TIBCO FOCUS®

Developing Applications

Release 8207.27.0
March 2021
DN1001057.0321

Copyright © 2021. TIBCO Software Inc. All Rights Reserved.

Contents

1. Customizing Your Environment 23

When Do You Use the SET Command? 23

Coding a SET Command 23

Types of SET Parameters 27

Calculations. 28

Data and Metadata. 29

Date Manipulation Tasks. 33

Graph Tasks. 34

Memory Setup and Optimization Tasks. 36

Report Code, Content, and Processing Tasks. 38

Report Layout and Display Tasks. 42

Security Tasks. 48

Terminal Tasks. 48

SET Parameter Syntax 50

ACCBLN. 54

ACROSSLINE. 54

ACROSSPRT. 54

ACROSSTITLE. 55

ACRSVRBTITL. 56

ALL 57

ALLOWCVTERR. 57

ALTBACKPERLINE. 58

ASNAMES. 59

AUTOFIT. 60

AUTOINDEX. 60

AUTOPATH. 61

AUTOSTRATEGY. 61

AUTOTABLEF. 62

BASEURL. 62

BINS. 62

BLANKINDENT. 63

BOTTOMMARGIN. 64

BUSDAYS. 64
BYDISPLAY. 64
BYPANEL. 65
CACHE. 66
CARTESIAN. 66
CDN. 67
CENT-ZERO. 68
CNOTATION. 68
COLLATION. 69
COMPMISS. 70
COMPOUND. 70
COMPUTE. 71
COUNTWIDTH. 72
CSSURL. 72
CURRENCY_DISPLAY. 73
CURRENCY_ISO_CODE. 73
CURRENCY_PRINT_ISO. 74
CURRSYMB. 74
CURSYM_D. 75
CURSYM_E. 75
CURSYM_F. 76
CURSYM_G. 76
CURSYM_L. 76
CURSYM_Y. 77
DATE_ORDER. 77
DATE_SEPARATOR. 78
DATEDISPLAY. 79
DATEFNS. 79
DATEFORMAT. 80
DATETIME. 80
DB_INFILE. 81
DBACSENSITIV. 81
DBAJOIN. 82

DBASOURCE. 82
DEFCENT. 83
DEFECHO. 84
DEFINES. 84
DIRECTHOLD. 85
DMH_LOOPLIM. 85
DMH_STACKLIM. 85
DMPRECISION. 86
DROPBLNKLINE. 86
DTSTRICT. 87
DUPLICATECOL. 87
EMBEDDABLE. 88
EMPTYREPORT. 88
EQTEST. 89
ERROROUT. 90
ESTRECORDS. 90
EUROFILE. 91
EXCELSERVURL. 91
EXL2KLANG. 91
EXL2KTXTDATE. 92
EXTAGGR. 92
EXTENDNUM. 93
EXTHOLD. 93
EXTRACT. 94
EXTSORT. 94
FIELDNAME. 95
FILE[NAME]. 95
FILTER. 96
FIXRET[RIEVE]. 96
FLOATMAPPING. 97
FOC144. 97
FOCEXURL. 98
FOCFIRSTPAGE. 98

FOCSTACK. 98
FORMULTIPLE. 99
HDAY. 100
HIDENULLACRS. 100
HLDCOM_TRIMANV. 100
HNODATA. 101
HOLDATTR. 102
HOLDFORMAT. 102
HOLDLIST. 103
HOLDMISS. 103
HOLDSTAT. 104
HTMLARCHIVE. 104
HTMLCSS. 105
HTMLEMBEDIMG. 105
HTMLENCODE. 106
INDEX. 106
JOIN_LENGTH_MODE (JOINLM). 107
JOINOPT. 108
KEEPDEFINES. 108
KEEPFILTERS. 109
LANG[UAGE] 110
LAYOUTGRID. 112
LEADZERO. 112
LEFTMARGIN. 112
LINES. 113
MATCHCOLUMNORDER. 113
MAXDATAEXCPT. 114
MAXLRECL. 115
MDICARDWARN. 115
MDIENCODING. 116
MDIPROGRESS. 116
MESSAGE. 117
MISS_ON. 117

MISSINGTEST. 118
MULTIPATH. 119
NODATA. 119
NULL. 120
OLDSTYRECLEN. 120
ONFIELD. 120
ORIENTATION. 121
OVERFLOWCHAR. 121
PAGE[-NUM]. 122
PAGESIZE. 122
PANEL. 124
PARTITION_ON. 125
PASS. 125
PCOMMA. 125
PCTFORMAT. 127
PDFLINETERM. 127
PERMPASS 128
PHONETIC_ALGORITHM. 129
PRFTITLE. 129
PRINT. 129
PRINTDST. 130
PRINTPLUS. 130
PSPAGESETUP. 131
QUALCHAR. 131
QUALTITLES. 132
RANK. 132
RECAP-COUNT. 133
RECORDLIMIT. 133
RIGHTMARGIN. 134
RPAGESET. 134
SAVEDMASTERS. 134
SAVEMATRIX. 135
SHADOW. 135

SHIFT. 136
SHORTPATH. 136
SHOWBLANKS. 137
SORTMATRIX. 138
SORTMEMORY. 138
SPACES. 138
SQLTOPTTF. 139
SQUEEZE. 139
STYLE[SHEET]. 140
SUBTOTALS. 141
SUMMARYLINES. 141
SUMPREFIX. 142
TESTDATE. 142
TIME_SEPARATOR. 143
TITLELINE. 143
TITLES. 143
TOPMARGIN. 144
UNITS. 144
USER. 145
USERFCHK. 145
USERFNS. 146
WARNING. 147
WEEKFIRST. 147
WPMINWIDTH. 148
XLSXPAGEBRKIGNORE. 149
XRETRIEVAL. 149
YRTHRESH. 149

2. Managing Applications 151

What Is an Application? 151
Application Commands Overview 153
Search Path Management Commands 156
APP PATH. 156

APP PREPENDPATH. 157
APP APPENDPATH. 158
APP MAP. 159
APP SET METALOCATION_SAME. 161
APP ? METALOCATION_SAME. 161
APP SHOWPATH. 161
Application and File Management Commands 162
APP CREATE. 162
APP COPY. 164
APP COPYF[ILE]. 165
APP MOVE. 166
APP MOVEF[ILE]. 167
APP DELETE. 168
APP DELETEF[ILE]. 168
APP PROPERTY CODEPAGE. 169
APP RENAME. 169
APP RENAMEF[ILE]. 170
Designating File Types for APP Commands. 171
Output Redirection Commands 175
APP HOLD. 177
APP HOLDDATA. 178
APP HOLDMETA. 178
APP FI[LEDEF]. 179
Application Metadata Commands and Catalog Metadata 179
Retrieving Basic Information. 179
STATE. 180
APP LIST. 181
APP QUERY. 182
Retrieving Extended Catalog Information. 184
catalog/sysapps. 184
catalog/sysfiles. 185
APP HELP 187
Accessing Metadata and Procedures 187

Search Rules. 187
Creation Rules for Procedure Files. 188
Locating Master Files and Procedures. 188
Accessing Existing Data Files. 190
Creation Rules for Data Files. 190
Data Set Names. 195
Allocating Temporary Files 196

3. Managing Flow of Control in an Application 201

Uses for Dialogue Manager 201
Dialogue Manager Variables Overview. 204
Dialogue Manager Processing 205
Creating a Procedure 208
Including Comments in a Procedure. 209
Sending a Message to the User. 210
Controlling User Access to Data. 211
Creating a Startup Procedure. 212
Executing and Terminating a Procedure 213
Executing Procedures. 213
Executing Stacked Commands and Continuing the Procedure. 214
Executing Stacked Commands and Exiting the Procedure. 215
Canceling the Execution of a Procedure. 216
Locking Procedure Users Out of FOCUS. 217
Navigating a Procedure 217
Branching Unconditionally. 218
Branching Conditionally. 219
Looping in a Procedure. 222
Incorporating Another Procedure With -INCLUDE. 225
Nesting Procedures With -INCLUDE. 228
Calling Another Procedure With EXEC. 228
Developing an Open-Ended Procedure. 229
Using Variables in a Procedure 230
Local Variables. 232

Global Variables. 233
System Variables. 234
Statistical Variables. 242
Special Variables. 245
Querying the Values of Variables and Parameters. 245
Supplying and Verifying Values for Variables 247
Supplying a Default Variable Value. 250
Supplying Variable Values in an Expression. 251
Reading Variable Values From and Writing Variable Values to an External File. 256
Reading or Writing an Entire File. 263
EDAGET: Reading a File of a Specified Type. 263
EDAPUT: Writing a File of a Specified Type. 265
Supplying Variable Values on the Command Line. 267
Prompting Directly for Values With -PROMPT. 270
Prompting for Values on Screens With -CRTFORM. 271
Prompting for Values on Menus and Windows With -WINDOW. 271
Prompting for Values Implicitly. 271
Verifying User-Supplied Values Against a Set of Format Specifications. 272
Verifying User Input Against a Pre-Defined List of Values. 273
Manipulating and Testing Variables 275
Testing Variables for Length, Type, and Existence. 276
Replacing a Variable Immediately. 279
Validating Variable Values Without Data File Access: REGEX. 281
Concatenating Variables. 283
Creating an Indexed Variable. 284
Creating a Standard Quote-Delimited String. 285
Performing a Calculation on a Variable. 289
Changing a Variable Value With the DECODE Function. 289
Extracting Characters From a Variable Value With the EDIT Function. 290
Removing Trailing Blanks From Variables With the TRUNCATE Function. 291
Calling a Function. 293
Using Variables to Alter Commands. 295
Using Numeric Amper Variables in Functions 296

Determining Amper Variable Data Type. 296
Manipulating Amper Variables. 296
Using an Amper Variable in an Expression. 297
Using Amper Variables as Subroutine Parameters. 298
Using a Numeric Amper Variable as a Numeric Subroutine Parameter. 298
Using a Numeric Amper Variable as an Alphanumeric Subroutine Parameter. 299
Debugging a Procedure 300
Issuing an Operating System Command 307
Dialogue Manager Quick Reference 307
-* Command. 308
? Command. 308
-CLOSE Command. 308
-CRTCLEAR Command. 309
-CRTFORM Command. 309
-DEFAULT[S|H] Command. 310
-EXIT Command. 311
-GOTO Command. 311
-IF Command. 311
-INCLUDE Command. 312
-label Command. 313
-MVS Command. 313
-MVS RUN Command. 313
-PASS Command. 314
-PROMPT Command. 314
-QUIT Command. 315
-READ Command. 316
-READFILE Command. 317
-REMOTE Command. 317
-REPEAT Command. 317
-RUN Command. 318
-SET Command. 319
-TSO Command. 319
-TSO RUN Command. 320

