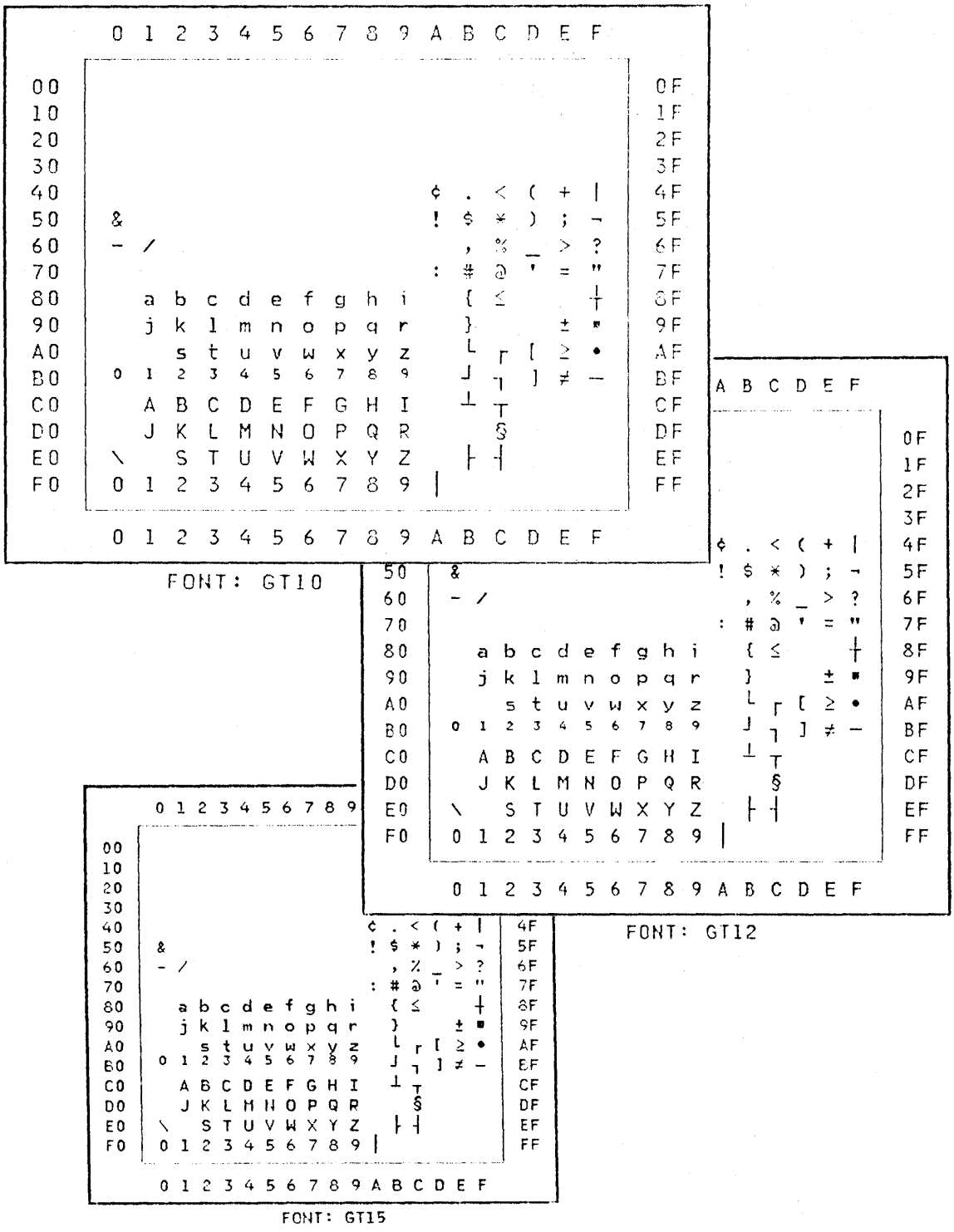


Figure 42. SCRIPT/VS Fonts: Gothic Text

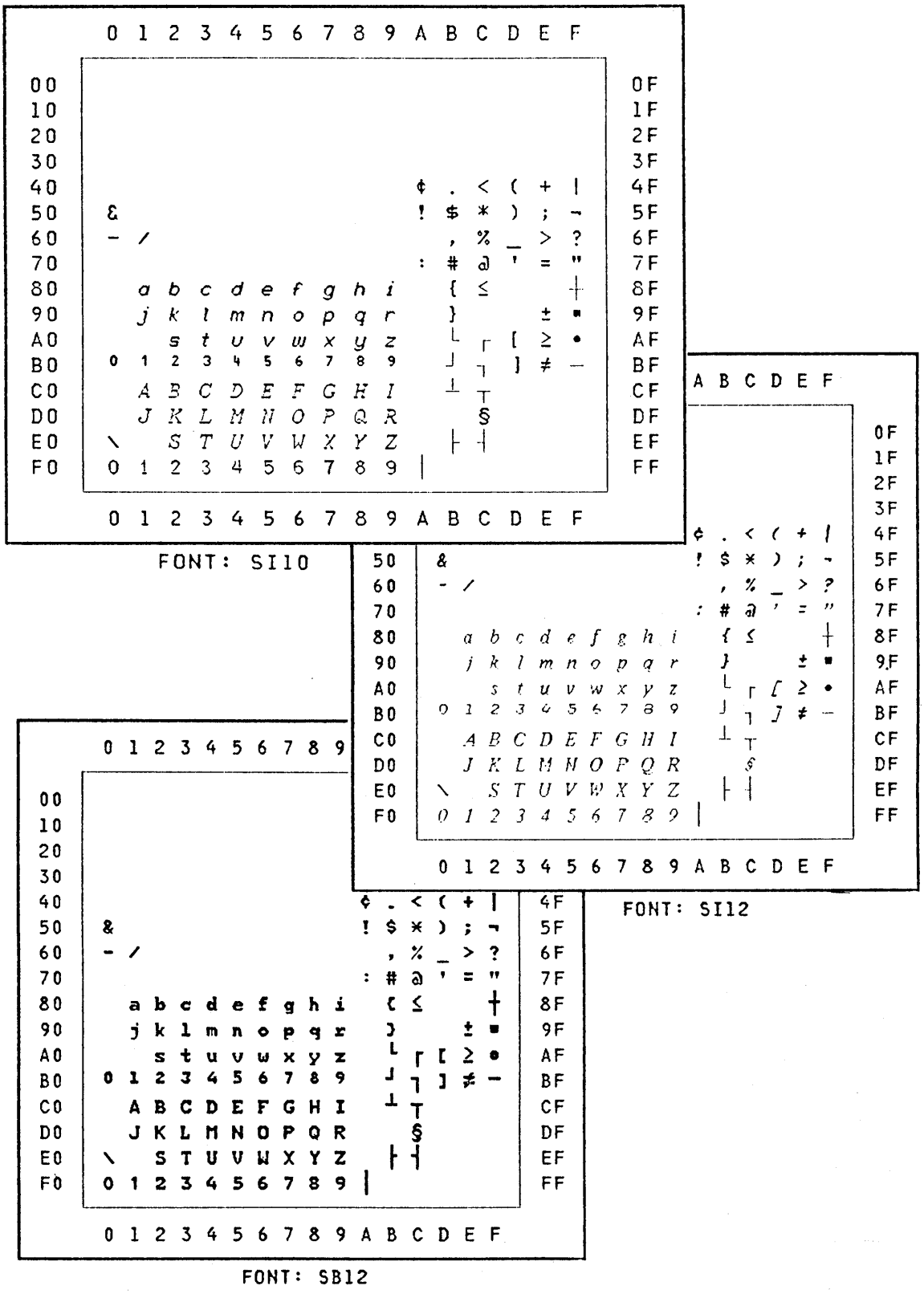


Figure 43. SCRIPT/VS Fonts: Serif Text

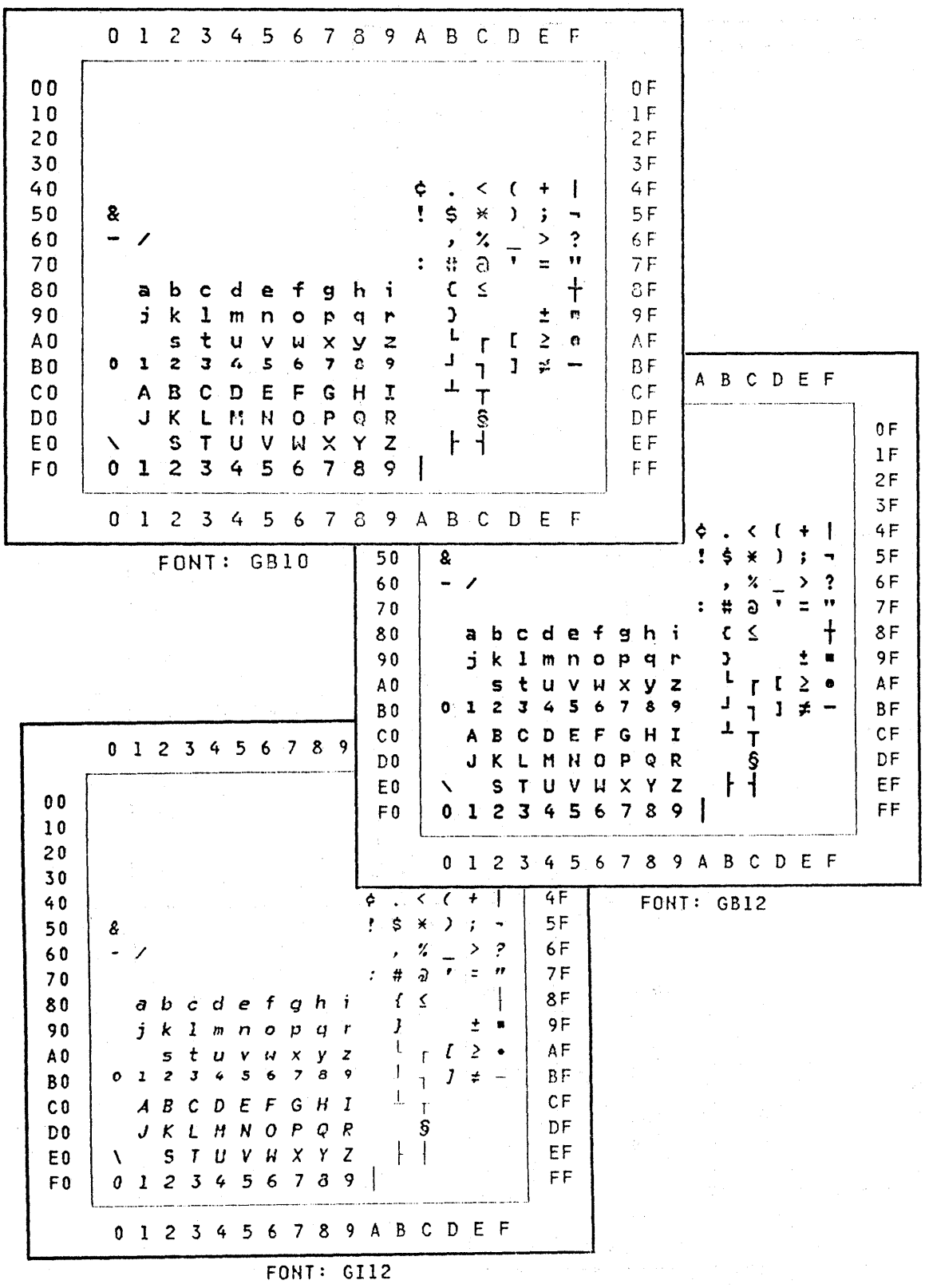


Figure 44. SCRIPT/VS Fonts: Gothic Highlight

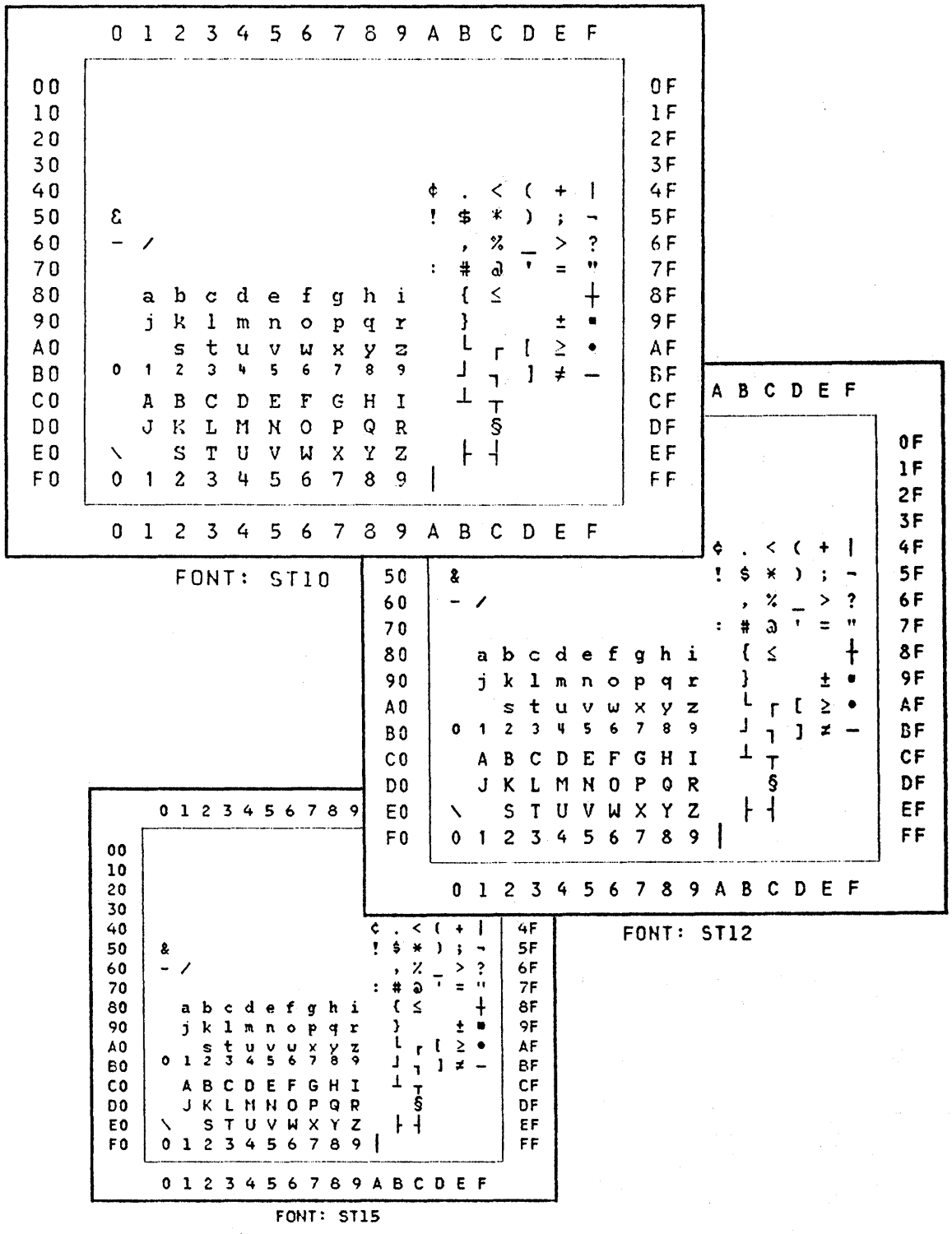
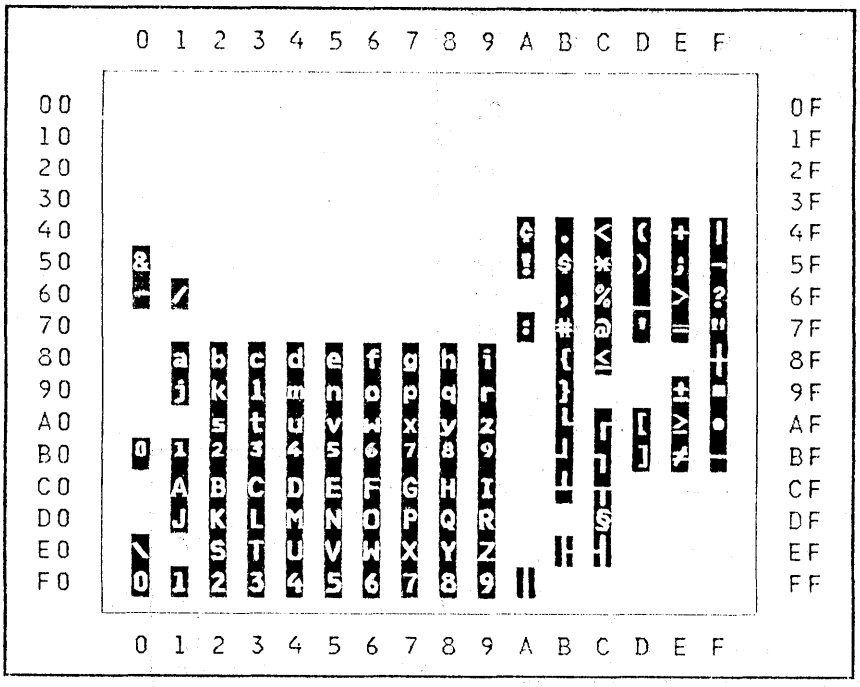
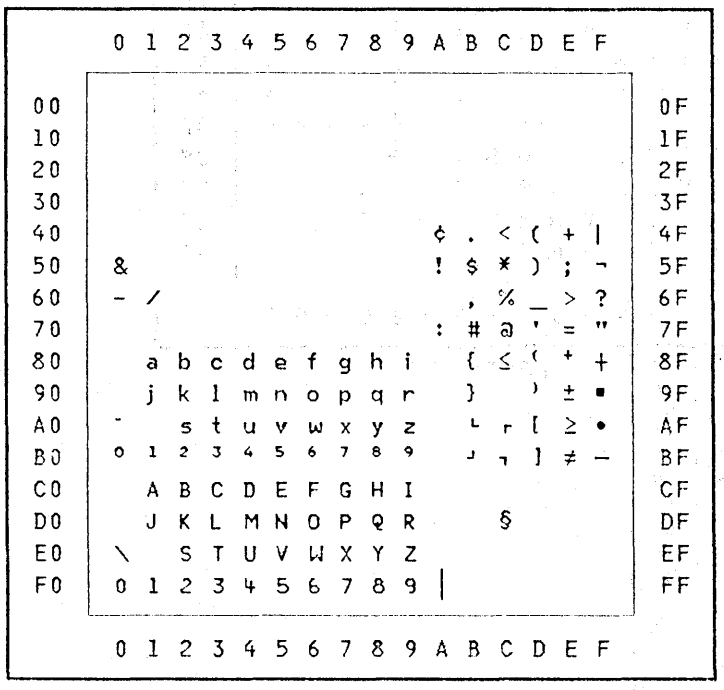


Figure 45. SCRIPT/VS Fonts: Serif Highlight

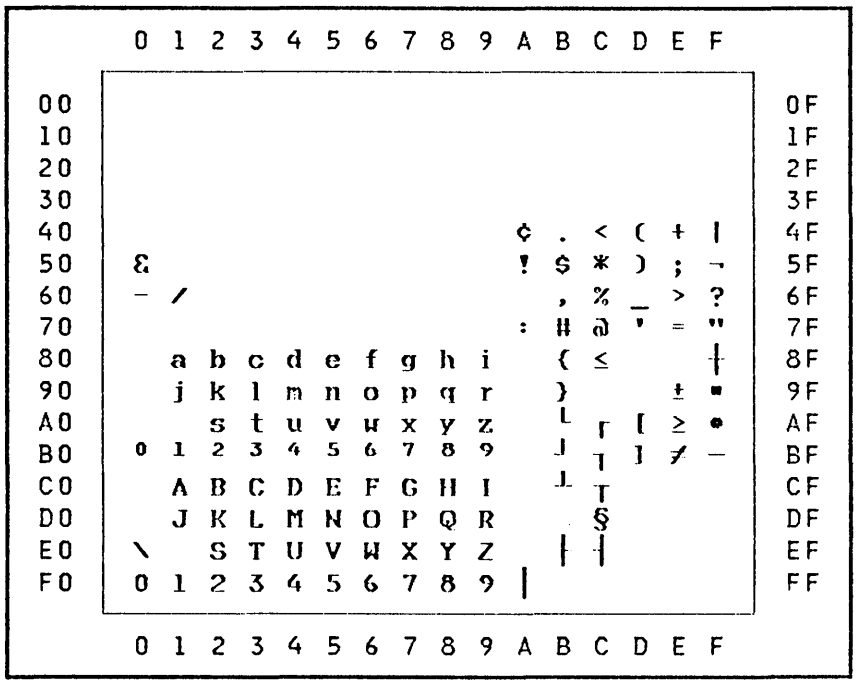


FONT: GR10

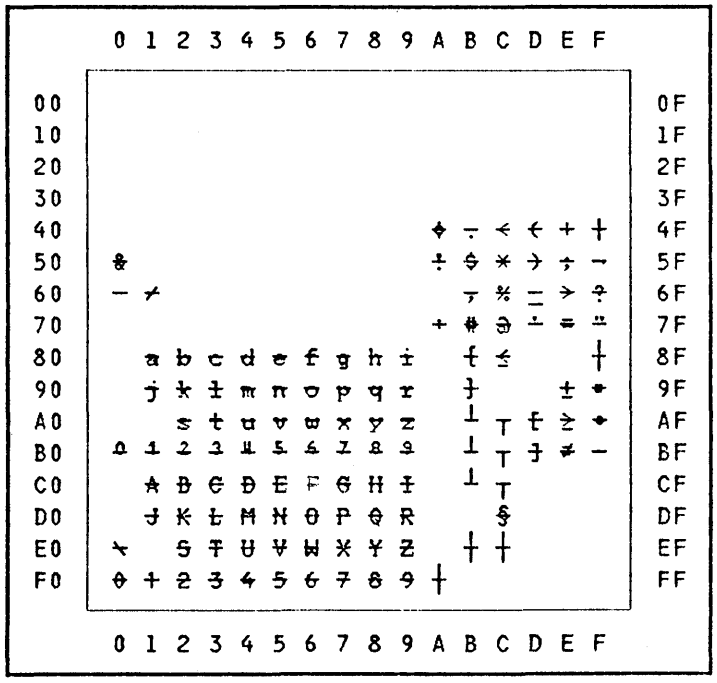


FONT: GP12

Figure 46. SCRIPT/VS Fonts: Gothic Special Purpose



FONT: RT10



FONT: S012

Figure 47. SCRIPT/VS Fonts: Serif Special Purpose

APPENDIX D. FORMATTING CONSIDERATIONS FOR THE 3800 PRINTER

This section contains some information and simple guidelines to help you when formatting documents for the 3800 Printer.

Before reading this section, you are expected to have a working knowledge of the control word syntax and functional capabilities of SCRIPT/VS. Additionally you should be familiar with the 3800 Printer hardware and its system control program (SCP) support. Information about the 3800 Printer can be found in IBM 3800 Printing Subsystem Programmer's Guide.

FONT MANAGEMENT

The current font may potentially change each time a new input line is processed. A new font may be started at any point in the input stream. This may be between words or within a continued word. At the time a new font is started, the following information is available to the formatter:

- Address of the "font width table"
- Table Reference Character (TRC) for this font
- Availability of "special blanks"[15]

Line formatting proceeds based on these parameters until another font change is requested.

Fonts of different pitch that do not contain special blanks cannot be used on the same output line. Failure to observe this restriction may result in severe column misalignment.

A single output line includes all data in all columns that occupy a single print line position on the output device.

Usually, the desired results may be achieved using SCRIPT/VS-supplied fonts which contain the special blanks. However, if a local font is required, it is recommended that you supply "graphmods" for the special blank character codes.

Care should be taken in the definition of new fonts to observe the following conventions:

- The font should contain a multiple of 64 characters less one. This ensures that the Writable Character Generation Matrix (WCGM) storage in the 3800 Printer is used efficiently (that is, has no unused WCGM positions).
- The first WCGM position (hexadecimal 00) should be assigned to the normal blank, usually hexadecimal 40, rather than SCRIPT/VS special blanks. Data positions in the output line that are not otherwise assigned (that is, unprintable characters) will be assigned to this character by the 3800 Printer.
- The underscore character (hexadecimal 6D) should be assigned to the 45th WCGM position (hexadecimal 2D). The 3800 Printer assumes that WCGM position 45 is the underscore character. If this position is unassigned, underscores will not appear in the output. Assigning this position to a character other than the underscore may cause unpredictable results.