-TYPE Command. 320
-WINDOW Command. 321
-WRITE Command. 322
-" " Command. 322
Dialogue Manager Defaults and Limits. 323

4. Testing and Debugging With Query Commands 327

Using Query Commands 328
Displaying Combined Structures 330
Displaying Virtual Fields 330
Displaying the Currency Data Source in Effect 332
Displaying Available Fields 332
Displaying the File Directory Table 333
Displaying Field Information for a Master File 335
Displaying Data Source Statistics 336
Displaying Defined Functions 338
Displaying HOLD Fields 339
Displaying JOIN Structures 339
Displaying National Language Support 340
Displaying LET Substitutions 341
Displaying Information About Loaded Files 341
Displaying Explanations of Error Messages 342
Displaying PF Key Assignments 343
Displaying the Release Number 343
Displaying Parameter Settings 344
Displaying Graph Parameters 346
Displaying the Site Code 347
Displaying Command Statistics 348
Displaying StyleSheet Parameter Settings 351
Displaying Information About the SU Machine 353
Displaying Data Sources Specified With USE 353
Displaying Global Variable Values 354
Reporting Dynamically From System Tables 355

Overview of System Table Synonyms. 355
SYSAPPS: Reporting on Applications and Application Files. 357
SYSCOLUM: Reporting on Tables and Their Columns. 358
SYSDEFFN: Reporting on DEFINE FUNCTIONS. 359
SYSERR: Reporting on Error Message Files. 360
SYSFILES: Reporting on Metadata or Procedure Directory Information. 361
SYSIMP: Reporting on Impact Analysis Information. 363
SYSINDEX: Reporting on Index Information. 364
SYSKEYS: Reporting on Key Information. 365
SYSRPDIR: Reporting on Stored Procedures. 366
SYSSET: Reporting on SET Parameters. 367
SYSSQLOP: Reporting on Function Information. 367
SYSTABLE: Reporting on Table Information. 368
Reporting on Data Types. 369

5. Defining a Word Substitution 371

The LET Command 371
Variable Substitution 374
Null Substitution 376
Multiple-Line Substitution 377
Recursive Substitution 378
Using a LET Substitution in a COMPUTE or DEFINE Command 379
Checking Current LET Substitutions 379
Interactive LET Query: LET ECHO 380
Clearing LET Substitutions 381
Saving LET Substitutions in a File 382
Assigning Phrases to Function Keys 382

6. Enhancing Application Performance 385

FOCUS Facilities 385
Loading a File 385
Loading Master Files, FOCUS Procedures, and Access Files. 387
Displaying Information About Loaded Files. 388
Saving Master Files in Memory for Reuse 389

Accessing a FOCUS Data Source 392
Using MINIO. 392
Determining If a Previous Command Used MINIO. 393

7. Working With Cross-Century Dates 397

When Do You Use the Sliding Window Technique? 397
The Sliding Window Technique 398
Defining a Sliding Window. 399
Creating a Dynamic Window Based on the Current Year. 400
Applying the Sliding Window Technique 401
When to Supply Settings for DEFCENT and YRTHRESH. 401
Date Validation. 402
Defining a Global Window With SET 402
Defining a Dynamic Global Window With SET 405
Querying the Current Global Value of DEFCENT and YRTHRESH 408
Defining a File-Level or Field-Level Window in a Master File 409
Defining a Window for a Virtual Field 416
Defining a Window for a Calculated Value 422
Additional Support for Cross-Century Dates 427
Default Date Display Format. 427
Date Display Options. 428
System Date Masking. 428
Date Functions. 428
Date Conversion. 428
Century and Threshold Information. 428
Date Time Stamp. 429

8. Euro Currency Support 431

Integrating the Euro Currency 431
Converting Currencies 432
Creating the Currency Data Source 433
Identifying Fields That Contain Currency Data 436
Activating the Currency Data Source 438
Processing Currency Data 439

Querying the Currency Data Source in Effect 443
Punctuating Numbers 443
Selecting an Extended Currency Symbol 445

9. Designing Windows With Window Painter 449

Introduction 449
How Do Window Applications Work? 450
Window Files and Windows 451
Types of Windows You Can Create 452
Vertical Menus 452
Horizontal Menus 452
Text Input Windows 453
Text Display Windows 454
File Names Windows 454
Field Names Windows 455
File Contents Windows 456
Return Value Display Windows 456
Execution Windows 458
Multi-Input Windows 460
Creating Windows 461
Creating a Horizontal Menu 461
Pull-down Menus 464
Creating a Multi-Input Window 466
Integrating Windows and the FOCEXEC 469
Transferring Control in Window Applications 470
Return Values 472
Goto Values 473
Returning From a Window to Its Caller 474
Window System Variables 474
&WINDOWNAME 474
&WINDOWVALUE 474
Testing Function Key Values 475
Executing a Window From the FOCUS Prompt 476

Tutorial: A Menu-Driven Application 478
Creating the Application FOCEXEC. 479
Creating the Window File. 481
Creating the Text Display Window Named BORDER. 483
Creating the Text Display Window Named BANNER. 487
Creating the Vertical Menu Window Named MAIN. 489
Creating the Vertical Menu Window Named EXECTYPE. 495
Creating the File Names Window Named EXECNAME. 497
Executing the Application. 499
Window Painter Screens 499
Invoking Window Painter. 500
Entry Menu. 500
Main Menu. 502
Window Creation Menu. 504
Window Design Screen. 506
Window Options Menu. 509
Utilities Menu. 521
Transferring Window Files 524
Creating a Transfer File. 525
Transferring the File to the New Environment. 526
Editing the Transfer File. 526
The Format of the Transfer File. 526
Operating Environment Considerations. 530
Compiling the Transfer File. 532

A. Master Files and Diagrams 535

EMPLOYEE Data Source 535
EMPLOYEE Master File. 537
EMPLOYEE Structure Diagram. 538
JOBFILE Data Source 538
JOBFILE Master File. 539
JOBFILE Structure Diagram. 539
EDUCFILE Data Source 540

EDUCFILE Master File. 540
EDUCFILE Structure Diagram. 541
SALES Data Source 541
SALES Master File. 542
SALES Structure Diagram. 543
PROD Data Source 543
PROD Master File. 544
PROD Structure Diagram. 544
CAR Data Source 544
CAR Master File. 545
CAR Structure Diagram. 546
LEDGER Data Source 546
LEDGER Master File. 547
LEDGER Structure Diagram. 547
FINANCE Data Source 547
FINANCE Master File. 547
FINANCE Structure Diagram. 548
REGION Data Source 548
REGION Master File. 548
REGION Structure Diagram. 548
COURSES Data Source 549
COURSES Master File. 549
COURSES Structure Diagram. 549
EMPDATA Data Source 549
EMPDATA Master File. 550
EMPDATA Structure Diagram. 550
EXPERSON Data Source 550
EXPERSON Master File. 551
EXPERSON Structure Diagram. 551
TRAINING Data Source 551
TRAINING Master File. 552
TRAINING Structure Diagram. 552
COURSE Data Source 552

COURSE Master File. 552
COURSE Structure Diagram. 553
JOBHIST Data Source 553
JOBHIST Master File. 553
JOBHIST Structure Diagram. 553
JOBLIST Data Source 553
JOBLIST Master File. 554
JOBLIST Structure Diagram. 554
LOCATOR Data Source 554
LOCATOR Master File. 554
LOCATOR Structure Diagram. 555
PERSINFO Data Source 555
PERSINFO Master File. 555
PERSINFO Structure Diagram. 555
SALHIST Data Source 556
SALHIST Master File. 556
SALHIST Structure Diagram. 556
PAYHIST File 556
PAYHIST Master File. 556
PAYHIST Structure Diagram. 557
COMASTER File 557
COMASTER Master File. 558
COMASTER Structure Diagram. 559
VIDEOTRK, MOVIES, and ITEMS Data Sources 559
VIDEOTRK Master File. 560
VIDEOTRK Structure Diagram. 561
MOVIES Master File. 562
MOVIES Structure Diagram. 562
ITEMS Master File. 562
ITEMS Structure Diagram. 563
VIDEOTR2 Data Source 563
VIDEOTR2 Master File. 563
VIDEOTR2 Structure Diagram. 564

Gotham Grinds Data Sources 564
GGDEMOG Master File. 565
GGDEMOG Structure Diagram. 566
GGORDER Master File. 566
GGORDER Structure Diagram. 567
GGPRODS Master File. 567
GGPRODS Structure Diagram. 568
GGSALES Master File. 568
GGSALES Structure Diagram. 569
GGSTORES Master File. 569
GGSTORES Structure Diagram. 569
Century Corp Data Sources 570
CENTCOMP Master File. 571
CENTCOMP Structure Diagram. 571
CENTFIN Master File. 572
CENTFIN Structure Diagram. 572
CENTHR Master File. 573
CENTHR Structure Diagram. 575
CENTINV Master File. 576
CENTINV Structure Diagram. 576
CENTORD Master File. 577
CENTORD Structure Diagram. 578
CENTQA Master File. 579
CENTQA Structure Diagram. 580
CENTGL Master File. 580
CENTGL Structure Diagram. 581
CENTSYSF Master File. 581
CENTSYSF Structure Diagram. 581
CENTSTMT Master File. 582
CENTSTMT Structure Diagram. 583

B. Error Messages 585

Accessing Error Files 585

Displaying Messages 585

Legal and Third-Party Notices 587

Chapter 1

Customizing Your Environment

You can use the SET command to change parameters that govern your FOCUS environment.

In this chapter:

- When Do You Use the SET Command?
- Coding a SET Command
- Types of SET Parameters
- SET Parameter Syntax

When Do You Use the SET Command?

If you are an application developer, use the SET command to:

- Help you work efficiently and meet your testing and debugging needs.
- Provide a uniform and appropriate run-time environment for your end users with desirable defaults that do not need customization.

If you are creating your own ad hoc reports, use SET commands when you need to tailor the report presentation or content to meet your individual needs.

Coding a SET Command

The following guidelines apply to SET command syntax:

- You can set several parameters in one command by separating each with a comma.
- You can include as many parameters as you can fit on one line. If you exceed one line, repeat the SET command for each new line.
- You can set many, but not all, parameters using ON TABLE SET or ON GRAPH SET within a request. Parameters that cannot be set in this way are specified in the detailed description.

For the specific syntax of a parameter with its valid values, see *SET Parameter Syntax* on page 50.

Syntax: How to Set Parameters

```
SET parameter = option[, parameter = option,...]
```

where:

parameter

Is the setting you wish to change.

option

Is a valid value for the parameter.