- The last WCGM position (hexadecimal 3F) should not be assigned to any character code.

---

[15] Hexadecimal 11 is 10 pitch. Hexadecimal 12 is 12 pitch. Hexadecimal 13 is 15 pitch.

TAB, BACKSPACE, UNDERSCORE RESOLUTION

It is necessary to resolve backspaces and tabs before line formatting can begin. The tab, backspace, or underscore characters,[16] when processed, cause changes in the input line data string.

If no fill character has been specified in the tab definition, tabs are expanded by inserting hexadecimal 40 characters or special blanks in appropriate combinations to fill the space from the character preceding the tab to the next defined tab stop position. Current character position is measured in pels from the beginning of the column, including indention.

A minimum space of 23 pel is required from the current line position to the tab stop. If the space is less than 23 pel, the next defined tab stop is used.

This minimum value guarantees that the tab expansion will end within 1 pel of the desired tab stop position if special blanks are available. See "Inline Space Management" on page 339 for more information.

All data to the left of a tab expansion is considered to be a single word segment. No wordspaces to the left of the tab will be considered for justification purposes.

Normal line folding[17] at wordspaces is also inhibited to the left of a tab expansion.

Backspaces and underscores normally are presented to the formatter in one of the following character triplet configurations:

1. character, backspace, underscore

   In this form, the subject character is underscored.

2. underscore, backspace, character

   In this form, the subject character is underscored.

3. character, backspace, character

   In this form, the first character is deleted from the input line. That character position is then occupied by the character immediately following the backspace. This is done because overprinting is not possible on the 3800 Printer.

   The character with the highest collating sequence will always be the second character and consequently will be the one to be printed on the 3800 Printer.

When an underscore is encountered out of triplet context, it is treated as data; no special processing is done.

INTERWORD SPACE

If justification is on and special blanks are present, all interword spaces in the input line are translated to hexadecimal 13, the 15-pitch blank.

When an input line does not fill the column width, a wordspace is added to the end. Successive input lines are formatted with intervening spaces until the column width is filled.

- If justification is on, a single wordspace is added.

---

[16] Hexadecimal 05, 16, and 6D are tab, backspace, and underscore, respectively.
[17] .FO ON or .CO ON.

- If justification is off, and the input line ends with a full stop character, two wordspaces are added.
- When the input line ends in a continuation character, no wordspace is added.

REVISION CODE CHARACTERS

The revision code character is normally placed immediately preceding each changed line and is followed by a blank. Because the RC field has a variable width based on the width of the RC character, it is necessary to measure and format it in the same way as text data.

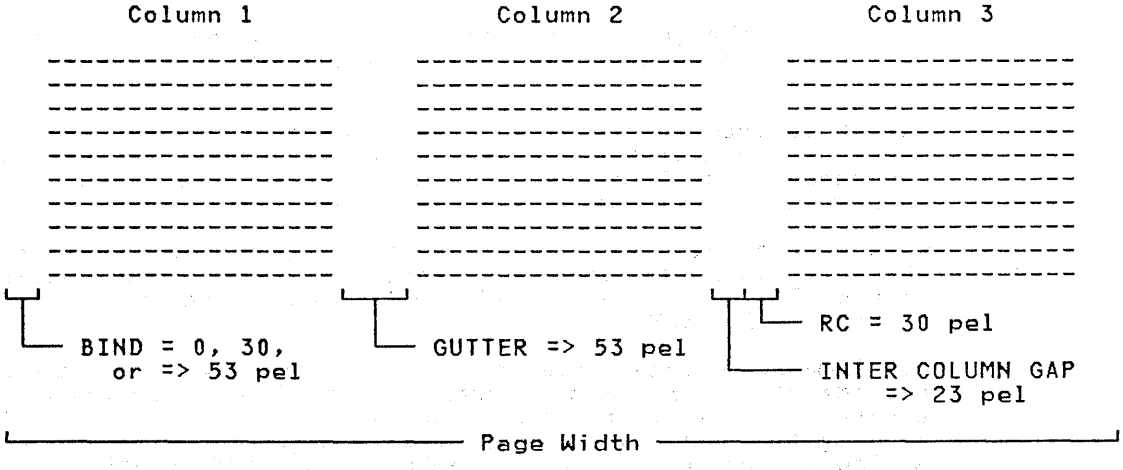
It is most desirable for the first character of each text line to start in the same relative position. To ensure this, the RC character and its blank must have a combined width that does not vary from line to line. If special blanks are present, this is achieved by combining the RC character with a special blank which brings the total width of RC and blank to 30 pel. The following table shows relative widths:

RC WIDTH	BLANK WIDTH
12	18
15	15
18	12

The RC field is allocated from the bind space in column 1 and from the gutter space in columns 2 through n. If insufficient space is available it will not appear in the output.

A minimum space of 53 pel is required in the gutter to ensure column alignment within 1 pel. The bind space should be a minimum space of 30 pel to provide adequate space for the RC field.

The following example shows the relationship of the RC field to the bind and gutter space in a three-column text section.



INLINE SPACE MANAGEMENT

Any time there is a need to fill some space in the output line, a space character string must be generated which will have a measured length equal to that of the desired space. This character string will be hexadecimal 40 characters, or a combination of special blanks. The accuracy of the length of a space string has a direct effect on the column alignment of the output line of which that string is a part.

The best accuracy that may be hoped for with the hexadecimal 40 string is ± one-half hexadecimal 40 character width. With care, one may set parameters in such a way as to minimize the probability that half-character alignment errors will occur.

One-half-character rounding errors may occur when space units are not specified in multiples of the character width. This occurs because the space unit value is resolved to native device units, pels in the horizontal direction, and may not always be satisfied accurately with a string of hexadecimal 40 characters. The situation is not affected by the magnitude of the request but by the relationship of the space value to the width of the hexadecimal 40 character at the point in time when the space is generated.

The best accuracy that may be hoped for with the special blank string is ± one pel. This level of accuracy is the best attainable on the 3800 Printer and is quite satisfactory for most applications. This level of accuracy can normally be expected when the following simple guidelines are followed:

- Define tabs so that data preceding the tab always has room to end at least 23 pel before the defined tab stop position.
- If you are not using the default, bind should be set to one of the following values:

  0, 30 pel, or greater than 52 pel.
- Define columns so that the gutter is at least 53 pel wide.
- Specify indent values equal to or greater than 23 pel.
- Limit split text filler strings to 1 or 2 characters.

If special blanks are present and the above guidelines are not followed, inline space errors of ± six pel may be encountered. This is because it is not always possible to satisfy space requests accurately if the value requested is less than 23 pel.

With a 10-pitch font, errors of ± 9 pel may be encountered.

Alignment errors may potentially occur any time a space string is generated. The cumulative effect across a multicolumn line may be much greater than six pel.

Figure 48 on page 341 illustrates the alignment errors that will be encountered when using special blanks and space values of less than 23 pel.

As can be seen from the table, the degree of error varies considerably in the 0 to 6 pel range for any request of less than 23 pel. Any request equal to or greater than 23 pel can be satisfied within 1 pel by a combination of the special blanks.

When special blanks are present, and a space request is processed which can not be accurately satisfied, a warning message is issued. The line in error will be flagged in the right margin with "<---- SPACE ERROR."

The formatter takes steps to ensure that inline space is specified outside the error windows shown in the table. The area in which errors are most likely to occur is in the gutter space. This is because the column definition may be disregarded when processing a line longer than column length with the EXTEND option of the .FO [Format Mode] control word. For this reason, the use of EXTEND is not recommended.

BOX PROCESSING

Boxes are supported in logical overlay mode relative to the output line; the box characters are overlaid on the output line after it is completely processed. Characters in the box line that occupy the same print position as a text character are printed; the text character does not appear in the output.

REQUESTED (PEL)	ACTUAL (PEL)	ERROR (PEL)
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 48. Justification Alignment Error for 3800 Printer Output: Horizontal space cannot be reliably generated for distances less than 23 pel.

The use of special blanks is inhibited while box processing is in effect. That is, from the .BX definition input line to the .BX OFF input line. Slight variations in column alignment may occur in transition from the normal formatting environment, when special blanks are available for inline space management, to the box formatting environment, when hexadecimal 40 characters are used for inline space management. See "Inline Space Management" on page 339 for more information.

The following restrictions apply to box processing only:

- All characters in the box line and all characters in the text within the box must be of the same width.
  - Truly proportional fonts may not be used for the box characters or the text within a box. The special blank characters may be present in the font, but all other characters must be of the same width.
  - All fonts used within a box must be of the same pitch as the box font. The box font is the current font at the time the box is defined.

FORMATTER ESCAPE CHARACTER

Formatted lines contain imbedded controls which are prefixed with the "escape" character hexadecimal 27. This use of hexadecimal 27 by SCRIPT/VS precludes its use as a data character code.

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.

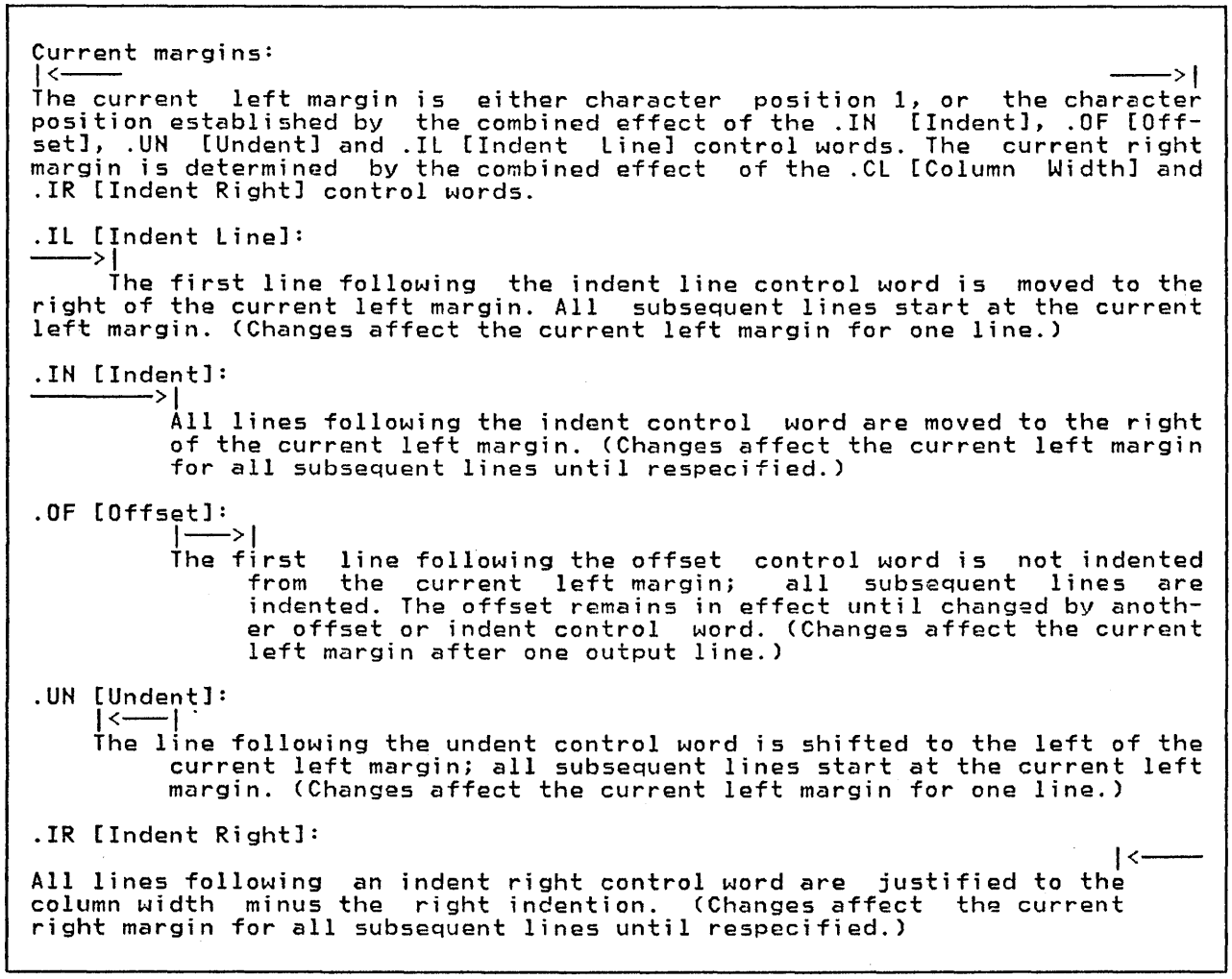


Figure 49. How the Current Margins Are Established

ampersand: The "&" character.

When an ampersand begins a character string, SCRIPT/VS assumes the character string is a symbol name. If the symbol name is defined, SCRIPT/VS replaces the symbol with its value (unless symbol substitution is off).

In running footings, running headings, and running titles, the ampersand is usually the page number symbol.

When encountered by itself on the right side of a .SE [Set Symbol] control word, it is interpreted as the page number symbol.

APF: See Application Processing Function.

Application Processing Function (APF): In GML processing, the processing that is performed when a document element or attribute is recognized. In SCRIPT/VS, an APF is implemented as a sequence of control words, possibly intermixed

with text and symbols, in one of three forms: macro definition, value of a symbol, or imbedded file.

attribute: A characteristic of a document (or document element) other than its type or content. For example, the security level of a document or the depth of a figure.

attribute label: In GML markup, a name of an attribute that is entered in the source document when specifying the attribute's value.

back matter: In a book, those sections (such as glossary and index) that are placed after the main chapters or sections.

balancing: In multicolumn formatting, the process of making column depths on a page approximately equal.

batch environment: The environment in which non-interactive programs are executed.

binding edge: The edge of a page to be bound, stapled, or drilled. Defined with the BIND option of the SCRIPT command. (See also Figure 21 on page 298.)

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. (See also Figure 21 on page 298.)

boldface: A heavy-faced type. Also, printing in this type.

bottom margin: On a page, the space between the body or the running footing, if any, and the bottom edge of the page. The bottom margin area includes the bottom title lines, if any. (See also Figure 21 on page 298.)

bottom title: Up to six lines of data repeated at the bottom of consecutive pages (or of consecutive odd- or even-numbered pages) in the footing space. (See also Figure 21 on page 298.)

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.[18]

character spacing: The space between characters in a word.

cicero: In the Didot point system, a unit of 0.1776 inches (4.512 millimeters) used in measuring typographical material.

CMS: An interactive processor that operates within VM/370.

column width: The width of each text column on a page. Specified with the .CL (Column Width) control word. (In multicolumn formatting, all columns on a page usually have the same width.) (See also Figure 21 on page 298.)

command: A request from a terminal or specified in a batch processing job for the performance of an operation or the execution of a particular program. For example, a request given at a terminal for SCRIPT/VS to format a document, or for an editor to edit a line of text.

comment: A control word line which is ignored by SCRIPT/VS. Such lines begin with either ".*" or ".cm". (See "Chapter 3. Basic Text Processing" on page 29 .)

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 mar-

---

[18] American National Dictionary for Information Processing

gin when indention is changed with the .IN (Indent), .UN (Undent), .IL (Indent Line), and .OF (Offset) control words. (See also Figure 49 on page 343.)

current line: The line in a source document at which a computer program (such as an editor or a formatter) is positioned for processing.

debug: To detect, trace, and eliminate errors in computer programs and SCRIPT/VS documents.

default value: A value assumed by a computer program when a control word, command, or control statement with no parameters is processed.

dictionary: A collection of "word stems" that is used with the spelling verification and automatic hyphenation functions.

Didot point system: A standard printer's measurement system on which type sizes are based. A Didot point is 0.0148 inches (0.376 millimeters). There are 12 Didot points to a cicero. (See also cicero and point.)

document: (1) A publication or other written material. (2) A machine-readable collection of lines of text or images, usually called a source document. (See also output document and source document.)

document conversion processor: A computer program that processes a machine-readable document which includes formatting controls written in one formatter language, to produce a machine-readable document which includes formatting controls appropriate for another formatter language.

document library: A set of VSAM data sets, accessible in a batch environment, that contain documents and related files.

duplex: A mode of formatting appropriate for printing on both sides of a sheet.

EBCDIC: Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.[19]

edit: To create or modify the contents of a document or file. For example, to insert, delete, change, rearrange, or copy lines.

editor: A computer program that processes commands to enter lines into a document or to modify it.

eject: In formatting, a skip to the next column or page.

em: A unit of measure for a particular font that is equal to the point size of that font.

extended symbol processing: The processing of a symbol whose value causes the remainder of the line to be stacked and later processed as a new input line.

fill character: The character that is used to fill up a space; for example, blanks used to fill up the space left by tabbing.

float: (1) (noun) A keep (group of input lines kept together) whose location in the source file may vary from its location in the printed document. (2) (verb) Of a keep, to be formatted in a location different from its location in the source file.

flush: Having no indention.

fold: (1) To translate the lowercase characters of a character string into uppercase. (2) To place that portion of a line which does not fit within a column on the next output line.

font: An assortment of type, all of one size and style.

font set: The set of fonts to be used in formatting a source document.

footing: Words located at the bottom of the text area. (See also running footing, bottom title, and Figure 21 on page 298.)

footing margin: That part of the bottom margin area between the body of the page or running footing, if any, and the bottom title(s), which is located in the footing space. (See also Figure 21 on page 298.)

footing space: That part of the bottom margin that is available for bottom title(s). (See also Figure 21 on page 298.)

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.