Example: Setting a Single Parameter

In the following example, the PAGE-NUM parameter suppresses default page numbering.

```
SET PAGE-NUM = OFF

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
END
```

The output is:

```
LAST_NAME      FIRST_NAME

STEVENS        ALFRED
SMITH          MARY
JONES          DIANE
SMITH          RICHARD
BANNING        JOHN
IRVING         JOAN
ROMANS         ANTHONY
MCCOY          JOHN
BLACKWOOD      ROSEMARIE
MCKNIGHT       ROGER
GREENSPAN      MARY
CROSS          BARBARA
```

Example: Setting Multiple Parameters

The following example sets two parameters in one command in a stored procedure. The first parameter, NODATA, changes the default character for missing data from a period to the word NONE. The second parameter, PAGE-NUM, suppresses default page numbering.

```
SET NODATA = NONE, PAGE-NUM = OFF
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY EMP_ID
ACROSS DEPARTMENT
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
END
```

In the output, NONE appears when there is no salary information for a specific employee because that employee does not work in the department that is referenced. There is no page number at the top of the output.

The output is:

```
                        DEPARTMENT
EMP_ID          MIS                PRODUCTION

071382660             NONE         $11,000.00
112847612       $13,200.00               NONE
117593129       $18,480.00               NONE
119265415             NONE          $9,500.00
119329144             NONE         $29,700.00
123764317             NONE         $26,862.00
126724188             NONE         $21,120.00
219984371       $18,480.00               NONE
326179357       $21,780.00               NONE
451123478             NONE         $16,100.00
543729165        $9,000.00               NONE
818692173       $27,062.00               NONE
```

Syntax: How to Set Parameters in a Report Request

```
ON TABLE SET parametervalue [AND parametervalue ...]
```

where:

parameter
Is the setting you wish to change.

value
Is a valid value for the parameter.

Example: Setting Parameters in a Report Request

In the following example, the command ON TABLE SET changes the default character for missing data from a period to the word NONE and suppresses default page numbering.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY EMP_ID
ACROSS DEPARTMENT
ON TABLE SET NODATA NONE AND PAGE-NUM OFF
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
END
```

In the output, NONE appears when there is no salary information for a specific employee. There is no page number at the top of the output.

The output is:

```
                        DEPARTMENT
EMP_ID          MIS                PRODUCTION

071382660             NONE         $11,000.00
112847612       $13,200.00               NONE
117593129       $18,480.00               NONE
119265415             NONE          $9,500.00
119329144             NONE         $29,700.00
123764317             NONE         $26,862.00
126724188             NONE         $21,120.00
219984371       $18,480.00               NONE
326179357       $21,780.00               NONE
451123478             NONE         $16,100.00
543729165        $9,000.00               NONE
818692173       $27,062.00               NONE
```

Syntax: How to Set Parameters in a Graph Request

```
ON GRAPH SET parametervalue [AND parametervalue ...]
```

where:

parameter

Is the setting you wish to change.

value

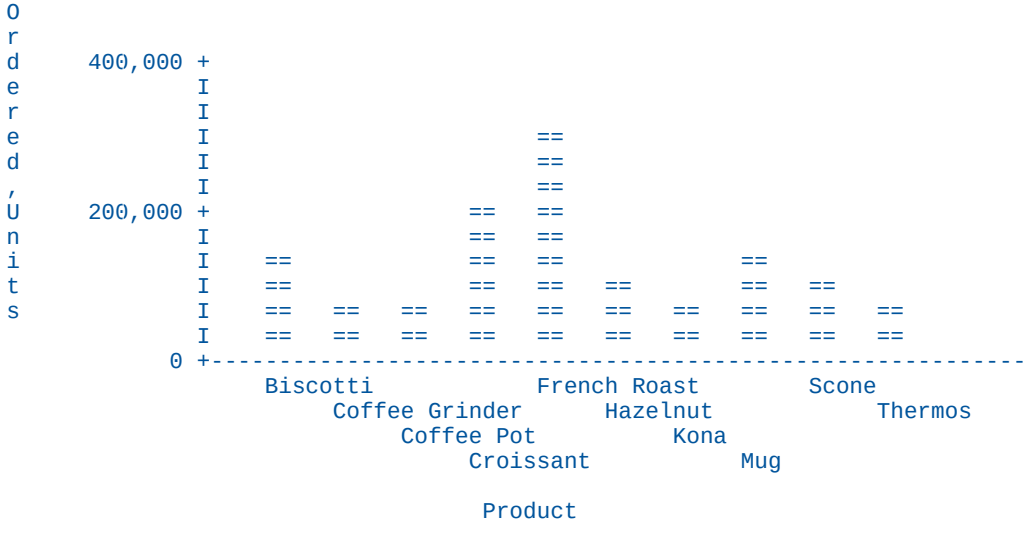
Is a valid value for the parameter.

Example: Setting Parameters in a Graph Request

In the following example, the command ON GRAPH SET changes the default setting for the 3D parameter to OFF.