---

[19] American National Dictionary for Information Processing

format: (1) (noun) The shape, size, and general makeup of a printed document. (2) (verb) To prepare a document for printing in a specified format.

formatting mode: In document formatting, the state in which input lines are concatenated and the resulting output lines are justified.

formatter: (1) A computer program that prepares a source document to be printed. (2) That part of SCRIPT/VS that formats input lines for a particular logical device type.

front matter: In a book, those sections (such as preface, abstract, table of contents, list of illustrations) that are placed before the main chapters or sections.

Generalized Markup Language (GML): A language that may be used to identify the parts of a source document without respect to particular processing.

GML: Generalized Markup Language

gutter: In multicolumn formatting, the space between columns. (See also Figure 21 on page 298.)

hanging indention: The indention of all lines of a block of text, following the first line (which is not indented the same number of spaces). Specified with the .OF (Offset) or .UN (Undent) control word. (See also Figure 49 on page 343.)

head-level: The typeface and character size associated with the words standing at the beginning of a chapter or chapter topic.

heading: Words located at the beginning of a chapter or section or at the top of a page. (See also head-level, running heading, and top title, and Figure 21 on page 298.)

heading margin: That part of the top margin area between the body of the page or running heading, if any, and the top title, which is located in the heading space. Specified with the .HM (Heading Margin) control word. (See also Figure 21 on page 298.)

heading space: That part of the top margin area that is available for top title(s). Specified with the .HS (Heading Space) control word. (See also Figure 21 on page 298.)

hexadecimal: Pertaining to a number system based on 16, using the sixteen digits 0, 1, . . . 9, A, B, C, D, E, and F. For example, hexadecimal 1B equals decimal 27. (See also EBCDIC.)

indent: To set typographical material to the right of the left margin.

indention: The action of indenting. The condition of being indented. The blank space produced by indenting. Specified with the .IN (Indent), .UN (Undent), .OF (Offset), and .IL (Indent Line) control words. (See also hanging indention and Figure 49 on page 343.)

initial caps: Capital letters occurring as the first letter of each word in a phrase. To set a phrase in initial caps is to capitalize the first letter of each word in the phrase.

initial value: A value assumed by SCRIPT/VS for a formatting function until the value is explicitly changed with a control word. The initial value is assumed even before the control word is encountered, whereas the default value is assumed when the control word is issued without parameters. (See also default value.)

input device: A machine used to enter information into a computer system (for example, a terminal used to create a document).

input line: A line, as entered into a source file, to be processed by a formatter.

interactive: Pertaining to an application in which entries call forth a response from a system or program, as in an inquiry system. An interactive system might also be conversational, implying a continuous dialog between the user and the system. Interactive systems are usually communicated with via terminals, and respond immediately to commands. (See also foreground.)

interactive environment: The environment in which an interactive processor operates.

italic: A typestyle with characters that slant upward to the right.

JCL: Job control language.

job control language (JCL): A language of control statements used to identify a computer job or describe its requirements to the operating system.[20]

job control statement: A statement that provides an operating system with information about the job being run.

justify: To insert extra blank space between the words in an output line to cause the last word in the line to

---

[20] American National Dictionary for Information Processing

reach the right margin. As a result, the right-hand edge of each output line is aligned with preceding and following output lines.

keep: (noun) In a source document, a collection of lines of text to be printed in the same column. When the vertical space remaining in the current column is insufficient for the block of text, the text is printed in the next column. (In the case of single-column format, the next column is on the next page.)

layout: The arrangement of matter to be printed. (See also format.)

leader: (1) Dots or hyphens (as in a table of contents) used to lead the eye horizontally. (2) The divider between text and footnotes on a page (usually a short line of dashes, although you can redefine it).

left-hand page: The page on the left when a book is opened; usually even-numbered.

line spacing: The space between the baseline of one output line and the baseline of the adjacent output line.

lowercase: Pertaining to small letters as distinguished from capitals; for example, "a, b, g" rather than "A, B, G."

machine-readable: Data in a form such that a machine can acquire or interpret (read) it from a storage device, from a data medium, or from another source.

macro: An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language.[21] In SCRIPT/VS, a macro is a sequence of one or more control words, symbols, and input lines. A macro's definition can be recursive.

macro substitution: During formatting, the substitution of control words, symbols, and text for a macro.

margin: (1) The space above, below, and on either side of the body of a page. (2) The left or right limit of a column. (See also Figure 21 on page 298.)

mark up: (verb) (1) To determine the markup for a document. (2) To insert markup into a source document.

markup: (noun) Information added to a document that enables a person or system to process it. Markup may describe the document's characteristics, or it may specify the actual processing to be performed. In SCRIPT/VS, markup consists of GML tags, attribute labels and values, and control words.

offset: (verb) To indent all lines of a block of text, except the first line. (noun) The indention of all lines of a block of text following the first line. (See also Figure 49 on page 343.)

option: Information entered with the 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.

paginate: To number pages.

parameter: Any one of a set of properties whose values determine the characteristics or behavior of something. The syntax of some SCRIPT/VS control words includes parameters, which establish the properties of a formatting function or a printed page.

PDS: Partitioned Data Set.

pel: The unit of horizontal measurement for the IBM 3800 Printing Subsystem. One pel equals approximately 1/180th inch.

pica: A unit of about 1/6 inch used in measuring typographical material. Similar to a cicero in the Didot point system.

pitch: A number that represents the amount of horizontal space a font's character occupies on a line. For example, 10-pitch means 10 characters per inch, or each character is 0.1 (1/10) inch wide. 12-pitch means 12 characters per inch, and 15-pitch means 15 characters per inch.

point: (1) A unit of about 1/72 of an inch used in measuring typographical material. There are twelve points to the pica. (2) In the Didot point system, a unit of 0.0148 inches. 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

---

[21] American National Dictionary for Information Processing

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.)

right-hand page: The page on the right when a book is opened; usually odd-numbered.

rule: (noun) A straight horizontal or vertical line used, for example, to separate or border the parts of a figure or box.

running footing: A footing that is repeated above the bottom margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the page's body (text area). (See also Figure 21 on page 298.)

running heading: A heading that is repeated below the top margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the page's body (text area). (See also Figure 21 on page 298.)

running title: In SCRIPT/VS, up to six lines of data that may be repeated in the top or bottom margin area of consecutive pages (or of odd- or even-numbered pages.)

section: When an output page has two or more single-column parts with the same or different column-widths, or a single-column part and a multicolumn part, or two or more different multicolumn parts, each part of the output page is called a section.

small caps: Capital letters in the same style as the normal capital letters in a font, but approximately the size of the lowercase letters.

source document: A machine-readable collection of lines of text or images that is used for input to a computer program.

space: A blank area separating words or lines.

symbol: A name in a source document that can be replaced with something else. In SCRIPT/VS, a symbol is replaced with a character string. SCRIPT/VS may interpret the character string as a number, a character string, a control word, or another symbol.

symbol substitution: During formatting, the replacement of a symbol with a character string which SCRIPT/VS may interpret as a value (numeric, character string, or control word) or as another symbol.

tab: (1) (noun) A preset point in the typing line of a typewriter-like terminal. A preset point in an output line. (2) (verb) To advance to a tab for printing or typing. (3) a tab character, hexadecimal 05.

tag: In GML markup, a name for a type of document (or document element) which is entered in the source document to identify it. For example, ":p." might be the tag used to identify each paragraph.

terminal: A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

text line: An input line that contains only text.

title: See running title.

token: A string of characters which is treated as a single entity. In SCRIPT/VS, a parameter passed to a macro in one of the local variables &*1, ... &*n. (See "Chapter 12. Writing SCRIPT/VS Macro Instructions" on page 137.)

top margin: On a page, the space between the body or running heading and the top edge of the page. The top margin includes the top titles, if any. (See also Figure 21 on page 298.)

top title: Up to six lines of data repeated at the top of consecutive pages (or of consecutive odd- or even-numbered pages) in the heading space. (See also running title and Figure 21 on page 298.)

TRC: Table Reference Character. In printer SYSOUT datasets, a second control byte, following the carriage control byte, which indicates which font the record is to be printed in. The presence of TRCs is indicated by the JCL parameter DCB=OPTCD=J.

TSO: An interactive processor within OS/VS2.

typeface: All type of a single style. There might be several fonts (different sizes) with the same typeface or style.

typeset: (1) (verb) To arrange the type on a page for printing. (2) (adjective) Pertaining to material that has been set in type.

underscore: (1) (noun) A line printed under a character. (2) (verb) To place a line under a character. To underline.

unformatted mode: (1) In document formatting, the state in which each input line is processed and printed without formatting. Other SCRIPT/VS control words remain in effect and are recognized. (2) In document printing using the UNFORMAT option, the state in which each input line (control words as well as text) is printed as it exists in the input, in the order in which it is processed. No formatting is done.

uppercase: Pertaining to capital letters, as distinguished from small letters; for example, "A, B, G" rather than "a, b, g."

widow: One or two lines of a paragraph that are printed separately from the rest of the paragraph.

word spacing: The space between words in a line. Also called wordspace.

SPECIAL CHARACTERS

"*" parameter
  of .RC control word 86, 267
& 343
&A' converting numeric to a base-26 number 124
&B inserting bulleted item 171
&E' verifying existence of symbol 122
&Hx inserting numbered head levels 172
&L' determining length of symbol's value 123
&Nx inserting a numbered item 172
&P starting new paragraph 172
&R' converting symbol to Roman numeral 124
&T' analyzing the type of the symbol 123
&toc generating a table of contents 172
&U' converting lowercase to uppercase 125
&V' returning current value of symbol 123
&$RET special symbol 128

A

"A" parameter
  of .SK control word 278
  of .SP control word 279
accounting notes in input 49
ADD parameter
  of .DU control word 226
addenda dictionary
  adding or deleting words 226
  building an 160
  hyphenating words 159
adding lines
  to the table of contents 74
additional SCRIPT/VS formatting features
  drawing boxes 87
    box within a box 91
    boxes in a horizontal row 92
    .BX [Box] control word 207
    centering text within a box 89
    formatting text within a box 88
    horizontal row 92
    middle portion of box in another (larger) box 92
    only the bottom line 94
    only the middle portion 93
    only the top line 92
    open at the top and bottom 93
    stacking one box on another 90
  conditional text sections 102
  conditional character 82
  footnotes 84, 233
  keeping text together 82
    control words not allowed in keeps 83
  marking updated material 86
  starting text
    at top of column 85
    at top of page 85
    conditional column ejects 85
    conditional page ejects 85
    delaying imbedding of input text 85
  using fonts with the IBM 3800 Printing Subsystem 94
  using special characters 77
    character translation for terminal output 81
    customizing your keyboard 77
    defining special characters 81, 217
    input character translation 78
    using symbols for special characters 79
ALPH parameter
  of .PN control word 63, 262
alphabetic page numbering 63, 262
aligning text
  with the right margin 42, 271
ALL parameter
  of .IT control word 246
  of .RT control word 272
altering hyphenation parameters 157
ampersand
  as the page number symbol 61
  glossary 343
  inhibiting substitution 121
.AP [Append] control word 105
  description 203
  naming file to be appended 106
APF
  developing preprocessor 154
  glossary 343
  mapping from GML tag 146
  symbols within starter set 147
append control word 105
  description 203
  naming file to be appended 106
appendices
  device table maintenance 323
  font table maintenance 325
  fonts supplied with SCRIPT/VS 327
  formatting considerations for the 3800 Printer 337
  SCRIPT/VS summary 297
appending files 105
  naming 106
  symbols set when 129
application function sets 3
arabic numerals 63, 262
ARABIC parameter
  of .PN control word 63, 262
array elements, controlling 134
  accessing the index counter 135
  setting the index counter 135
array separator characters
  defining 81
array of symbols
  assigning values 275
  defining values 275
arrays of values
  controlling 134
  symbols for 133
ASEP parameter
  of .DC control word 81, 217
asterisk parameter
  of .RC control word 86, 267

ATMS-II conversion 187
  ATMS control--SCRIPT/VS macro
    relationship between 194
  converting documents
    conversion techniques 188
    formatting control 189
    non-format command 188
    to SCRIPT/VS format 187
ATMS-II GML identifier 188
attribute label 344
  glossary 344
attributes
  of a symbol's value 122, 318
  in GML 149
automatic
  hyphenation verification 157
  item numbering 174
  spelling verification 157
avoiding
  an endless loop with macros 141
  a text period in column one 47

B

back matter 344
backspace resolution
 for 3800 Printer 338
balance columns control word 67
  description 204
balancing 344
baseline
 vertical space of current line 278
basic page parameters 51
basic text formatting 29
  aligning text
    with both margins 43, 211
    with right margin 43, 271
  blank, leading 250
  blank lines 40
    multiple 41
  breaks 34, 206
  capitalizing text 44, 293
  centering text 42, 211
  changing the margin 35
    current left margin 35
    example of 39
    with tabs 39
  column
    begin 209
    conditional begin 209
    description 210
    multiple 254
    width 212
  column formatting
   for multiple columns 204
  comments in input file 49, 212
  comments in output file 257
  concatenate mode 31, 213
  control word separator 48, 215
  double spacing 225
  ejecting to a new page 45
    odd or even page 46, 230, 259
    page eject mode 46, 259
  footing
    running 269
    space 235
  font
    previous 261
    saving 278
  format mode 29, 234
  GML tags and control words 29
  grouping control words 48
  guidelines for entering text 46
  hanging indentation 38
  head levels
    defining 221, 240
    headnote 237
    heading space 238
    running 270
  hyphenating 239
  imbedding files 244
  implicit formatting conventions 31
  indenting text
    definition 35
    .IL control word 243
    .IN control word 245
    .UN control word 292
  inserting SCRIPT/VS file
   after another prints 203
  justify mode 29, 248
  line length 252
  line spacing
    definition 40
    .LS control word 252
    .SL control word 279
  literals 251
  margin
    bottom 205
    footing 232
    heading 237
    top 287
  new font 204
  null line 256
  offset text 258
  page eject
    conditional 214
    control word description 260
    delaying file until next 222
    even page 230
    odd page 259
  page length 261
  page numbering mode 262
  page number symbol 264
  paragraphing 174
  period in column one 47
  positioning lines on page 42
  revision code 267
  single column mode 274
  single space mode 280
  skipping lines 278
  special characters, defining 217
  spelling verification 281
  splitting text 282
  table of contents 265, 285
  tabs
    leading 253
    setting 284
  titles
    bottom 206
    bottom, even page 227
    bottom, odd page 256
    running 272
    top 289
    top, even page 230
    top, odd page 259
  underlining text 44
    .UC control word 290
    .UD control word 291
    .US control word 294
  using tabs 32
    fill characters 33
    setting tabs 32, 284
    with indentation 39
basic text formatting 29
batch environment
  calling SCRIPT/VS processor 6
  glossary 344
  using SCRIPT/VS in 153