```
GRAPH FILE GGORDER
SUM QUANTITY
ACROSS PRODUCT_DESC
ON GRAPH SET 3D OFF
END
```

The output is:



Types of SET Parameters

This topic lists the types of tasks that can be accomplished, and the SET parameters that allow you to perform these tasks. If a single parameter applies to more than one activity, it appears in more than one category. For more detailed descriptions, as well as the syntax for each parameter, see *SET Parameter Syntax* on page 50.

The following are the types of tasks performed with SET parameters.

Calculations	Affects the way calculations are performed in FOCUS.
Data and Metadata	Determines the way data is stored and processed.

Date Manipulation Tasks	Controls the way dates are processed and displayed on reports.
Graph Tasks	Controls the processing and display of graphs.
Memory Setup and Optimization Tasks	Affects the memory and optimization of your application.
Report Code, Content, and Processing Tasks	Determines the content and processing of a request.
Report Layout and Display Tasks	Affects the display of a report.
Security Tasks	Controls user access to data sources and procedures.
Terminal Tasks	Specifies the options for display in your terminal.

Calculations

The following parameters control the behavior of calculations in FOCUS.

`AGGR[RATIO]`

Determines the ratio of aggregation based on retrieved records and the final size of the answer set.

`CDN`

Specifies the punctuation used in numeric notation.

`COMPUTE`

Controls the compilation of expressions.

`DMPRECISION`

Specifies precision of numeric values in Dialogue Manager -SET commands to calculate accurate numeric variable values.

`FLOATMAPPING`

Takes advantage of decimal-based precision numbers for all numeric processing for floating point numbers.

`MISS_ON`

Sets a default value, either SOME or ALL for MISSING ON in DEFINE and COMPUTE.

MISSINGTEST

Determines whether the IF expression in IF-THEN-ELSE tests is checked for missing values.

MODCOMPUTE

Controls compilation of MODIFY calculations.

PARTITION_ON

Sets the partition size for statistical functions.

USERFCHK

Controls the level of verification applied to DEFINE FUNCTION arguments and Information Builders-supplied function arguments.

USERFNS

Determines whether an Information Builders-supplied function or a locally-written function with the same name is used.

Data and Metadata

The following parameters determine the way data is stored and processed.

ACCBLN

Accepts blank or zero values for fields with ACCEPT commands in the Master File.

ASNAMES

Controls the FIELDNAME attribute in a HOLD Master File.

BLKCALC

Enables system-determined blocking for HOLD files written to DASD.

COUNTWIDTH

Expands the default format of COUNT fields from a five byte integer to a nine byte integer.

DATEFORMAT

Specifies the order of the date components (month/day/year) when date-time values are entered in a formatted-string or translated-string format.

DEFINES

Compiles virtual fields into machine code to improve performance.

DIRECTHOLD

Controls whether HOLD Files in FOCUS format are created directly.

DTSTRICT

Controls the use of strict processing for date-time fields.

EQTEST

Controls whether the characters $ and $* are treated as wildcard characters or normal characters in selection criteria.

EUROFILE

Activates the data source that contains information for the currency you want to convert.

FIELDNAME

Controls the use of long and qualified field names.

FOCALLOC

Automatically allocates FOCUS files.

HIPERFOCUS

Activates HiperFOCUS.

HIPERINSTALL

Installs or disables HiperFOCUS.

HLDCOM_TRIMANV

Controls whether trailing blanks are retained when the output is held in a delimited format.

HNODATA

Controls missing values propagated to a HOLD file.

HOLDFORMAT

Determines the default format for HOLD files.

HOLDLIST

Determines what fields in a report request are included in the HOLD file.

HOLDMISS

Distinguishes between missing data and default data (zeros or blanks) in a HOLD file.

HOLDSTAT

Determines if comments and DBA information are included in HOLD Master Files.

HTMLARCHIVE

Packages HTML and DHTML reports together with image files into a single web archive document (.mht file).

HTMLENCODE

Encodes data within HTML output.

INDEX

Is the indexing scheme used for indexes.

KEEPDEFINES

Controls whether a virtual field created for a host or joined structure is retained after a JOIN command is run.

MASTER

Enables use of blank delimited (Fusion) Master File syntax, and provides increased enforcement of syntax rules in comma delimited Master File syntax.

MAXDATAEXCPT

Enables you to change the number of data exceptions allowed before the session is terminated.

MAXLRECL

Specifies the maximum length of a record described with the Master File OCCURS attribute.

MDICARDWARN

Displays a warning message when the cardinality of a dimension exceeds a specified value.

MDIENCODING

Enables retrieval of output from an MDI file without reading the data source.

MDIPROGRESS

Displays messages about the progress of an MDI build.

MINIO

Determines whether a block is read more than once when reading or writing to a file.

NULL

Enables creation of a variable-length comma or tab delimited HOLD file that differentiates between a missing value and a blank string or zero value.

OLDSTYRECLEN

Determines whether the record length, LRECL, is set to the current setting of LRECL=0, or the older setting of LRECL=512.

PCOMMA

Enables retrieval of comma delimited files created by a PC application or HOLD FORMAT COM command.

PREFIX

Specifies the prefix of existing data sets automatically allocated by FOCUS.

QUALCHAR

Specifies the qualifying character to be used in qualified field names.

RANK

Determines how rank numbers are assigned in a request when multiple data values fall into the same rank category.

SAVEDMASTERS

Saves a Master File to memory after it has been used in a request.

SHADOW

Activates the Absolute File Integrity feature.

SHIFT

Controls the use of *shift* strings.

SUSI

See *Simultaneous Usage Reference Manual for z/OS* .

SUTABSIZE

See *Simultaneous Usage Reference Manual for z/OS* .

TRACKIO

Gathers more pages to fill a track before reading or writing the pages to disk.

WEBARCHIVE

Packages multiple EXL2K files into a single file.

WEEKFIRST

Specifies what day of the week is the start of the week.

WIDTH

Used for communication between 3270 terminals and the operating system.

WPMINWIDTH

Specifies a minimum width for format WP output files.

XRETRIEVAL

Controls the retrieval of data when previewing a report.

XFOCUSBINS

Defines the number of pages of memory to use as buffers for XFOCUS data sources.

Date Manipulation Tasks

The following parameters control the way dates are processed and displayed in reports.

BUSDAYS

Specifies which days are considered business days and which are not.

DATE_ORDER

Specifies the order of date components.

DATE_SEPARATOR

Specifies the separator for date components.

DATEDISPLAY

Controls the display of a base date

DATEFNS

Activates year 2000-compliant versions of date subroutines.

DATETIME

Sets date and time in reports .

DEFCENT

Defines a default century for your application.

EXL2KTXTDATE

Controls whether translated dates are sent as date values with format masks instead of text values.

HDAY

Specifies the holiday file from which to retrieve dates that are considered holidays.

LEADZERO

Avoids the truncation of leading zeros.

TIME_SEPARATOR

Specifies the separator for time components for the &TOD variable.

TESTDATE

Temporarily alters the system date in order to test a dynamic window.

YRTHRESH

Defines the start of a 100-year window.

Graph Tasks

The following parameters control the processing and display of graphs. For information about these parameters, see the *Creating Reports* Manual.

AUTOTICK

Sets the tick mark intervals for graphs.

BARNUMB

Places summary numbers at the end of bars on bar charts, or slices on pie charts.

BARSPACE

Specifies the number of lines separating the bars on bar charts.

BARWIDTH

Specifies the number of lines per bar on bar charts.

BSTACK

Specifies whether bar chart bars are stacked or placed side by side.

DEVICE

Specifies the plotting device or terminal to be used.

FRAME

For GDDM graphics, indicates if you want a frame around your graph.

GCOLOR (or GRIBBON)

Depending on device type, determines black and white or color patterns or ribbons used.

GMISSING

Specifies whether variables with the value specified in GMISSVAL are to be ignored.

GMISSVAL

Specifies the variable value that represents missing data.

GPROMPT

Specified whether FOCUS should prompt for graph parameters.

GRIBBON

Same as GCOLOR.

GRID

Draws a grid of parallel horizontal lines at the vertical class marks on the graph.

GTREND

Specifies the use of basic linear regression to alter the X and Y axis values in a SCATTER graph.

HAUTO

Performs automatic scaling of the horizontal axis for the given values.

HAXIS

Specifies the width, in characters, of the horizontal axis.

HCLASS

Specifies the horizontal interval mark when AUTOTICK is OFF.

HISTOGRAM

Draws a histogram instead of a curve when the values on the horizontal axis are not numeric.

HMAX

Sets the maximum value on the horizontal axis when automatic scaling is not used (HAUTO=OFF).

HMIN

Sets the minimum value on the horizontal axis when automatic scaling is not used (HAUTO=OFF).

PAUSE

Specifies whether there is a pause for paper adjustment on the plotter after the request is executed.

PIE

Specifies a pie chart.

PLOT

Specifies the width and height settings for certain devices.

PRINT

Specifes whether the graph is printed or displayed on the terminal.

TERM[INAL]

Specifies the plotting device or terminal to be used.

VAUTO

Performs automatic scaling of the vertical axis for the given values.

VAXIS

Specifies the length of the vertical axis, in lines.

VCLASS

Specifies the vertical interval mark when AUTOTICK is OFF.

VGRID

Draws a grid at the horizontal and vertical class marks of the graph.

VMAX

Sets the maximum value on the vertical axis when automatic scaling is not used (VAUTO=OFF).

VMIN

Sets the minimum value on the vertical axis when automatic scaling is not used (VAUTO=OFF).

VTICK

Sets the vertical axis interval mark when AUTOTICK is OFF.

VZERO

Treats missing values on the vertical axis as zeros.

Memory Setup and Optimization Tasks

The following parameters control the memory and optimization of your application.

AUTOINDEX

Retrieves data faster by automatically taking advantage of indexed fields in a FOCUS data source.

AUTOPATH

Dynamically selects an optimal retrieval path.

AUTOSTRATEGY

Determines when FOCUS stops the search for a key field specified in a WHERE or IF test.

BINS

Specifies the number of pages of memory used for data source buffers.

CACHE

Stores FOCUS data source pages in memory and buffers between the data source and BINS.

COMPUTE

Controls the compilation of expressions.

DEFINES

Compiles virtual fields into machine code to improve performance.

DMH_LOOPLIM

Controls the number of loop iterations allowed in Dialogue Manager.

DMH_STACKLIM

Controls the number of lines allowed in FOCSTACK.

ESTRECORDS

Passes the estimated number of records to be sorted in the request.

FIXRETRIEVE

Enables keyed retrieval from a fixed format sequential file, such as a HOLD file.

FOCSTACK

Specifies the amount of space, in thousands of bytes, used by FOCUS commands waiting for execution.

HLISUTRACE

Records the last 20 events that the FOCUS Database Server performed.

HLISUDUMP

Is used for debugging FOCUS Database Server problems.

IBMLE

This parameter is no longer functional. FOCUS is fully LE compliant, and all FOCUS applications must be LE compliant

IMMEDTYPE

Tells FOCUS where to send line mode output.

SQLTOPTTF

Enables the SQL Translator to generate TABLEF commands instead of TABLE commands.

SUWEDGE
Keeps Master Files on a FOCUS Database Server open between requests.

ZIIP
Enables you to offload specific categories of FOCUS processing to a zIIP specialty engine.

Report Code, Content, and Processing Tasks

The following parameters affect the content or processing of a report.

ALL

Handles missing segment instances in a report.

ALLOWCVTERR

Controls the display of a row of data that contains an invalid date format.

ASNAMES

Controls the FIELDNAME attribute in a HOLD Master File.

AUTOTABLEF

Avoids creating the internal matrix based on the features used in the query.

BUSDAYS

Specifies which days are considered business days.

CARTESIAN

Generates a report containing all combinations of non-related data instances in a multi-path request containing a PRINT or LIST command.

CDN

Specifies punctuation used in numeric notation.

CENT-ZERO

Displays a leading zero in decimal-only numbers.

COLLATION
Controls ordering of alphanumeric values.

COMPMISS

Controls whether the missing attribute is propagated to reformatted fields in a report request.

COMPUTE

Controls the compile of expressions.

DATEDISPLAY

Controls the display of date format fields that contain the value zero.

DATEFNS

Activates year 2000-compliant versions of date subroutines.

DATETIME

Sets date and time in a report .

DBAJOIN

Controls whether DBA restrictions are treated as report filters or are added to the join conditions.

DB_INFILE

Controls whether the expression generated by the DB_INFILE function for use against a relational data source is optimized.

DEFCENT

Defines a default century for your application.

DEFECHO

Defines a default value for the &ECHO variable for your application.

EMPTYREPORT

Controls the output generated when a report request retrieves zero records.

ERROROUT

Terminates a request and returns a message when an error is encountered.