.BC [Balance Columns] control word
definition 67
description 204
begin font control word 94
description 204
beginning a new column 69, 85
.BF [Begin Font] control word
definition 117
description 204
BIND option 17, 299
binding edge 344
blank, leading 250
blank lines 40, 278
input lines that begin with 142
.BM [Bottom Margin] control word
definition 53
description 205
(see also .FS control word)
body 344
boldface 344
bottom line of a box
printing it by itself 94
bottom margin 53
control word description 205
glossary 344
(see also .FS control word)
BOTTOM parameter
of .RT control word
definition 54
description 272
bottom running titles 54-59
glossary 344
(see also .FS control word)
bottom title control word 206
box 87-94
centering text within 89
characters 78
control word description 207
defining a 87
example of 87
description 207
drawing only bottom line 94
drawing only middle 93
within a larger box 91
drawing only top line 92
examples 208
in a horizontal row 92
overview of 9
parts of a 87
processing 340
stacking one on another 90
within a box 91
branching in input file 236
branching in macro 236
.BR [Break] control word
definition 34
description 206
BR parameter
of .DH control word 221
break 344
break control word 34, 206
breaks 34
control words that cause 47, 311
page 66
.BT [Bottom Title] control word
description 206
building an addenda dictionary 159
bullets 175
.BX [Box] control word
definition 87-96
description 207

C

"C" parameter
of .SK control word 278
of .SP control word 279
calling SCRIPT/VS processor 6
CAN parameter
of .BX control word 207
CANCEL parameter
of .RF control word 269
of .RH control word 270
cancelling a symbol 122
capitalizing words
lowercase in document 28
individually 44
caps 344
caption 344
CATALOG parameter
of .DD control word 219
.CB [Column Begin] control word
definition 69
description 209
.CC [Conditional Column Begin] control word
definition 69
description 209
.CD [Column Definition] control word
definition 65, 67
description 210
.CE [Center] control word
definition 42
description 211
center control word 42, 211
CENTER parameter
of .FO control word 234
centering text 42
ATMS-II conversion 190
between margins 211
with running footings 60
with running headings 60
within a box 89
changing the
font 94
left margin 35
left margin for only one line 37
right margin 35
SCRIPT command 177
SCRIPT/370 control words 178
CHAR parameter
of .BX control word 207
character, special 79
affects SCRIPT/VS processing 81
glossary 344
using symbols for 81
character set 344
character spacing 344
character string, symbol values 123
character translation
for terminal output 81
in the input line 78
in the output line 77
characters
that delimit sentences 82
that delimit words 82
that delimit words
for spelling verification 319
characters not underscored
by default 319
CHARS option 17, 299
cicero 344
.CL [Column Width] control word
definition 65
description 212

glossary 344
line length 252
(see also .LL control word)
CLOSE parameter
of .EF control word 228
.CM [Comment] control word
definition 50
description 212
CMS 344
CMS file naming conventions 13
CMS processing a system command 283
.CO [Concatenate Mode] control word
definition 31
description 213
suspending (with a break) 34
column
balancing 69, 204
begin control word 69, 209
beginning a new 69, 85, 209
for head levels 72
conditional begin 69, 209
boundary, extending 234
definition 65
description 210
eject 210
keeping text together 248
positions 67
saving current definition 274
truncating length 212
width 68, 212, 252, 344
column begin control word 69, 209
column one
avoid a text period in 47
enter all text in 47
column definition control word
definition 65, 67
description 210
column margins, splitting text 282
column width control word 212
(see also .LL control word)
combining SCRIPT/VS files 105
command 344
command conversion, non-format 188
comment control 192
comment control word 49
description 212
comments in SCRIPT/VS files
definition 49
.CM control word 212
comments, output 257
communicating with TSO 115
communicating with VM/370 114
compatibility
changes to the SCRIPT command 177
with SCRIPT/370 input files 10, 177
with TSO/FORMAT 197
composition 344
compound symbols 119
concatenate mode 31
control word description 213
suspending (with a break) 34
concatenating lines
canceling or restoring 213, 234
preventing 206
concatenation 344
conditional
column begin 69, 209
column eject 69, 85
input file processing 214
page eject 45, 85, 214
section 99, 102, 214
text sections 99
column begin control word
definition 69
description 209
conditional section control word
definition 102
description 214
nesting 103
conditional page eject 45, 85
control word description 214
conditional processing 99
conditional sections 102
.IF control word 99
of input line 241
overview of 10
special techniques 100
with symbols 104
CONT parameter
of .DC control word 82, 217
contents, table of
adding lines to the 74
building one automatically 73
control word description 285
entries generated by head levels 71-72
file to generate automatic 265
overview of 9
printing with output document 74
using TWOPASS option with 75
continuation character 81
CONTINUE option 18, 299
diagnostic aid 163
control blocks, displaying 170
control word 344
control word line 344
control word line execution 227
control word separator 48
redefining the 48, 81
control words
.AP [Append] 203, 301
assist debugging 165
.BC [Balance Columns] 204, 301
.BF [Begin Font] 204, 301
.BM [Bottom Margin] 205, 301
.BR [Break] 206, 301
.BT [Bottom Title] 206, 301
.BX [Box] 207, 301
.CB [Column Begin] 209, 301
.CC [Conditional Column Begin] 210, 301
.CD [Column Definition] 210, 301
.CE [Center] 211, 301
changes for compatibility 178
.CL [Column Width] 212, 302
.CM [Comment] 212, 302
.CO [Concatenate Mode] 213, 302
.CP [Conditional Page Eject] 214, 302
.CS [Conditional Section] 214, 302
.CW [Control Word Separator] 215, 302
.DC [Define Character] 217, 302
.DD [Define Data File-id] 219, 302
defaults 4
.DH [Define Head Level] 221, 302
.DI [Delay Imbed] 222, 302
directly entered 155
.DM [Define Macro] 223, 303
.DS [Double Space Mode] 225, 303
.DU [Dictionary Update] 226, 303
.EB [Even Page Bottom Title] 227, 303
.EC [Execute Control] 227, 303
.EF [End of File] 228, 303
.EM [Execute Macro] 229, 303
.EP [Even Page Eject] 230, 303
.ET [Even Page Top Title] 230, 303
.EZ [EasySCRIPT] 231, 303

.FM [Footing Margin] 232, 303
.FN [Footnote] 233, 303
.FO [Format Mode] 234, 304
.FS [Footing Space] 235, 304
and GML tags 29
.GO [Goto] 236, 304
grouping the 48
.HE [Heading] 304
.HM [Heading Margin] 237, 304
.HN [Headnote] 237, 304
.HS [Heading Space] 238, 304
.HW [Hyphenate Word] 239, 304
.HY [Hyphenate] 239, 304
.Hn [Head Level n] 240, 304
.IF [If] 241, 305
.IL [Indent Line] 243, 305
.IM [Imbed] 244, 305
.IN [Indent] 245, 305
initial settings 4
.IR [Indent Right] 246, 305
.IT [Input Trace] 246, 305
.JU [Justify Mode] 248, 305
.KP [Keep] 248, 305
.LB [Leading Blank] 250, 305
.LI [Literal] 251, 305
.LL [Line Length] 252, 305
.LS [Line Spacing] 252, 306
.LT [Leading Tab] 253, 306
.LY [Library] 253, 306
.MC [Multicolumn Mode] 254, 306
.MG [Message] 255, 306
modifiers 199
.MS [Macro Substitution] 256, 306
.NL [Null Line] 256, 306
not allowed in keeps 83
notational conventions 200
.OB [Odd Page Bottom Title] 257, 306
.OC [Output Comment] 257, 306
.OF [Offset] 258, 306
.OP [Odd Page Eject] 259, 306
.OT [Odd Page Top Title] 259, 307
.PA [Page Eject] 260, 307
parameters of 3
.PF [Previous Font] 261, 307
.PL [Page Length] 261, 307
.PN [Page Numbering Mode] 262, 307
.PP [Paragraph Start] 264, 307
.PS [Page Number Symbol] 264, 307
.PT [Put Table of Contents] 265, 307
.QQ [Quick Quit] 266, 307
.QU [Quit] 266, 307
.RC [Revision Code] 267, 307
.RD [Read Terminal] 268, 307
.RE [Restore Status] 268, 308
redefining 140
.RF [Running Footing] 269, 308
.RH [Running Heading] 270, 308
.RI [Right Adjust] 271, 308
.RT [Running Title] 272, 308
.RV [Read Variable] 273, 308
.SA [Save Status] 274, 308
.SC [Single Column Mode] 274, 308
.SE [Set Symbol] 275, 308
Set Label 202, 301
separating multiple 215
separator 48, 215
.SF [Save Font] 278, 308
.SK [Skip] 278, 308
.SL [Set Line Space] 279, 308
.SP [Space] 279, 309
space units 200
.SS [Single Space Mode] 280, 309
.SU [Substitute Symbol] 280, 309
.SV [Spelling Verification] 281, 309
.SX [Split Text] 282, 309
.SY [System Command] 283, 309
syntax 199
.TB [Tab Setting] 284, 309
.TC [Table of Contents] 285, 309
.TE [Terminal Input] 286, 309
testing sequence 169
that cause breaks 47
.TI [Translate Input] 287, 309
.TM [Top Margin] 287, 310
.TR [Translate Character] 288, 310
.TT [Top Title] 289, 310
.TY [Type on Terminal] 290, 310
type 1 199
.UC [Underscore and Capitalize] 290, 310
.UD [Underscore Definition] 291, 310
.UN [Undent] 292, 310
.UP [Uppercase] 293, 310
.US [Underscore] 294, 310
.WF [Write To File] 295, 310
when to use 7
.ZZ [Diagnostic] 296, 310
control word
modifier 199
separator control word
redefining 48
description 215
(see also .DC control word)
control words
that case a break 47, 311
that take effect on next page 311
initial values based on logical output device 313
within
a footnote 312
a footing 312
a heading 312
a keep 312
a running footing 312
a running heading 312
control left reference numbering 192
control message printing
MESSAGE option 22
control values, symbols for 128
conventions
implicit SCRIPT/VS formatting 31
conversion, non-format command 188
conversion program operation 188
conversion technique for ATMS-II 187
converting ATMS documents 11, 187
counter, setting the heading 173
counters 193
.CP [Conditional Page Eject] control word
definition 45, 81
description 214
creating a customized letter 111
creating a TSO/FORMAT compatible environment 197
creating your own profiles 145
cross-references to EasySCRIPT headings 173
.CS [Conditional Section] control word
definition 99, 102
description 214
nesting 103
current column definition
saving 274
current font identification
saving 278
current left margin 35, 344

current line
  glossary 345
  vertical space to baseline 279
current page number, setting 131
current page title 289
CTL parameter
  of .IT control word
    changing 167
    description 245
customizing for mass mailing 111
customizing your keyboard 81
.CW [Control Word Separator] control word
  definition 48
  description 215
  (see also .DC control word)
CW parameter
  of the .DC control word
    definition 81
    description 217

D

data set identifier (see .DD control word)
date control 194
date, printing system 127
.DC [Define Character] control word
  definition 81
  description 217
  used in conditional processing 101
  (see also .PS control word)
.DD [Define Data File-id] control word
  defining DSMUTWTF 109
  description 219
  used when appending file 107
  used when imbedding file 107
DD parameter
  of .DD control word 219
debugging
  by tracing processing actions
    overview of 11
  glossary 345
  with the SCRIPT command 163
  your GML macros 150
decimal page numbering 62
default value 345
default values
  for logical output devices 325
define character control word 217
  used in conditional processing 101
define data file-id control word
  defining DSMUTWTF 109
  description 219
  used when appending a file 107
  used when imbedding a file 107
define head level control word
  changing EasySCRIPT defaults 173
  definition 73
  description 221
define macro control word
  definition 137
  description 223
  redefining SCRIPT/VS control words 139
  substituting values for symbols 141
  (see also .MS control word)
defining
  a box 87
  characters to be underlined 44
  a head level 73, 221
  macro definitions 223
  special characters
    that affect SCRIPT/VS 81
    that the formatter recognizes 217
  special text characters 77-80
defining a page layout 51
  basic page parameters 51
  changing line length 53
  changing the page length 53
  page numbering 62
  running headings and footings 59
    page numbering 61
  top and bottom running titles 54
    allocating space for 57
    defaults 59
    defining 62
    multiline running titles 55
DEL parameter
  of .DU control word 226
DELAY parameter
  of .KP control word 83, 248
delayed keeps 83, 248
delaying imbedding of text 85
  control word description 222
  storing lines in DSMUTDIM 110
delimiter characters
  for a sentance 81
  for a symbol 119
  for a word 82
DEST option 19, 299
destination of output, specifying the
  and the logical output device 17
  overview of 11
device and font table maintenance 323
DEVICE option 19, 299
.DH [Define Head Level] control word
  changing EasySCRIPT defaults 173
  description 221
.DI [Delay Imbed] control word
  definition 110
  description 222
  storing input in DSMUTDIM 110
diagnostic aids 163
diagnostic control word
  description 296
  enabling the dump option 163
  (see also DUMP option of SCRIPT command)
dictionary 345
dictionary update
  changes for compatability 181
  control word 226
  hyphenating 158
  verifying spelling 158
Didot point system 345
displaying control blocks 170
displaying output at terminal
  TERM option 27
displaying sequence of SCRIPT/VS processing 165
.DM [Define Macro] control word
  definition 137
  description 223
  substituting values for symbols 141
  redefining SCRIPT/VS control words 140
  (see also .MS control word)
document 345
  attributes, identifying 145
  conversion processor 345
  element
  glossary 345

handling functions 10
identifying document attributes 145
library 345
marking for SCRIPT/370 10
profile
creating your own 145
defining primary attributes 146
role of 145
two formatting passes
TWOPASS option 27
double space mode control word
canceling 280
definition 41
description 225
doublespaced output lines 41, 225
canceling 280
drawing lines
characters to be underlined 44
underlining words 44
.DS [Double Space Mode] control word
canceling 280
definition 41
description 225
DSMTERMI, terminal input file 299
DSMTERMO, terminal output file 290, 299
DSMUTDIM, storing delayed, imbedded input text 110, 299
DSMUTMSG, error messages 299
DSMUTTOC, table of contents 299
DSMUTWTF, writing into file 299
writing to output file id 295
inserting lines into a file 109
DSN parameter
of .DD control word 219
.DU [Dictionary Update] control word
description 226
DUMP option 19, 299
diagnostic aid 163
(see also .ZZ control word)
DUMP parameter
of .ZZ control word 296
diagnostic aid 163
dumping control blocks 296
duplex 345

E

EasySCRIPT 171
bullets 175
control word description 231
cross-references to headings 173
examples of formatting 174
formats 172
heading defaults 173
headings 172
item numbering, automatic 174
lists, unnumbered 175
paragraphs 174
setting the heading counter 173
tables of contents 175
tags 171
.EB [Even Page Bottom Title] control word
description 227
EBCDIC 345
.EC [Execute Control] control word
description 227
edit 345
editor 345
.EF [End of File] control word
description 228
terminate formatting file 110
used with imbedded files 111
eject
column 69, 85
for head levels 72
glossary 345
page 45, 85, 214
specifying odd or even page 46
em 345
.EM [Execute Macro] control word
description 229
redefining SCRIPT/VS control words 139
enable the .SV control word 25
enable the .ZZ control word 19
end of embedded control 188
end of file control word 228
terminate formatting file 110
used with imbedded files 111
entering
control words, guidelines for 46
SPIE exit routines
preventing: NOSPIE option 23
text, guidelines for 46
.EP [Even Page Eject] control word
description 230
ERASE parameter
of .WF control word 295
error messages
control information in 163
printing with MESSAGE option 22
errors, continue processing after
CONTINUE option 18
.ET [Even Page Top Title] control word
description 230
even numbered pages, causing 230
even page bottom title control word 227
even page eject control word 230
even page top title control word 230
EVEN parameter
of .IF control word 241
of .PA control word 46, 260
of .RF control word 59, 269
of .RH control word 59, 270
of .RT control word 54, 272
execute control control word 227
execute macro control word
description 229
redefining SCRIPT/VS control words 139
executing
CMS commands during SCRIPT/VS processing 114
CP commands during SCRIPT/VS processing 114
line as control word line 227
line as macro line 229
with TSO 115
explicit paragraphing specification 189
EXTEND parameter
of .FO control word 31, 234
extended symbol processing 133, 345
.EZ [EasySCRIPT] control word
definition 231
description 231

F

"F" parameter
of .SX control word 282
figures, numbering 131

figure number prefixes 133
figure number suffixes 133
file
  append 105
  end of 228
  imbed 105, 244
  that contains options
   OPTIONS option 23
FILE option 19, 299
file-id associated with file or data
 set identifier 219
fill characters
  between split text 44
  between tab positions 33
  glossary 345
float 345
FLOAT parameter
  of .KP control word 83, 249
floating keeps 83, 249
floating skip 191
flush 345
.FM [Footing Margin] control word
  definition 57
  description 232
  glossary 345
.FN [Footnote] control word
  definition 84
  description 233
  glossary 345
.FO [Format Mode] control word
  definition 29
  description 234
  glossary 346
  (see also .JU control word)
FOLD parameter
  of .FO control word 31, 234
font 345
font management 337
FONT parameter
  of .DH control word 221
font, previous 261
font set 345
font table
  field descriptions 326
  maintenance 325
  updating 325
font width table example 326
fonts (3800 Printer only) 6
  beginning new fonts 204
  highlight fonts 320, 329
  provided with SCRIPT/VS 320, 327
  saving current font 278
  special fonts 320, 329
  specifying with CHARS option 17
  supplied with 3800
   Printer 321, 327
  text fonts 320, 329
  using with IBM 3800 Printer 94
footing 345
footing margin 58
  description 232
  glossary 345
footing space 58
  description 235
  glossary 345
footings, running 59
  ATMS-II conversion 190
  definition of 59
    where to put 62
  description 269
  formatting environment 97
  glossary 348
  line length 252
  page numbers in 61, 264
footnote 345
footnote control word 232
footnote environment, the 98
footnotes 84
  leader 84
  overview of 9
forcing a page eject 45
  conditional 214
  .PA control word 260
  even page 230
foreground 345
format 346
format mode 29
  control word description 234
  glossary 346
  (see also .JU control word)
formatted text mode 189
formatter 346
formatter escape character 341
formatting a document 13
  control conversion 189
  dictionary update 226
  terminating 110
  using EasySCRIPT tags 171
  with two formatting passes
   TWOPASS option 27, 164
formatting control conversion from
 ATMS-II
  comment control 192
  control left reference numbers 192
  counters 193
  date control 194
  explicit paragraphing
   specification 189
  floating skip 191
  formatted text mode 189
  headings and footings 190
  hyphenation control 192
  implicit paragraphing 190
  include floating keeps 192
  justification 191
  keep text 190
  line spacing 191
  overstrike 192
  page definition 192
  page margin control 191
  page number control 193
  paragraph numbering 191
  revision markers 193
  skip lines conditionally 190
  start new page 192
  stop code 193
  text alignment controls 191
  text block indention 192
  text line indention 192
  text split 193
  triplets and backspaces 194
  unconditional skip 191
  unformattable center text 190
  unformatted text mode 189
  uppercase control 194
  width and depth controls 191
  widow zone control 192
formatting considerations
  for 3800 Printer 337
formatting controls
  conversion 189
  defined by macros 137
formatting conventions
  redefining SCRIPT/VS functions 142
formatting environment, the
 SCRIPT/VS 97
  restoring the current 98
  saving the current 98
formatting, EasySCRIPT examples 174
formatting features of SCRIPT/VS 77

formatting functions
basic 29
implicit conventions 31
overview of 8
formatting mode 346
formatting text in a box 87
FRAC parameter
of .PN control word 62, 262
front matter 346
.FS [Footing Space] control word
definition 57
description 235
glossary 345
FTB (see font table)
full stop 31
defining characters for 82

G

GDOCPROF document profile 145
primary document attributes 146
generalized markup language (GML)
tags 3
and control words 29
glossary 346
glossary 343
GML 346
GML, filename default for symbol library 125
GML, implementing with EasySCRIPT 171
GML macros
debugging 150
for attribute processing 149
GML markup and control words 29
GML parameter
of .DC control word 217
GML starter set
defined 145
macros for attribute processing 149
GML support in SCRIPT/VS 145
GML tags
defined by macros 137
for text items 146
processed as symbols 120
to APF mapping 146
.GO [Goto] control word 236
(see also Set Label control word)
Gothic
highlight SCRIPT/VS fonts 332
special purpose SCRIPT/VS fonts 334
text SCRIPT/VS fonts 330
goto control word 236
graphic effects, special 156
grouping SCRIPT/VS control words 48
guidelines for entering text 46
gutter 346

H

handling directly entered control words 155
hanging indentation 37
example of 38
glossary 346
head levels 71-73
characteristics of 71
defaults 173
defining 73, 221
description 240
glossary 346
macros for 73, 223
overview of 8
page ejects 72
setting heading counter 173
spacing 72
heading 346
heading margin 57
control word description 237
glossary 346
heading space 57
control word description 238
glossary 346
headings, running 59
ATMS-II conversion 190
counter, setting 173
defaults in EasySCRIPT 173
definition of 59
where to put 62
.DH control word 221
formatted by EasySCRIPT 172
formatting environment 97
.Hn control word 240
line length 252
numbered automatically 172
on page one 62
page numbers in 61, 264
.RH control word 270
setting heading counter 173
headings and footings 190
headnote control word 237
hexadecimal 346
highlight fonts provided with SCRIPT/VS 320, 329
highlighted phrases 9
.HM [Heading Margin] control word
definition 57
description 237
glossary 346
.HN [Headnote] control word
description 237
(see also .RH control word)
.Hn [Head Level n] control word
description 240
(see also head levels)
(see also .TC control word)
horizontal row, boxes in a 92
horizontal space units 5
how
automatic hyphenation works 157
SCRIPT/VS works 3
to define a macro 137
to substitute values for symbols 141
.HS [Heading Space] control word
definition 57
description 238
glossary 346
.HW [Hyphenate Word] control word
definition 157
description 239
.HY [Hyphenate] control word
definition 157
description 239
hyphenate control word 157, 239
hyphenate word control word 157, 239
hyphenation
altering the parameters 157
automatic 157, 239
control 192
.DU control word 226
of words 188
overview of 9
single occurrence of a word 157, 239

.H0 [Head Level 0] control word
  definition 71
  description 240
  overriding defaults 221
.H1 [Head Level 1] control word
  definition 71
  description 240
.H2 [Head Level 2] control word
  definition 72
  description 240
.H3 [Head Level 3] control word
  definition 72
  description 240
.H4 [Head Level 4] control word
  definition 72
  description 240
.H5 [Head Level 5] control word
  definition 72
  description 240
.H6 [Head Level 6] control word
  definition 72
  description 240

I

identifier message, suppressing
  QUIET option 26
identifying
  lines of file or macro 202
  updated material, overview 10
.IF control word
  definition 99
  description 241
  terminal output characters 81
IGNORE parameter
  of .CS control word
    definition 102, 103
    description 214
.IL [Indent Line] control word
  definition 37
  description 243
.IM [Imbed] control word
  customizing a letter 111
  description 244
  merging documents from several sources 111
IMBED parameter
  of .WF control word
    definition 105
    description 295
imbedding separate files
  customizing a letter 111
  delayed 85, 110
  example of 86
  .IF control word 99
  .IM control word 105, 244
  merging documents from several sources 111
  naming file to be imbedded 106
  overview of 10
  symbols set when 129
  terminating processing 110
implicit paragraphing
 specification 190
implicit SCRIPT/VS formatting conventions 31
.IN [Indent] control word
  definition 35
  description 245
  footnote environment 98
  include floating keeps 192
INCLUDE parameter
  of .CS control word 102, 214
indent
  control word 245
  footnote environment 98
  glossary 346
  line control word 37, 243
  right control word 37, 246
  simple 36
  single line 37
  with tabs 39
indention
  control words in combination 39
  gossary 346
indenting text 35
  all but first line of block 258
  at the left margin 245
  at the right margin 37, 246
  example of 39
  for the next line only 37
  using tabs with 39
  (see also .UN control word)
index counter of array
  accessing 135
  setting 135
INDEX parameter
  of .SE control word 275
  locating symbols in document 118
index to SCRIPT/VS Summary 297
inhibiting substitution of symbols 121
initial caps 346
initial value 346
inline keeps 82, 248
inline space management 339
INLINE parameter
  of .KP control word 85, 248
implicit paragraphing 190
include floating keeps 192
input character translation 78
input device 346
input file
  characteristics 4
  entering control words while processing 286
  entering text while processing 286
input lines
  beginning with a blank or tab 142
  cancel concatenation of 234
  capitalizing, automatically 293
  dynamically put into file 108
  glossary 346
  indenting next (undent) 292
  leading blanks 250
  leading tabs 253
  preventing concatenation of 206
  print without formatting 165
  processed as text 251
  restore concatenation of 234
  saving for subsequent processing 10
  shifting next (undent) 292
  start in column one 47
  substituting values for symbol names 119
  trace information about 246
  translating characters in 79
  underscoring, automatically 294
  unresolved symbols 120
input stream, format with SCRIPT/VS 153
input substitution trace
  capabilities of 166
  control word 246
  displaying sequence of SCRIPT/VS processing 165
  output line generated by 165

stepping through input trace 167
input text, translating 287
input trace control word
capabilities of 166
description 246
displaying sequence of SCRIPT/VS processing 165
stepping through input trace 167
insert file (see .AP control word)
integer page numbering 63
interactive 346
interactive environment
calling SCRIPT/VS in 7
glossary 346
interactive processing during formatting
executing with TSO 115
executing with VS/370 114
overview of 11
using SCRIPT/VS 112
interword space 338
introduction to SCRIPT/VS 1
.IR [Indent Right] control word
definition 37
description 246
.IT [Input Trace] control word
capabilities of 166
description 246
displaying sequence of SCRIPT/VS processing 165
stepping through input trace 167
italic 346

J

JCL 346
job control language 346
.JU [Justify Mode] control word
definition 30
description 248
(see also .FO control word)
justify 346
justify mode control word 30, 248
justification 30
alignment error 341
formatting control conversion 191
of output lines 248
(see also .FO control word)
justify mode 30, 248

K

keep 347
keep control word 82, 248
keep environment, the 97
keep text 190
keeping text together 82-84
conditional column ejects 81
conditional page ejects 81
control words not allowed 83
.KP control word 248
overview of 10
using 83
keyboard, customizing your 81
.KP [Keep] control word
definition 82
description 248

L

layout of an output page 52
glossary 347
multicolumn 65
.LB [Leading Blank] control word
definition 31, 47
description 250
LDT (logical device table) 323
leader 347
LEADER parameter
of .FN control word 84, 233
leading blank 31, 47
control word description 250
leading tab 31, 47
control word description 253
left-hand page 347
left margin 35, 245
LEFT parameter
of .FO control word 30, 234
length
of output page 53, 261
of output column line (width) 65
of output line (page width) 53
.LI [Literal] control word
definition 47
description 251
used with macros from the starter set 149
LIB option 21, 177
symbol substitution 125
(see also .LY control word)
LIB parameter
of .DD control word 219
of .DM control word 223
of .SE control word 275
library control word
description 253
searching for unresolved macros 143
used in symbol substitution 126
libraries, symbol and macro
and the LIB option 21
and the SEARCH option 26
symbol definitions 125
line length 53
control word description 252
line spacing
between output lines 41
canceling 280
control word description 252
formatting control conversion 191
glossary 347
positioning lines on page 42
with blank lines 41
(see also .DS control word)
(see also .SS control word)
line spacing control word 252
lines, blank 41
list of
attributes of a symbol's value 318
characters
marking ending of word 319
marking beginning of word 319
underscored by default 319

control words
not allowed 312
processed once in footing 312
processed once in heading 312
take effect on next output page 311
to break input lines 311
values based on logical default device 313
fonts provided with SCRIPT/VS for using with the 3800 Printer 320
fonts provided with 3800 Printer 321
illustrations, numbering pages 133
utility files that SCRIPT/VS creates or uses 299
lists, unnumbered 175
literal control word
definition 47
description 251
with macros from starter set 149
.LL [Line Length] control word
definition 53
description 252
local symbols for macros 138
logical device characteristics 20, 313
logical device table 323
logical device table field descriptions 323
logical output device 4
and output destination 17
special values for 325
specify with DEVICE option 19
lowercase 347
.LS [Line Spacing] control word
between output lines 42
canceling 280
description 252
formatting control conversion 191
glossary 347
with blank lines 42
(see also .DS control word)
(See also .SS control word)
.LT [Leading Tab] control word
description 253
.LY [Library] control word
description 253
searching for unresolved macros 143
used in symbol substitution 126

M

MAC parameter
of .IT control word 246
of .LY control word 253
machine-readable 347
macro 347
macro calls
cancel automatic 256
initiate automatic 256
macro definitions
.DM control word 223
retrieving from a library 253
substituting values for symbols 141
macro instructions
defining 137
defining head levels with 73
for secondary attribute processing 148
invoking 199
local symbols for 138
messages in 170
naming conventions 138
overview of 10
using 137
within starter set APFs 148
writing 137
macro libraries
convert ATMS-II documents to SCRIPT/VS format 187
identify with LIB option 21
identify with SEARCH option 26
specify 143
macro substitution
avoiding an endless loop 140
control word description 256
glossary 347
redefining SCRIPT/VS control words 140
(see also .EC control word)
macro processing, symbols set 130
main dictionary hyphenation 158
margin
bottom 51, 205
center output lines between 211
current left 35
changing the 36
footing 232
glossary 347
left, indenting 245
right 36
aligning text 42, 271
indenting 246
top 51
margins
splitting text between 43
fill characters for 44
marking line for reference in a .GO 202
marking updated material 86
mark up 347
markup 347
master document fragment 189
master file
structure using imbeds 105
used with imbedded files 107
.MC [Multicolumn Mode] control word
definition 70
description 254
(see also .SC control word)
merging documents from several sources 111
MESSAGE option 22, 299
diagnostic aid 163
message, writing
control word description 255
in macros 170
.MG [Message] control word 255
middle portion of a box
printing by itself 93
within a larger box 92
MINPT paramter
of .HY control word 239
modifying
head-level definition 73
unpredictable processing results 156
.MS [Macro Substitution] control word
avoiding an endless loop 141
description 256
glossary 347
redefining SCRIPT/VS control words 139
(see also .EC control word)
multiple column formatting

cancel column balance 204
restore column balance 204
multiple spacing output text 252
multicolumn mode control word 254
(see also .SC control word)
multicolumn processing 65
beginning a new column 69
column balancing 69
defining a multicolumn layout 65
an example of 68
page section breaks 66
restoring 254
section breaks 66
suspending and resuming 69
multiline running titles 55
allocating space for 57
mutually exclusive SCRIPT command options 16

N

name a disk file for output
FILE option 19
naming the output file 20
name a file containing options
OPTIONS option 23
name a remote output station
DEST option 19
naming conventions
for macros 138
naming the input file 13
NBR parameter
of .DH control word 221
new page (see .PA control word)
NEW parameter
of .BX control word 207
new column, starting a 69
.NL [Null Line] control word
description 256
processing empty input lines 142
NOADD parameter
of .HY control word 239
of .SV control word 281
NOJ parameter
of .DH control word 221
non-format command conversion 188
NOPROF option 23, 299
NORM parameter
of .PN control word 63, 262
normal page numbering 61, 263
NOSPIE option 23, 299
diagnostic aid 164
NOSTART parameter
of .PA control word 260
NOSTEM parameter
of .SV control word 281
notational conventions 200
notes (as footnotes) 84
NOWAIT option 23, 299
NPA parameter
of .DH control word 221
NTC parameter
of .DH control word 221
NTO parameter
of .DH control word 221
NTS parameter
of .DH control word 221
null value for symbols 122, 139
null line 32
control word description 256
in conditional processing 101
processing empty input lines 142
NUM parameter
of .SV control word 281
NUMBER option 23, 299
diagnostic aid 164
numbering figures 131
numbering pages 61
NUP parameter
of .DH control word 221
NUS parameter
of .DH control word 221

O

.OB [Odd Page Bottom Title] control word
description 257
.OC [Output Comment] control word
description 257
ODD parameter
of .IF control word 241
of .PA control word 46, 260
of .RF control word 59, 269
of .RH control word 59, 270
of .RT control word 54, 272
odd page bottom title control word 257
odd page eject control word 259
odd page top title control word 259
.OF [Offset] control word
definition 38
description 258
glossary 347
OFF parameter
of .BX control word 207
of .DC control word 217
of .DM control word 223
of .SE control word 275
OFFNO parameter
of .PN control word 262
offset 38
control word description 258
example of 39
glossary 347
text 38
OJ parameter
of .DH control word 221
ON/OFF parameter
of .RC control word 267
ON and OFF parameters
of .BC control word 204
of .CE control word 211
of .CO control word 213
of .CS control word 214
of .DI control word 222
of .EP control word 230
of .EZ control word 231
of .FN control word 233
of .FO control word 234
of .HY control word 239
of .IT control word 246
of .JU control word 248
of .KP control word 248
of .LI control word 251
of .LY control word 253
of .MS control word 256
of .OP control word 259
of .PA control word 260
of .PN control word 262
of .RC control word 267
of .RF control word 269
of .RH control word 270
of .RI control word 271
of .SU control word 280
of .SV control word 281

of .TE control word 286
of .UC control word 290
of .UD control word 291
of .UP control word 293
of .US control word 294
of .WF control word 295
of .ZZ control word 296
.OP [Odd Page Eject] control word
description 259
option 347
OPTIONS option 23, 299
options of the SCRIPT command
name a file that contains
OPTIONS option 23
summary of 15
used with TSO 16
.OT [Odd Page Top Title] control word
description 259
output comment control word 257
output device 347
output document
destination of 11
and the logical device 17
DEVICE option 19
glossary 347
print it: PRINT option 25
print part: PAGE option 24
overview of 9
print table of contents 74
saved in a sequential file
FILE option 19
output file
write to file control word 295
writing to 108
output lines
column justification 234
double spacing 225
.FO control word 234
generated by input tracing 165
glossary 347
.JU control word 248
multiple spacing 252
right adjusting 271
single space 280
vertical spacing before next 279
output page
bottom margin 205
bottom title 206
column begin 209
column definition 210
column width 212
conditional column begin 209
double spacing 225
eject even page 230
even page bottom title 227
even page top title 230
formatting environment 98
layout of 52
left margin 245
length (depth) 261
multicolumn 65, 210
right margin 246
running title 272
system date and time 126
top margin 287
top title 289
overriding head levels 221
overstrike 192

P

"P" parameter
of .SK control word 278
of .SP control word 279
.PA [Page Eject] control word
definition 45
description 260
PA parameter
of .DH control word 221
page
breaks 66
definition 192
dimentions, basic 51
length 53
lines on 42
margin control 191
number control 193
numbering 62
setting with symbols 131
with prefixes 63
sections 66
selectively print
PAGE option 24
width 53, 252
page eject
conditional control word
description 214
control word description 45, 258
delay portion of file 222
even page 46, 230
for head levels 72
forcing new page 45
odd page 46, 257
terminate processing 266
page eject mode 46
page image, shift to right
BIND option 17
page layout
defining a 51
functions, overview of 8
multicolumn 65
beginning a new column 69
example of 66
suspending and resuming 69
parameters 51
layout picture 298
page length control word
definition 53
description 261
page number symbol
definition 62
description 264
in headings and footings 61
redefining the 81
page numbering mode control word
definition 62
description 262
PAGE option 24, 300
diagnostic aid 164
paginate 347
paper adjustment, prevent prompting
NOWAIT option 23
paragraph start control word
definition 174
description 264
explicit specification 189
implicit specification 190
paragraphing
explicit specification 189
implicit specification 190
numbering 191
.PP control word 264
specified in EasySCRIPT 174
parameters
altering the hyphenation 157
define formatting environment 97
of control words 3
passing to input files 129

parts of a box 87
passing parameters to input files 129
pel 347
period, guidelines for use 47
.PF [Previous Font] control word
  definition 94
  description 261
  (see also .SF control word)
physical output devices 4
pica 347
pitch 347
.PL [Page Length] control word
  definition 53
  description 261
.PN [Page Numbering Mode] control word
  definition 62
  description 262
point 347
positioning text on the page 42
postprocessor, using SCRIPT/VS 153
.PP [Paragraph Start] control word
  definition 174
  description 264
  explicit specification 189
  implicit specification 190
PREF parameter
  of .PN control word 63, 262
prefixes
  for figure numbers 133
  for page numbers 63, 262
  removed from words 160
preprocessor, using SCRIPT/VS 7, 153
  developing APFs and profiles 154
prevent entering SPIE exit routines
  NOSPIE option 23
prevent prompt for paper adjustment
  NOWAIT option 23
previous font control word
  definition 94
  description 261
PRINT option 25, 300
print
  file name and line number
    NUMBER option 23, 164
  input lines without formatting
    UNFORMAT option 28,165
  lowercase letters as uppercase
    UPCASE option 28
  pages selectively
    PAGE option 24, 164
  part of output document 9
  separate pages at terminal
    STOP option 26
  table of contents 74
PRINT parameter
  of .IF control word 241
printers 4
PROC parameter
  of .DD control word 219
processing
  empty input lines 142
  input conditionally 10
  interactively during formatting 11
  lines that begin with a blank 142
  lines that begin with a tab 142
  modifying unpredictable
   results 156
  preparing for 156
  symbols and macros overview 10
produce printer output
  PRINT option 25
PROFILE option 25, 300
profiles 3
  converting ATMS-II documents to
   SCRIPT/VS format 187
  developing preprocessor 154
  glossary 347
  specifying: PROFILE option 25
  suppressing: NOPROF option 23
proportional spacing 348
.PS [Page Number Symbol] control word
  definition 621
  description 264
  in headings and footings 61
  redefining the 81
PS parameter
  of the .DC control word
    definition 81
    description 217
.PT [Put Table of Contents] control
 word
  description 265
  (see also .TC control word)
PUNC parameter
  of the .DC control word 81, 217
put table of contents control word
  description 265
  (see also .TC control word)
putting messages in macros 170

Q

.QQ [Quick Quit] control word
  description 266
  terminating formatting of file 110
.QU [Quit] control word
  description 266
  terminating formatting of file 110
quick quit control word
  description 266
  terminating formatting of file 110
QUIET option 26, 300
quit control word
  description 266
  terminating formatting of file 110

R

ragged right 30, 348
RB parameter
  of .DC control word 217
.RC [Revision Code] control word
  definition 86
  description 267
  source document management 155
  (see also source document
   management)
.RD [Read Terminal] control word
  description 268
  interactive SCRIPT/VS
   processing 112
.RE [Restore Status] control word
  description 268
  restoring current formatting
   environment 98
  (see also .SA control word)
read terminal control word
  description 268
  interactive SCRIPT/VS
   processing 112
read variable control word 273
  interactive SCRIPT/VS
   processing 112
  (see also .SE control word)
redefining symbols 155
redefining control word separator 48

regular keeps 83
restore status control word
  description 268
  restoring current formatting environment 98
  (see also .SA control word)
retrieving from a library 253
return code from CMS command 128
return code from CP command 128
revision code characters 339
revision code control word 86
  description 267
  source document management 155
revision codes 86
revision markers 193
.RF [Running Footing] control word
  definition of 59
    where to put 62
  description 269
  glossary 348
  line length 252
  margins 232
  page number symbols 264
  page numbers in 61, 62
  (see also .FS control word)
.RH [Running Heading] control word
  definition of 59
    where to put 62
  description 270
  glossary 348
  line length 252
  margins 236
  page number symbols 264
  page numbers in 61, 62
  (see also .HN and .HS control words)
.RI [Right Adjust] control word
  definition 42
  description 271
right adjust control word 42, 271
right-hand page 348
right justification 30
right margin
  aligning text with the 42
  changing the 36, 245
RIGHT parameter
  of .FO control word 234
role of a document profile 145
roman numeral page numbering 63, 262
ROMAN parameter
  of .PN control word 63, 262
root words for spelling verification 160
.RT [Running Title] control word
  allocating space for 57
  default for 59
  definition of 54
    where to put 62
  description 272
  even page bottom title control word 227
  even page top title control word 230
  glossary 348
  line length 252
  multiline titles 55
    allocating space for 57
  odd page bottom control word 256
  Li2.odd page top control word 258
rule 348
RUN parameter
  of .IT control word 246
running footings
  definition of 59
    where to put 62
  description 269
  glossary 348
  line length 252
  margins 232
  page number symbols 264
  page numbers in 61, 62
  (see also .FS control word)
running headings
  definition of 59
    where to put 62
  description 270
  glossary 348
  line length 252
  margins 236
  on page one 62
  page number symbols 264
  page numbers in 61, 62
  (see also .HN and .HS control words)
running titles
  allocating space for 57
  bottom title control word 206
  default for 59
  definition of 54-59
    where to put 62
  description 272
  even page bottom title control word 227
  even page top title control word 230
  glossary 348
  line length 252
  multiline titles 55
    allocating space for 57
  odd page bottom control word 256
  odd page top control word 258
.RV [Read Variable] control word
  description 273
  interactive SCRIPT/VS processing 112
  (see also .SE control word)

S

.SA [Save Status] control word
  description 274
  saving current formatting environment 98
  (see also .RE control word)
save font control word 95, 278
save status control word 274
  saving current formatting environment 98
  (see also .RE control word)
saving input lines for subsequent processing 10
.SC [Single Column Mode] control word
  definition 69
  description 274
  (see also .MC control word)
SCRIPT command 13
  as diagnostic aid 163
  BIND option 17
  CHARS option 17
  compatability changes to 177
  CONTINUE option 18
  defaults 16
  DEST option 19
  DEVICE option 19
  DUMP option 19
  examples 17
  FILE option 19
  in TSO 197

LIB option 21
MESSAGE option 22
mutually exclusive options 16
NOPROF option 23
NOSPIE option 23
NOWAIT option 23
NUMBER option 23
options 14
OPTIONS option 23
PAGE option 24
PRINT option 25
PROFILE option 25
QUIET option 26
SEARCH option 26
setting symbols with 128
SPELLCHK option 26
STOP option 26
summary of options 15
SYSVAR option 26
TERM option 27
TSO, options with special meaning
DEST option 19
LIB option 21
PRINT option 25
SEARCH option 26
TWOPASS option 27
UNFORMAT option 28
UPCASE option 28
using the 13
SCRIPT options summary 299
SCRIPT/VS
as a preprocessor 7
as a subroutine 7
control values, symbols for 128
control words
changes for compatability 178, 182
compatible with TSO/FORMAT 197
modified by macros 137
selecting 8
when to use 7
dictionary 158
files, combining 105
fonts
Gothic highlight 332
Gothic special purpose 334
Gothic text 330
fonts provided 320, 327
Serif highlight 333
Serif special purpose 335
Serif text 331
formatting environment, the 97
basic 29
saving 274
summary 315
functions 8
implicit formatting conventions 31
input file characteristics 4
logical device characteristics 20, 313
processor
calling the 6
summary 297
system symbols 126, 316
terms for parts of page 298
utility files 299
SCRIPT/370 dictionary 181
.SE [Set Symbol] control word
cancelling a symbol 122
defining primary document attributes 146
description 275
explanation 104
GML tags 83
macros used instead 137
symbols in your document 117
with the SCRIPT command 129
(see also .RV control word)
SEARCH option 26, 300
sections
breaks 66
conditional text 99
glossary 348
page 66
selecting control words 8
selectively print pages
PAGE option 24
semi-colon 48
separating multiple control words 215
separator, control word 48
redefining the 48
separator characters for an array
defining the 81
SEQ parameter
of .DD control word 219, 120
Serif
highlight SCRIPT/VS font 333
special purpose SCRIPT/VS font 335
text SCRIPT/VS font 331
Set Label control word 202
set line space control word
definition 41
description 279
SET parameter
of .HY control word 239
set symbol control word
cancelling a symbol 122
defining primary document attributes 149
definition 104
description 275
macros used instead 137
symbols in your document 117
with the SCRIPT command 129
(see also .RV control word)
setting tabs 32
setting the heading counter 173
.SF [Save Font] control word
definition 95
description 278
shift the page image to right
BIND option 17
simple indentions 36
single column mode control word
definition 69
description 274
single line indention 37
single space mode control word
definition 41
description 280
single space output 280
single words, hyphenating 157
.SK [Skip] control word
definition 40
description 278
SKBF parameter
of .DH control word 221
skip lines conditionally 190
skip lines control word 278
.SL [Set Line Space] control word
definition 41
description 279
some uses for tabs 33
source document management 155
source text, translating characters 288
small caps 348
SNAP parameter
of .IT control word 246
source document 348

.SP [Space] control word
definition 41
description 279
space 348
space lines control word
definition 41
description 279
space management, inline 339
space units 5, 200
spacing
between first line of text and heading 238
between last line of text and footing 235
between output lines 41
for head levels 72
with blank lines 41
SPAF parameter
of .DH control word 221
special characters 77
ATMS-II codes 194
defining 217
for terminal output 81
using symbols for 81
special fonts provided with SCRIPT/VS 320, 327
special symbols
set with the SYSVAR option 26
special techniques for conditional processing 100
specify fonts
CHARS option 17
specify a library
SEARCH option 26
specify a profile
PROFILE option 25
specify symbol and macro libraries
LIB option 21
specifying a macro library 143
SPELLCHK option 26, 300
diagnostic aid 164
spelling verification 158
and SPELLCHK option 26, 164
assist debugging 165
characters to delimit words 319
.DU control word 226
fallibility 161
overview of 9
.SV control word 281
SPIE exit routines, prevent entering
NOSPIE option 23, 164
split text control word 44, 282
splitting text
between the margins 43, 282
string of 282
with running headings and footings 60
.SS [Single Space Mode] control word
definition 41
description 280
(see also .DS and .LS control words)
stacking one box on another 90
start new page 192
starter set
APFs, symbols within 147
developing preprocessor APFs and profiles 154
macros for secondary attribute processing 148
starting
a new column 69
a new page 45
text at top of column 85
text at the top of the page 85
stem processing 160
STEP parameter
of .IT control word 245
stop code 193
STOP option 26, 300
STOP parameter
of .DC control word 217
.SU [Substitute Symbol] control word
description 280
inhibiting 121
SUB parameter
of .IT control word 246
subdocument identifier 188
subroutine, using SCRIPT/VS 7
substitute symbol control word
description 280
inhibiting 121
SUBSTR parameter
of .SE control word 275
defining symbols in document 118
suffixes for figure numbers 133
suffixes removed from words 161
summary of
the head level characteristics 314
the options of the SCRIPT command 299
the parameters saved using .SA control word 315
the SCRIPT/VS control words and parameters 301
the SCRIPT/VS system symbols 316
SUP parameter
of .HY control word 239
suppressing the profile
NOPROF option 23
suspending
and resuming multicolumn processing 69
concatenation with a break 34
.SV [Spelling Verification] control word
assist debugging 165
definition 158
description 281
fallibility 161
.SX [Split Text] control word
definition 43
description 282
fill characters for 44
.SY [System Command] control word
description 283
specifying TSO commands and procedures 115
SYM paramter
of .LY control word 253
symbol 348
symbol and macro libraries 125
symbol libraries
defining symbols 125
identifying with LIB option 21
identifying with SEARCH option 26
macro definitions 125
symbol names 276
how SCRIPT/VS substitutes values 119
SCRIPT/VS system 126
symbol processing 104
extended 133
.EZ control word 231
in your document 117
overview of 10
symbol substitution 348
symbol values 276
attributes of 122, 318
determining current value 123

determining existence 123
determining length of 123
determining the type of 123
symbols
analyzing the type of 122
assigning values to 275
cancelling 122
compound 119
conditional processing with 104
converting
lowercase to uppercase 125
numeric symbol to base-26 number 122
numeric symbol to Roman numeral 124
current page number set 131
defining values 275
extended processing 133
for arrays of values 134
for SCRIPT/VS control values 128
for special characters 79, 81
in document 117
inhibiting substitution 121
local for macros 138
numbering figures 131
page number 264
redefining 155
restrictions when using 130
retrieving from a library 253
returning the current value of 123
separate multiple control words 215
set
when a file is.appended 129
when a file is imbedded 129
when a macro is processed 130
with the SCRIPT command 129
special &$RET 128
substitution 280
system date and time 126
system variable 26
unresolved 120
within starter set APFs 147
(see also .SE control word)
SYSOUT parameter
of .IF control word 241
with .IF control word 100
SYSPAGE parameter
of .IF control word 241
with .IF control word 100
system command control word
description 283
specifying TSO commands 115
specifying TSO procedures 115
system date and time, symbols for 126
system symbols for SCRIPT/VS control values 128
system symbol names 126
SYSVAR option 26, 300

T

tab characters, defining 284
tab setting control word 32
description 284
for ATMS-II conversion 189
tab resolution for 3800 Printer 338
table of contents, automatic generation 175
tabs
fill characters 33
glossary 348
indent with 39
leading 253
overview of 9
setting 32
uses for 33
using 32
with indentation 39
table of contents
adding lines to the 74
building one automatically 73
control word description 285
entries generated by head levels 71-72
file used to generate automatic 265
overview of 9
printing with output document 74
using TWOPASS option with 75
(see also .PT control word)
tags, GML 29, 348
.TB [Tab Setting] control word
definition 32
description 284
for ATMS-II conversion 189
.TC [Table of Contents] control word
description 285
(see also .Hn and .PT control words)
TC parameter
of .DH control word 221
TCIN parameter
of .DH control word 221
.TE [Terminal Input] control word
description 286
interactive SCRIPT/VS processing 112
techniques for conditional processing, special 100
TERM option 27, 300
TERM parameter
of .DD control word 219
of .IF control word 241
terminal
displaying
one line of information 290
output at: TERM option 27
entering a line during SCRIPT/VS processing 268
entry used to test control word sequence 169
glossary 348
output, special characters for 81
print pages at: STOP option 26
read variable control word 273
terminal input control word
description 286
interactive SCRIPT/VS processing 112
terminating formatting of file 110
terminate processing
immediately (see .QQ control word)
with a final page eject (see .QU control word)
text alignment controls 191
text block indention 192
text fonts provided with SCRIPT/VS 320, 327
text formatting
conditional 99
basic 29
SCRIPT/VS 29
text items
GML tags for 146
(see also .LI control word)
text line 348
text line indention 192

text split 193
THRESH parameter
of .HY control word 239
.TI [Translate Input] control word
definition 79, 98
description 287
used to inhibit symbol substitution 121
titles, running
allocating space from bottom margin 235
allocating space to top margin 238
bottom line 206
default for 59
definition 54-59
even page bottom 227
even page top 230
formatting environment 97
line length 252
multiline running titles 55
allocating space for 57
odd page bottom 256
odd page top 257
page number symbol 264
running title 272
saving 206
top 289
where to put definition 62
.TM [Top Margin] control word
definition 53
description 287
TN translate table for 1403 Printer 320
TO parameter
of .DH control word 221
top line of a box
printing it by itself 92
within a running heading 95
top margin 53
control word description 287
glossary 348
TOP parameter
of .RT control word 54, 272
top running titles
default for 59
definition 54-59
top title 348
top title control word 289
.TR [Translate Character] control word
definition 77, 97
description 288
used to inhibit symbol substitution 121
trace information about input lines displayed 246
tracing processing actions
diagnostic aid 165, 166
overview of 11
stepping through an input trace 167
translate character control word
definition 77, 98
description 288
used to inhibit symbol substitution 121
translate input control word
definition 79, 97
description 287
used to inhibit symbol substitution 121
translating characters
on the input line 78
on the output line 81
triplets and backspaces 194
TRUNC parameter
of .FO control word 31, 234
TS parameter
of .DH control word 221
TSO
glossary 348
input file naming conventions 14
system commands 283
TSO/FORMAT
compatibility with 197
creating compatible environment 197
.TT [Top Title] control word
description 289
two formatting passes
TWOPASS option 27, 164
TWOPASS option 27, 300
and the table of contents 75
diagnostic aid 164
.TY [Type on Terminal] control word
description 290
interactive SCRIPT/VS processing 113
type 1 control words 199
type on terminal control word
description 290
interactive SCRIPT/VS processing 113
typeface 348
typeset 348
typestyles and fonts 6

U

.UC [Underscore and Capitalize] control word
definition 44
description 290
.UD [Underscore Definition] control word
definition 44
description 291
.UN [Undent] control word
definition 39
description 292
(see also .IN control word)
unconditional skip 191
undent control word 39, 292
underlining 44
and capitalizing words 44, 290
automatically 294
characters to be 44, 291
underscore 349
underscore control word 44, 293
underscore and capitalize control word
definition 44
description 290
underscore definition control word
characters not underscored by default 319
definition 44
description 291
underscore resolution for 3800 Printer 338
underscoring (see underlining)
UNFORMAT option 28, 300
diagnostic aid 165
unformattable center text 190
unformatted text mode 189, 349
unpredictable processing results, modifying 156
unresolved symbols 120
.UP [Uppercase] control word
definition 44

description 293
UP parameter
of .DH control word 221
UPCASE option 28, 300
updated material, identifying 86
overview of 10
updating a logical device table (LDT) 323
uppercase 349
uppercase control 194
uppercase control word 293
.US [Underscore] control word
definition 44
description 294
US parameter
of .DH control word 221
users of SCRIPT/VS 2
using
GML tags 7
SCRIPT/VS
as a postprocessor 153
as a preprocessor 7, 153
as a subroutine 7
in a batch environment 153
with other programs 153
terminal entry tests control word sequence 169
the SCRIPT command 13

V

verifying spelling
diagnostic aid 164
.DU control word 226
overview of 10
vertical space 40
blank 278, 279
vertical space units 5

W

.WF [Write To File] control word
definition 108
description 295
multiple files 109
when should you use macros? 137
when to use control words 7
who uses SCRIPT/VS? 2
width
of column 68
of page (line length) 53
width and depth controls 191
widow 349
widow zone control 192
WORD parameter
of the .DC control word 82, 217
word space 338, 349
write to file control word 108
description 295
multiple files 109
writing messages 255
writing SCRIPT/VS macro instructions 137

X

Y

Z

.ZZ [Diagnostic] control word
description 296
diagnostic aid 163
(see also DUMP option of the SCRIPT command)

NUMERIC SUBJECTS

3800 Printer formatting considerations 337
3800 Printer fonts 321, 327

Document Composition Facility:
User's Guide
SH20-9161-0

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

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 to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

Reader's Comment Form

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

Fold and Tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Document Composition Facility: User's Guide Printed in U.S.A. SH20-9161-0

SH20-9161-0

Document Composition Facility: User's Guide Printed in U.S.A. SH20-9161-0



International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601