ESTRECORDS

Passes the estimated number of records to be sorted in the request.

EXCELSERVURL

Specifies the location to be used to zip the file components that comprise an EXCEL 2007 file (.xlsx).

EXL2KLANG

Specifies the language used for Microsoft® Excel requests. This language must be the same as the language of Excel on the browser machine.

EXTAGGR

Enables aggregation in an external sort.

`EXTHOLD`

Enables you to use an external sort to create HOLD files.

`EXTRACT`

Activates Structured HOLD Files for a request.

`EXTSORT`

Activates the external sorting feature.

`FIELDNAME`

Controls the use of long and qualified field names.

`FILENAME`

Specifies a file to be used, by default, in commands.

`FILTER`

Activates declared filters.

`FOC144`

Suppresses warning message FOC144, which reads *Warning Testing in Independent sets of Data* .

`FORMULTIPLE`

Allows you to include the same value of a FOR field in multiple rows of the FML matrix.

`HNODATA`

Controls missing values propagated to a HOLD file.

`HOLDATTR`

Includes the TITLE and ACCEPT attributes from the original Master File in the HOLD Master File.

`JOINLM`

Controls whether strict equality is required or partial key joins are supported for record-oriented adapters. JOINLM is a synonym for JOIN_LENGTH_MODE.

`JOINOPT`

Ensures proper alignment of report output by correcting for lagging (missing) values. Also enables joins between fields with different numeric data types.

`KEEPDEFINES`

Controls whether a virtual field created for a host or joined structure is retained after a JOIN command is run.

LANG[UAGE]

The LANG[UAGE] parameter specifies the National Language Support (NLS) environment. It sets the language of FOCUS error messages and can also be used to set the language of report titles if the Master File Description contains alternate language TITLE attributes.

LEADZERO

Avoids the truncation of leading zeros.

MESSAGE

Controls the display of informational messages.

MULTIPATH

Controls whether MATCH requests use grouped or ungrouped processing.

NODATA

Determines the character string that indicates missing data in a report.

ONFIELD

Controls whether ON phrases are ignored for fields not referenced in a request.

PAUSE

Pauses before displaying a FOCUS report on the terminal.

PARTITION_ON

Controls the partition size for statistical functions.

PFnn

Assigns a function to a PF key.

PDFLINETERM

Determines if an extra space is appended to each record of a PDF output file to facilitate proper file transfer between Windows and UNIX.

PHONETIC_ALGORITHM

Sets the phonetic algorithm to use with the PHONETIC function.

PRINTDST

Controls processing of reports that use the PRINT command in conjunction with multiple DST operators.

QUALCHAR

Specifies the qualifying character to be used in qualified field names.

SAVEMATRIX

Saves the matrix from your request to protect it from being overwritten when using Dialogue Manager commands.

SHORTPATH

Controls how tests against missing cross-referenced segment instances are processed in a left outer join.

SORTLIB

Tells FOCUS which sort package is installed at your site.

SORTMATRIX

Controls whether to employ in-memory sorting with decreased use of external memory.

SORTMEMORY

Controls the amount of internal memory available for sorting.

SUMMARYLINES

Permits the combination of fields with and without prefix operators on summary lines in one request.

SUMPREFIX

Allows users to choose the answer set display order when using an external sort to perform aggregation of alphanumeric or smart date formats.

TITLES

Uses predefined column titles in the Master File as column titles in report output.

Report Layout and Display Tasks

The following parameters affect the layout and display of a report.

ACROSSLINE

Controls underlining of column titles on report output. TITLELINE is a synonym.

ACROSSPRT

Reduces the number of report lines within each sort group when a request uses the PRINT command and an ACROSS phrase.

ACROSSTITLE

Controls whether ACROSS titles display above or to the left of ACROSS values.

ACRSVRBTITL

Controls the display of ACROSS column titles when there is only one displayed field for an ACROSS group.

ALTBACKPERLINE

Alternates the background color by line for reports that use positioned drivers, for example PDF, DHTML, PPT, and PPTX.

AUTOFIT

Controls resizing of HTML report output to fit its window.

BASEURL

Specifies a default location where your browser searches for relative URLs referenced in the HTML documents created by FOCUS.

BLANKINDENT

Clarifies relationships within an FML hierarchy by indenting the captions (titles) of values at each level.

BOTTOMMARGIN

Sets the bottom boundary for report contents on a page in a styled report.

BYDISPLAY

Displays a sort field on every row, column, or both in a report.

BYPANEL

Controls the repetition of BY fields on panels.

BYSCROLL

Scrolls report headings and footings along with the report contents.

CENT-ZERO

Displays a leading zero in decimal-only numbers.

COLUMNSCROLL

Enables you to scroll by column within the panels of a report provided that the report is wider than the screen width.

COMPOUND

Enables you to combine multiple reports into a single PDF or PS file to create a compound report.

CSSURL

Links an HTML report to an external cascading style sheet (CSS) file in order to style the report.

CURRENCY_DISPLAY

Defines the position of the currency symbol relative to the monetary number.

CURRENCY_ISO_CODE

Defines the ISO code for the currency symbol to use.

CURRENCY_PRINT_ISO

Defines what will happen when the currency symbol cannot be displayed by the code page in effect.

CURRSYMB

Sets a currency symbol to display on the report output when a numeric format specification uses the M or N display options.

CURSYM_D

Sets the characters to display on the report output when a numeric format specification uses the :D or :d display options.

CURSYM_E

Sets the characters to display on the report output when a numeric format specification uses the :E or :e display options.

CURSYM_F

Sets the characters to display on the report output when a numeric format specification uses the :F display option.

CURSYM_G

Sets the characters to display on the report output when a numeric format specification uses the :G display option.

CURSYM_L

Sets the characters to display on the report output when a numeric format specification uses the :L or :l display options.

CURSYM_Y

Sets the characters to display on the report output when a numeric format specification uses the :Y or :y display options.

CUSTOM-PAGE-LENGTH

Sets the page length for PAGESIZE=CUSTOM.

CUSTOM-PAGE-WIDTH

Sets the page width for PAGESIZE=CUSTOM.

DISPLAYROUND

Adds a small number to floating point and double-precision numbers for display, in order to correct rounding errors caused by conversion from binary to decimal.

DROPBLNKLINE

Eliminates blank lines from the report output.

DUPLICATECOL

Controls whether columns for multiple display commands are spread out or stacked on top of each other.

EXTENDNUM

Prevents visual overflow on reports.

FOCFIRSTPAGE

Assigns a page number to the first page of output.

HIDENULLACRS

Hides ACROSS columns containing only null values.

HTMLCSS

Creates an inline Cascading Style Sheet command in the HTML page that displays the report output.

HTMLEMBEDIMG

Determines whether to embed images and graphs directly into an HTML or DHTML .htm file.

LAN[GUAGE]

Specifies the National Language Support (NLS) environment. Sets the language of FOCUS error messages. Can also be used to set the language of report titles if the Master File Description contains alternate language TITLE attributes.

LAYOUTGRID

Displays a grid in the report output, which enables you to evaluate the correct placement of data and objects during your report design. This option is applicable only when using the PDF, PS, or DHTML report output.

LEFTMARGIN

Sets the left boundary for report contents on a page in a styled report.

LINES

Sets the maximum number of lines of printed output that appear on a page, from the heading at the top to the footing on the bottom. The OFFLINE-FMT parameter determines the format of printed report output generated from a request.

ORIENTATION

Specifies the page orientation for styled reports.

OVERFLOWCHAR

Changes the character that displays in a numeric report column when the column does not have enough space for the value.

PAGE[-NUM]

Controls the numbering of output pages.

PAGE-SCALE

Scales wide PDF report output to fit the width of the page.

PAGESIZE

Specifies the page size for StyleSheets.

PANEL

Sets the maximum line width of a report panel.

PAPER

Specifies the length of paper for printed output.

PCTFORMAT

Controls whether fields prefixed with PCT., RPCT., and PCT.CNT. display with the format of the original field or with a percent sign.

PRFTITLE

Generates readable and translatable column titles for prefixed fields on reports.

PRINT

Specifies the report output destination.

PRINTPLUS

Specifies enhancements to display alternatives.

PSPAGESETUP

Coordinates the paper source used by a PostScript printer with the PAGESIZE parameter setting.

QUALTITLES

Uses qualified column titles in report output when duplicate field names exist in a Master File.

REBUILDMSG

Allows direct control over the frequency with which REBUILD issues messages.

RECAP-COUNT

Includes lines containing a value created with RECAP when counting the number of lines per page for printed output.

RIGHTMARGIN

Sets the right boundary for report contents on a page.

SHOWBLANKS

Preserves leading and internal blanks in HTML and EXL2K report output.

SPACES

Sets the number of spaces between columns in a report.

SQUEEZE

Determines the column width in report output.

STYLE[SHEET]

Controls the format of report output by accepting or rejecting StyleSheet parameters.

SUBTOTALS

Controls whether summary lines display above or below the data.

TERM[INAL]

Selects the terminal type.

TITLELINE

Controls underlining of column titles. ACROSSLINE is a synonym.

TOPMARGIN

Sets the top boundary on a page for report output.

TRANTERM

Displays extended currency symbols on TSO.

UNITS

Specifies the unit of measure for page margins, column positions, and column widths.

WEBTAB

Encloses CRTFORM display fields in @ signs.

XLSXPAGEBRKIGNORE

Synchronizes FOCUS page breaks with Excel page breaks for format XLSX.

Security Tasks

The following parameters specify user access to data sources and procedures.

DBACSENSITIV

Controls whether password validation is case-sensitive.

DBASOURCE

Controls the source of access restrictions in a multi-file structure.

PASS

Enables user access to a data source or stored procedure protected by Information Builders security.

PERMPASS

The PERMPASS parameter establishes a user password that remains in effect throughout a session or connection.

USER

In FOCUS , enables user access to a data source or stored procedure protected by Information Builders security.

Terminal Tasks

The following parameters specify options for display in your terminal.

DISPLAY

Is the PC display mode selection.

EXTTERM

Enables the use of extended terminal attributes.

HOTMENU

Automatically displays the Hot Screen PF key legend at the bottom of the Hot Screen report.

SBORDER

Generates a solid border on the screen for full-screen mode.

SCREEN

Selects the Hot Screen facility.

TRMOUT

Suppresses all output messages to the terminal.

SET Parameter Syntax

This topic alphabetically lists the SET parameters that control the environment with a description and the syntax.

A	B	C	D
3D	BINS	CACHE	DATE_ORDER
ACCBLN	BLANKINDENT	CARTESIAN	DATE_SEPARATOR
ACROSSLINE	BOTTOMMARGIN	CDN	DATEDISPLAY
ACROSSPRT	BUSDAYS	CENT-ZERO	DATEFORMAT
ACROSSTITLE	BYDISPLAY	CNOTATION	DATETIME
ACRSVRBTITL	BYPANEL	COLLATION	DB_INFILE
ALL		COMPMISS	DBACSENSITIV
ALLOWCVTERR		COMPOUND	DBAJOIN
ALTBACKPERLINE		COMPUTE	DBASOURCE
ASNAMES		COUNTWIDTH	DEFCENT
AUTOFIT		CSSURL	DEFECHO
AUTOINDEX		CURRSYMB	DEFINES
AUTOPATH		CURSYM_D	DIRECTHOLD
AUTOSTRATEGY		CURSYM_E	DISPLAYROUND
AUTOTABLEF		CURSYM_F	DMH_LOOPLIM
		CURSYM_G	DMH_STACKLIM
		CURSYM_L	DMPRECISION
		CURSYM_Y	DRILLMETHOD
		CUSTOM_PAGE _LENGTH	DROPBLNKLINE
			DTSTRICT
		CUSTOM_PAGE _WIDTH	DUPLICATECOL

E	F	G	H
EMBEDDABLE	FIELDNAME		HDAY
EMPTYREPORT	FILECOMPRESS		HIDENULLACRS
EQTEST	FILENAME		HLDCOM_TRIMANV
ERROROUT	FILTER		HNODATA
ESTRECORDS	FIXRETRIEVE		HOLDATTRS
EUROFILE	FOC144		HOLDFORMAT
EXCELSERVURL	FOCEXURL		HOLDLIST
EXL2KLANG	FOCFIRSTPAGE		HOLDMISS
EXL2KTXTDATE	FOCSTACK		HOLDSTAT
EXTAGGR	FORMULTIPLE		HTMLARCHIVE
EXTENDNUM			HTMLCSS
EXTHOLD			HTMLEMBEDIMG
EXTRACT			HTMLENCODE
EXTSORT			

I	J	K	L
INDEX	JOIN_LENGTH_MODE	KEEPDEFINES	LANGUAGE
	JOINOPT	KEEPFILTERS	LAYOUTGRID
			LEADZERO
			LEFTMARGIN
			LINES
			LOOKGRAPH

M	N	O	P
MATCHCOLUMNORDER	NODATA	OLDSTYRECLEN	PAGE-NUM
MAXDATAEXCPT	NULL	ONFIELD	PAGE-SCALE
MAXLRECL		ORIENTATION	PAGESIZE
MDICARDWARN		OVERFLOWCHAR	PANEL
MDIENCODING			PARTITION_ON
MDIPROGRESS			PASS
MESSAGE			PCOMMA
MISS_ON			PCTFORMAT
MISSINGTEST			PDFLINETERM
MULTIPATH			PERMPASS
			PHONETIC_ALGORITHM
			PRFTITLE
			PRINT
			PRINTDST
			PRINTPLUS
			PSPAGESETUP

Q	R	S	T
QUALCHAR	RANK	SAVEDMASTERS	TESTDATE
QUALTITLES	RECAP-COUNT	SAVEMATRIX	TITLELINE
	RECORDLIMIT	SHADOW	TITLES
	RIGHTMARGIN	SHIFT	TOPMARGIN
	RPAGESET	SHORTPATH	
		SHOWBLANKS	
		SORTMATRIX	
		SORTMEMORY	
		SPACES	
		SQLTOPTTF	
		SQUEEZE	
		STYLEMODE	
		STYLESHEET	
		SUBTOTALS	
		SUMMARYLINES	
		SUMPREFIX	

U	V	W	X-Y-Z
UNITS		WARNING	XLSXPAGEBRKIGNORE
USER		WEBARCHIVE	XRETRIEVAL
USERCHK		WEEKFIRST	YRTHRESH
USERFNS		WPMINWIDTH	

ACCBLN

The ACCBLN parameter accepts blank or zero values for fields with ACCEPT commands in the Master File (see the *Describing Data* manual).

The syntax is:

```
SET ACCBLN = {ON|OFF}
```

where:

ON

Accepts blank and zero values for fields with ACCEPT commands unless blank or zero values are explicitly coded in the list of acceptable values. ON is the default value.

OFF

Does not accept blank and zero values for fields with ACCEPT commands unless blank or zero values are explicitly coded in the list of acceptable values.

ACROSSLINE

The ACROSSLINE parameter controls underlining of column titles on report output. TITLELINE is a synonym for ACROSSLINE.

The syntax is:

```
SET {ACROSSLINE|TITLELINE} = {ON|OFF|SKIP}
```

where:

ON

Underlines column titles on report output. ON is the default value.

OFF

Replaces the underline with a blank line.

SKIP

Specifies no underline and no blank line.

ACROSSPRT

The ACROSSPRT parameter reduces the number of report lines within each in a request that uses the PRINT command and an ACROSS phrase.

The PRINT command generates a report that has a single line for each record retrieved from the data source after screening out those that fail IF or WHERE tests. When PRINT is used in conjunction with an ACROSS phrase, many of the generated columns may be empty. Those columns display the missing data symbol.

To avoid printing such a sparse report, you can use the SET ACROSSPRT command to compress the lines in the report. The number of lines is reduced within each sort group by swapping non-missing values from lower lines with missing values from higher lines, and then eliminating any lines whose columns all have missing values.

Because data may be moved to different report lines, row-based calculations, such as ROW-TOTAL and ACROSS-TOTAL in a compressed report are different from those in a non-compressed report. Column calculations are not affected by compressing the report lines.

The syntax is:

```
SET ACROSSPRT = {NORMAL|COMPRESS}
```

where:

NORMAL

Does not compress report lines.

COMPRESS

Compresses report lines by promoting data values up within a sort group.

ACROSSTITLE

In a report that uses the ACROSS sort phrase to sort values horizontally across the page, by default, two lines are generated on the report output for the ACROSS columns. The first line displays the name of the sort field (ACROSS title), and the second line displays the values for that sort field (ACROSS value). The ACROSS field name is left justified above the first ACROSS value.

The ACROSSTITLE parameter enables you to display both the ACROSS title and the ACROSS values on one line in PDF, HTML, EXL2K, or EXL07 report output. You can issue the SET ACROSSTITLE = SIDE command. This command places ACROSS titles to the left of the ACROSS values. The titles are right justified in the space above the BY field titles. The heading line that is created, by default, to display the ACROSS title will not be generated.

This feature is designed for use in requests that have both ACROSS fields and BY fields. For requests with ACROSS fields but no BY fields, the set command is ignored, and the ACROSS titles are not moved.

The syntax is:

```
SET ACROSSTITLE = {ABOVE|SIDE}
```

where:

ABOVE

Displays ACROSS titles above their ACROSS values. ABOVE is the default value.

SIDE

Displays ACROSS titles to the left of their ACROSS values, above the BY columns.

ACRSVRBTITL

Using the SET ACRSVRBTITL command, you can control the display of an ACROSS column title in an ACROSS group. The behavior of the title is determined by the number of verb columns in the ACROSS group. The field count is affected by the following features, which add internal matrix columns to the report:

- Fields in a heading or footing.
- Fields whose display is suppressed with the NOPRINT phrase.
- Reformatted fields (which are normally counted twice).
- A COMPUTE command referencing multiple fields.

The syntax is:

```
SET ACRSVRBTITL = {HIDEONE|ON|OFF}

ON TABLE SET ACRSVRBTITL {HIDEONE|ON|OFF}
```

where:

HIDEONE

Suppresses the title when there is only one display field, or there is only one display field and the request contains one or more of the features that add internal matrix columns to the report. This value is the default.

ON

Always displays the title even if there is only one display field.

OFF

Suppresses the title when there is only one display field. Displays the title when there is only one display field and the request contains one or more of the features that add internal matrix columns to the report. This is legacy behavior.

ALL

The ALL parameter handles missing segment instances in a report.

The command SET ALL = ON specifies a left outer join. With a left outer join, all records from the host file display on the report output. If a cross-referenced segment instance does not exist for a host segment instance, the report output displays missing values for the fields from the cross-referenced segment.

If there is a screening condition on the dependent segment, those dependent segment instances that do not satisfy the screening condition are omitted from the report output, and so are their corresponding host segment instances.

The syntax is:

```
SET ALL = {ON|OFF|PASS}
```

where:

ON

Includes missing segment instances in a report when fields in the segment are not screened by WHERE or IF criteria in the request. The missing field values are denoted by the NODATA character, set with the NODATA parameter (for more information, see *NODATA* on page 119).

OFF

Omits missing segment instances from a report. OFF is the default value.

PASS

Includes missing segment instances in a report, regardless of WHERE or IF criteria in the request.

This option is not supported when MULTIPATH = COMPOUND (see *MULTIPATH* on page 119).

ALLOWCVTERR

The ALLOWCVTERR parameter applies to non-FOCUS data sources when converting from the way the date is stored (ACTUAL attribute) to the way it is formatted (FORMAT or USAGE attribute).

It controls the display of a row of data that contains an invalid date format (formerly called a smart date). When it is set to ON, the invalid date format is returned as the base date or a blank, depending on the settings for the MISSING and DATEDISPLAY parameters.

Note: The ALLOWCVTERR parameter is not supported for virtual fields.

The syntax is:

```
SET ALLOWCVTERR = {ON|OFF}
```

where:

ON

Displays a row of data that contains an invalid date format. When ALLOWCVTERR is set to ON, the display of invalid dates is determined by the settings of the MISSING attribute and DATEDISPLAY command.

The results are explained in the following table:

DATEDISPLAY	MISSING	RESULT
OFF	OFF	A blank is returned.
	ON	The value of the NODATA character (a period, by default) is returned. (See *NODATA* on page 119).
ON	OFF	The base date is returned (December 31, 1900, for dates with YMD or YYMD format; or January 1901, for dates with YM, YYM, YQ, or YYQ format).
	ON	The value of the NODATA character (a period, by default) is returned.

OFF

Does not display a row of data that contains an invalid date format and generates an error message. OFF is the default value.

ALTBACKPERLINE

The ALTBACKPERLINE attribute alternates the background color by line for reports that use positioned drivers, for example PDF, DHTML, PPT, and PPTX. This enables you to wrap a long field value, and alternate the background color of each line for that value, independent of borders. In order to apply alternating background color per line, you need to explicitly add the SET ALTBACKPERLINE=ON command to procedures that use WRAP.

The syntax is:

```
SET ALTBACKPERLINE = {ON|OFF}
```

where:

ON

Alternates background color by line.

OFF

Alternates background color by row. This is the default value.

ASNAMES

The ASNAMES parameter controls the FIELDNAME attribute in a HOLD Master File. When an AS phrase is used in a TABLE request, the specified literal is used as a field name in a HOLD file. It also controls how field names are specified for the values of an ACROSS field when a HOLD file is created.

The syntax is:

```
SET ASNAMES = {ON|OFF|MIXED|FOCUS|FLIP}
```

where:

OFF

Does not use the literal specified in an AS phrase as a field name in HOLD files, and does not affect the way ACROSS fields are named.

ON

Uppercases the literal specified in an AS phrase and propagates it as the field name in the HOLD Master File. Creates names for ACROSS fields that consist of the AS name value concatenated to the beginning of the ACROSS field value and controls the way ACROSS fields are named in HOLD files of any format.

MIXED

Uses the literal specified in an AS phrase for the field name, retaining the case of the AS name, and creates names for ACROSS fields that consist of the AS name value concatenated to the beginning of the ACROSS field value.

FOCUS

Uses the literal specified in an AS phrase as the field name and controls the way ACROSS fields are named only in HOLD files in FOCUS format. FOCUS is the default value.

FLIP

Propagates the field names in the original Master File to the alias names in the HOLD Master File and the alias names in the original Master File to the field names in the HOLD Master File.

AUTOFIT

The AUTOFIT parameter automatically resizes HTML report output to fit its window or frame and HTML5 graphs to fit their containers.

The syntax is:

```
SET AUTOFIT = {OFF|ON|RESIZE}

ON {GRAPH|TABLE} SET AUTOFIT {OFF|ON|RESIZE}
```

where:

`OFF`

Respects the dimensions specified by the data and styles for TABLE or by the HAXIS and VAXIS parameters for HTML5 graphs.

`ON`

Always resizes HTML report output to fit its window or frame and HTML5 graph output to fit its container.

`RESIZE`

Applies to HTML5 graphs only. Respects the dimensions specified by the HAXIS and VAXIS parameters initially, but resizes the graph output if the container is resized.

AUTOINDEX

The AUTOINDEX parameter speeds data retrieval by automatically taking advantage of indexed fields or multi-dimensional indexes (MDI) in most cases where TABLE requests contain equality or range tests on those fields or dimensions. This applies only to FOCUS and XFOCUS data sources.

AUTOINDEX is never performed when the TABLE request contains an alternate file view, for example, TABLE FILE *filename.fieldname*. Indexed retrieval is not performed when the TABLE request contains BY HIGHEST or BY LOWEST phrases and AUTOINDEX is ON.

The syntax is:

```
SET AUTOINDEX = {ON|OFF}
```

where:

`ON`

Uses indexed retrieval when possible. ON is the default value.

OFF

Uses indexed retrieval only when explicitly specified using an indexed view, for example, TABLE FILE *filename.indexed-fieldname*.

AUTOPATH

The AUTOPATH parameter dynamically selects an optimal retrieval path for accessing a FOCUS data source by analyzing the data source structure and the fields referenced, and choosing the lowest possible segment as the entry point. Use AUTOPATH only if your field is not indexed.

The syntax is:

```
SET AUTOPATH = {ON|OFF}
```

where:

ON

Dynamically selects an optimal retrieval path. ON is the default value.

OFF

Uses sequential data retrieval. The end user controls the retrieval path through *filename.segname*.

AUTOSTRATEGY

The AUTOSTRATEGY parameter determines when FOCUS stops the search for a key field specified in a WHERE or IF test. When set to ON, the search ends when the key field is found, optimizing retrieval speed. When set to OFF, the search continues to the end of the data source.

The syntax is:

```
SET AUTOSTRATEGY = {ON|OFF}
```

where:

ON

Stops the search when a match is found. ON is the default value.

OFF

Searches the entire data source.

AUTOTABLEF

The AUTOTABLEF parameter avoids creating the internal matrix based on the features used in the query. Avoiding internal matrix creation reduces internal overhead costs and yields better performance.

The syntax is:

```
SET AUTOTABLEF = {ON|OFF}
```

where:

ON

Does not create an internal matrix. ON is the default value.

OFF

Creates an internal matrix.

BASEURL

The BASEURL parameter specifies a default location where your browser searches for relative URLs referenced in the HTML documents created by FOCUS. This allows you to hyperlink to files, images, and Java files using only the file names rather than the full URLs.

The syntax is:

```
SET BASEURL = url
```

where:

url

Is the fully qualified directory in which additional HTML files, graphics files, and Java applet class files reside. If the URL represents a web server address, it must begin with http:// and end with a slash (/).

BINS

The BINS parameter specifies the number of pages of memory (blocks of 4,096 bytes) used for data source buffers.

The syntax is:

```
SET BINS = n
```

where:

n

Is the number of pages used for data source buffers. Valid values are 13 to 64. 64 is the default value. This is the recommended value.

BLANKINDENT

To clarify relationships within an FML hierarchy, the captions (titles) of values are indented at each level. You can use the BLANKINDENT parameter in an HTML, PDF, or PostScript report to specify the indentation between each level the hierarchy. You can use the default indentation for each hierarchy level or choose your own indentation value. To print indented captions in an HTML report, you must set the BLANKINDENT parameter to ON or to a number.

In PDF and PS reports, you may need to adjust the widths of columns to accommodate the indentations.

The syntax is:

```
SET BLANKINDENT = {ON|OFF|n}
```

where:

ON

Indents FML hierarchy captions 0.125 units for each space normally displayed before the caption. For child levels in an FML hierarchy, it indents 0.125 units for each space that would normally display between this line and the line above it.

OFF

Turns off indentations for FML hierarchy captions in an HTML report. For other formats, uses the default indentation of two spaces. OFF is the default value.

n

Is an explicit measurement in the unit of measurement defined by the UNITS parameter. This measurement is multiplied by the number of spaces that would normally display before the caption. For child levels in an FML hierarchy, it indents *n* units for each space that would normally display between this line and the line above it. The default number of spaces is two. Zero (0) produces the same report output as OFF. Negative values for n are not supported. They generate the following message, and the request processes as if BLANKINDENT=OFF:

```
VALID VALUES ARE OFF, ON OR A POSITIVE NUMBER (IN CURRENT UNITS)
```

BOTTOMMARGIN

The BOTTOMMARGIN parameter sets the StyleSheet bottom boundary for report contents on a page.

This parameter applies only to PostScript and PS report formats.

The syntax is:

```
SET BOTTOMMARGIN = {n|.250}
```

where:

n

Is the bottom margin, in inches, for report contents on a page. 0.25 inches is the default value.

BUSDAYS

The BUSDAYS parameter specifies which days are considered business days and which days are not if, your business does not follow the traditional Monday through Friday week.

The syntax is:

```
SET BUSDAYS = {week|_MTWTF_}
```

where:

week

Is SMTWTFS, representing the days of the week. Any day that you do not want to designate as a business day must be replaced with an underscore in the designated place for that day.

If a letter is not in its correct position, or if you replace a letter with a character other than an underscore, you receive an error message. _MTWTF_ is the default value.

BYDISPLAY

Within a sort group, the sort field value displays only on the first line of the rows or leftmost column of the columns for its sort group. However, you can display the appropriate BY or ACROSS field on every row in a report using the SET BYDISPLAY command. Although SET BYDISPLAY is supported for all output formats, it is especially important for making your report output more usable by Excel, which cannot sort columns properly when they have blank values in some rows.

This feature may enable you to avoid specifying the sort field twice, once as a display field and once for sorting (with the NOPRINT option).

The syntax is:

```
SET BYDISPLAY = {OFF|ON|BY|ACROSS|ALL}
```

where:

`OFF`

Displays a BY field value only on the first line or column of the report output for the sort group and on the first line or column of a page. OFF is the default value.

`ON or BY`

Displays the associated BY field value on every line of report output produced. BY is a synonym for ON.

`ACROSS`

Displays the relevant ACROSS field value on every column of report output produced.

`ALL`

Displays the relevant BY field value on every line of report output and the relevant ACROSS field value on every column of report output.

BYPANEL

The BYPANEL parameter applies only to HOTSCREEN.

It controls the repetition of BY fields on panels. When BYPANEL is specified, the maximum number of panels is 99. When BYPANEL is OFF, the maximum number of panels is four.

The syntax is:

```
SET BYPANEL = option
```

where:

option

Is one of the following:

`ON` repeats the sort field values on each report panel.

`OFF` does not repeat sort field values on each report panel. Fields are displayed only on the first panel, and columns may split between panels. This value is the default.

`0` does not repeat sort field values on each report panel, and columns do not split between panels.

n repeats *n* columns of sort fields on each report panel. The value for *n* can be equal to or less than the total number of sort fields specified in the request.

CACHE

The CACHE parameter controls the number of cache pages to be allocated. This command cannot be used with ON TABLE SET.

Stores 4K FOCUS data source pages in memory and buffers them between the data source and BINS.

When a procedure calls for a read of a data source page, FOCUS first searches BINS, then cache memory, and then the data source on disk. If the page is found in cache, FOCUS does not have to perform an I/O to disk.

When a procedure calls for a write of a data source page, the page is written from BINS to disk. The updated page is also copied into cache memory so that the cache and disk versions remain the same. Unlike reads, cache memory does not save disk I/Os for write procedures.

FOCSORT pages are also written to cache. When the cache becomes full, they are written to disk. For optimal results, set cache to hold the entire data source plus the size of FOCSORT for the request. To estimate the size of FOCSORT for a given request, issue the ? STAT command, then add the number of SORTPAGES listed to the number of data source pages in memory. Issue a SET CACHE command for that amount. If cache is set to 50, 50 4K pages of contiguous storage are allocated to cache.

To clear the CACHE setting, issue a SET CACHE = *n* command. This command flushes the buffer (everything in cache memory is lost).

The syntax is:

```
SET CACHE = {0|n}
```

where:

0

Allocates no space to cache, which is inactive. 0 is the default value.

n

Is the number of 4K pages of contiguous storage allocated to cache memory. The minimum is two pages. The maximum is determined by the amount of memory available. If HiperFOCUS is activated, the default cache size is 256 pages (1MB) and the cache is placed in a hiperspace.

CARTESIAN

The CARTESIAN parameter applies to requests containing PRINT or LIST.

It generates a report containing all combinations of non-related data instances in a multi-path request. ACROSS cancels this parameter.

The syntax is:

```
SET CARTESIAN = {ON|OFF}
```

where:

ON

Generates a report with non-related records.

OFF

Disables the Cartesian product. OFF is the default value.

CDN

The CDN parameter specifies punctuation used in numeric notation.

Continental Decimal Notation (CDN) is supported for output in TABLE requests. It is not supported in DEFINE or COMPUTE commands.

The syntax is:

```
SET CDN = option
```

where:

option

Is one of the following:

DOTS_COMMA or ON uses CDN. Sets the decimal separator as a comma and the thousands separator as a period. For example, the number 3,045,000.76 is represented as 3.045.000,76. ON should be used for Germany, Denmark, Italy, Spain, and Brazil.

Note: Numeric parameters that use CDN ON must be separated by a comma followed by a space in calls to functions.

COMMAS_DOT or OFF turns CDN off. For example, the number 3,045,000.76 is represented as 3,045,000.76. OFF is the default value. OFF should be used for the USA, Canada, Mexico, and the United Kingdom.

SPACES_COMMA or SPACE sets the decimal point as a comma, and the thousands separator as a space. For example, the number 3,045,000.76 is represented as 3 045 000,76. SPACE should be used for France, Norway, Sweden, and Finland.

SPACES_DOT or SPACEP sets the decimal point as a period and the thousands separator as a space. For example, the number 3,045,000.76 is represented as 3 045 000.76.

QUOTES_COMMA or QUOTE sets the decimal point as a comma and the thousands separator as an apostrophe. For example, the number 3,045,000.76 is represented as 3'045'000,76. QUOTE should be used for Switzerland.

QUOTES_DOT or QUOTEP sets the decimal point as a period and the thousands separator as an apostrophe. For example, the number 3,045,000.76 is represented as 3'045'000.76.

Note: If the display format of a report is Excel 2000 or later, Continental Decimal Notation is controlled by the settings on the computer. That is, numbers in report output are formatted according to the convention of the locale (location) set in regional or browser language options.

CENT-ZERO

The CENT-ZERO parameter displays a leading zero in decimal-only numbers. The setting of CDN determines whether a decimal point or comma is the decimal separator.

The syntax is:

```
SET CENT-ZERO = {ON|OFF}
```

where:

ON

Displays fractions with a leading zero. The fraction is preceded by either a decimal point or comma, depending on the CDN setting.

OFF

Does not display a leading zero. The fraction is preceded by either a decimal point or comma, depending on the CDN setting. OFF is the default value.

CNOTATION

Column notation assigns a sequential column number to each column in the internal matrix created for a report request. You can use column notation in COMPUTE and RECAP commands to refer to these columns in your request.

Because column numbers refer to columns in the internal matrix, they are assigned after retrieval and aggregation are completed. Columns not actually displayed on the report output may exist in the internal matrix. For example, calculated values used in the request generate one or more columns in the internal matrix. Fields with the NOPRINT option take up a column in the internal matrix, and a reformatted field generates an additional column for the reformatted value. Certain RECAP calculations, such as FORECAST or REGRESS generate multiple columns in the internal matrix.

BY fields are not assigned column numbers but, by default, every other column in the internal matrix is assigned a column number, which means that you have to account for all of the internally generated columns if you want to refer to the appropriate column value in your request. You can change this default column assignment behavior with the SET CNOTATION=PRINTONLY command, which assigns column numbers only to columns that display on the report output, or the SET CNOTATION=EXPLICIT command, which assigns column numbers to columns that are referenced in the request.

The syntax is:

```
SET CNOTATION={ALL|PRINTONLY|EXPLICIT}
```

where:

ALL

Assigns column reference numbers to every column in the internal matrix. ALL is the default value.

PRINTONLY

Assigns column reference numbers only to columns that display on the report output.

EXPLICIT

Assigns column reference numbers to all fields referenced in the request, whether it is displayed or not.

Note: This setting is not supported in an ON TABLE phrase.

COLLATION

The COLLATION parameter controls the ordering and matching of all language elements that involve comparison of two alphanumeric values.

The syntax is:

```
SET COLLATION = {BINARY|SRV_CI|SRV_CS|CODEPAGE}
```

where:

BINARY

Bases the collation sequence on binary values.

SRV_CI

Bases collation sequence on the LANGUAGE setting, and is case-insensitive.

SRV_CS

Bases collation sequence on the LANGUAGE setting, and is case-sensitive.

CODEPAGE

Bases collation sequence on the code page in effect, and is case-sensitive. CODEPAGE is the default value.

In most cases, CODEPAGE is the same as BINARY. The only differences are for Danish, Finnish, German, Norwegian, and Swedish in an EBCDIC environment.

COMPMISS

When a field is reformatted in a request (for example, SUM field/format), an internal COMPUTE field is created to contain the reformatted field value and displayed on the report output. If the original field has a missing value, that missing value can be propagated to the internal field by setting the COMPMISS parameter ON. If the missing value is not propagated to the internal field, it displays a zero (if it is numeric) or a blank (if it is alphanumeric). If the missing value is propagated to the internal field, it displays the missing data symbol on the report output.

The syntax is:

```
SET COMPMISS = {ON|OFF}
```

where:

ON

Propagates a missing value to a reformatted field.

OFF

Displays a blank or zero for a reformatted field. OFF is the default value.

COMPOUND

The COMPOUND parameter, which is used to create compound reports, combines multiple reports into a single PDF or PostScript (PS) file. Using COMPOUND enables you to concatenate reports with styled report formats (PDF, HTML, Power Point, Excel). You can also embed image files, including graphs saved as images, in a compound report.

For more information about creating compound reports, see the *Creating Reports* manual.

For a compound report that may contain different report types, the syntax is:

```
SET COMPOUND = {OPEN|CLOSE} [NOBREAK]
```

or

```
ON TABLE SET COMPOUND {OPEN|CLOSE}
```

Note that when you are using this syntax, you must also include the following code to identify the display format of each of the different reports to be concatenated:

```
ON TABLE {PCHOLD|HOLD|SAVE} [AS name] FORMAT formatname
```

If all of the reports in the compound set are of the same type, either PDF or PS, the syntax is:

```
ON TABLE {PCHOLD|HOLD|SAVE} [AS name] FORMAT {PDF|PS} {OPEN|CLOSE} [NOBREAK]
```

where:

name

Is the name of the generated file. The name is taken from the first request in the compound report. If no name is specified in the first report, the name HOLD is used.

formatname

Is the name of the styled report format. Valid formats include PDF, PS, HTML, PPT, and EXL2K.

OPEN

Is specified with the first report, and begins the concatenation process. A report that contains the OPEN attribute must be in PDF or PS format.

CLOSE

Is specified with the last report, and ends the concatenation process.

NOBREAK

Is an optional phrase that suppresses page breaks. By default, each report is displayed on a separate page. You can use NOBREAK selectively in a request to control which reports are displayed on the same page.

Note:

- You can save or hold the output from a compound report.
- Compound reports cannot be nested.
- Multi-pane reports cannot be used in a compound report.

COMPUTE

The COMPUTE parameter controls the compilation of calculations when a request is executed.

The syntax is:

```
SET COMPUTE = {COMPILED|OLD}
```

where:

COMPILED

Implements expression compilation at request run time, compiling only those expressions that are used in the request. COMPILED is the default value.

OLD

The value OLD has been deprecated and functions as COMPILED.

COUNTWIDTH

The COUNTWIDTH parameter expands the default format of COUNT fields from a five-byte integer to a nine-byte integer or a specified integer format supported in your operating environment.

The syntax is:

```
SET {COUNTWIDTH|LISTWIDTH} = {ON|OFF|n}
```

where:

ON

Expands the default format of COUNT fields from a five-byte integer to a nine-byte integer.

OFF

Does not expand the default format of COUNT fields from a five-byte integer to a nine-byte integer. OFF is the default value.

n

Enables you to specify a width for the COUNT field up to the maximum integer format supported in your operating environment.

CSSURL

The CSSURL parameter links an HTML report to an external cascading style sheet (CSS) file in order to style the report.

The syntax is:

```
SET CSSURL = link
```

where:

link

Is the URL location of the CSS file. This can be an absolute or relative link.

CURRENCY_DISPLAY

This parameter defines the position of the currency symbol relative to the monetary number.

The syntax is:

```
SET CURRENCY_DISPLAY = pos
```

where:

pos

Defines the position of the currency symbol relative to a number. The default value is *default*, which uses the position for the format and currency symbol in effect. Valid values are:

- **LEFT_FIXED.** The currency symbol is left-justified preceding the number.
- **LEFT_FIXED_SPACE.** The currency symbol is left-justified preceding the number, with at least one space between the symbol and the number.
- **LEFT_FLOAT.** The currency symbol precedes the number, with no space between them.
- **LEFT_FLOAT_SPACE.** The currency symbol precedes the number, with one space between them.
- **TRAILING.** The currency symbol follows the number, with no space between them.
- **TRAILING_SPACE.** The currency symbol follows the number, with one space between them.

Note: This setting is not supported with FORMAT EXL2K report output.

CURRENCY_ISO_CODE

This parameter defines the ISO code for the currency symbol to use.

The syntax is:

```
SET CURRENCY_ISO_CODE = iso
```

where:

iso

Is a standard three-character currency code such as USD for US dollars or JPY for Japanese yen. The default value is *default*, which uses the currency code for the configured language code.

Note: This setting is not supported with FORMAT EXL2K report output.

CURRENCY_PRINT_ISO

This parameter defines what will happen when the currency symbol cannot be displayed by the code page in effect, if the format of the field to be displayed includes the !C or :C extended currency symbol.

The syntax is:

```
SET CURRENCY_PRINT_ISO = {DEFAULT|ALWAYS|NEVER}
```

where:

DEFAULT

Replaces the currency symbol with its ISO code when the symbol cannot be displayed by the code page in effect. This is the default value.

ALWAYS

Always replaces the currency symbol with its ISO code.

NEVER

Never replaces the currency symbol with its ISO code. If the currency symbol cannot be displayed by the code page in effect, it will not be printed at all.

Note:

- Using a Unicode environment allows the printing of all currency symbols, otherwise this setting is needed.
- This parameter is not supported with FORMAT EXL2K report output.

CURRSYMB

The CURRSYMB parameter specifies a symbol used to represent currency when a numeric format specification uses the M or N display options. The default currency symbol depends on the code page being used.

The syntax is:

```
SET CURRSYMB = symbol
```

where:

symbol

Is any printable character or a supported currency code.

Note: In order to specify a dollar sign as the character, you must enclose it in single quotation marks (').

- USD or '$' specifies U.S. dollars.
- GBP specifies the British pound.
- JPY specifies the Japanese yen.
- EUR specifies the Euro.
- NIS specifies the Israeli new sheqel.

CURSYM_D

The CURSYM_D parameter specifies the characters used to represent currency when a numeric format specification uses the !D, :D, !d, or :d display options which, by default, display a floating (D) or fixed (d) dollar sign to the left of the number.

The syntax is:

```
SET CURSYM_D = currsym
```

where:

currsym

Specifies up to four printable characters.

CURSYM_E

The CURSYM_E parameter specifies the characters used to represent currency when a numeric format specification uses the !E, :E, !e, or :e display options which, by default, display a floating (E) or fixed (e) euro symbol to the left of the number.

The syntax is:

```
SET CURSYM_E = currsym
```

where:

currsym

Specifies up to four printable characters.

CURSYM_F

The CURSYM_F parameter specifies the characters used to represent currency when a numeric format specification uses the !F or :F display option which, by default, places a floating euro symbol to the right of the number. This command supports adding a blank space between the number and the currency symbol.

The syntax is:

```
SET CURSYM_F = currsym
```

where:

currsym

Specifies up to four printable characters. If the characters include a blank space, they must be enclosed in single quotation marks.

CURSYM_G

The CURSYM_G parameter specifies the characters used to represent currency when a numeric format specification uses the !G or :G display option which, by default, places a floating dollar sign to the right of the number. This command supports adding a blank space between the number and the currency symbol.

The syntax is:

```
SET CURSYM_G= currsym
```

where:

currsym

Specifies up to four printable characters. If the characters include a blank space, they must be enclosed in single quotation marks.

CURSYM_L

The CURSYM_L parameter specifies the characters used to represent currency when a numeric format specification uses the !L, :L, !l, or :l display options which, by default, display a floating (L) or fixed (l) British pound symbol to the left of the number.

The syntax is:

```
SET CURSYM_L = currsym
```

where:

currsym

Specifies up to four printable characters.

CURSYM_Y

The CURSYM_Y parameter specifies the characters used to represent currency when a numeric format specification uses the !Y, :Y, !y, or :y display options which, by default, display a floating (Y) or fixed (y) Japanese yen or Chinese yuan symbol to the left of the number.

The syntax is:

```
SET CURSYM_Y = currsym
```

where:

currsym

Specifies up to four printable characters.

DATE_ORDER

This parameter defines the order of date components for display.

The syntax is:

```
SET DATE_ORDER = {DEFAULT|DMY|MDY|YMD}
```

where:

DEFAULT

Respects the original order of date components. This is the default value.

DMY

Displays all dates in day/month/year order.

MDY

Displays all dates in month/day/year order.

YMD

Displays all dates in year/month/day order.

Note:

- DATE_ORDER overrides the specified date order for all date and date-time displays. To limit the scope to a request when using DATE_ORDER, use the ON TABLE SET phrase.

- To use this setting with the Dialogue Manager system variables, (for example, &DATE, &TOD, &YMD, &DATEfmt, and &DATXfmt) append the suffix .DATE_LOCALE to the system variable. This allows system variables that are localized to coexist with non-localized system variables.
- This parameter is not supported with FORMAT EXL2K report output.

DATE_SEPARATOR

This parameter defines the separator for date components for display.

The syntax is:

```
SET DATE_SEPARATOR = separator
```

where:

separator
Can be one of the following values.

- **DEFAULT**, which respects the separator defined by the USAGE format of the field.
- **SLASH**, which uses a slash (/) to separate date components.
- **DASH**, which uses a dash (-) to separate date components.
- **BLANK**, which uses a blank to separate date components.
- **DOT**, which uses a dot (.) to separate date components.
- **NONE**, which does not separate date components.

Note:

- DATE_SEPARATOR overrides the date separator for all date and date-time displays unless they include a translation display option (T,Tr, t, or tr), in which case the specified separator is produced.
- To use this setting with the Dialogue Manager system variables, (for example, &DATE, &TOD, &YMD, &DATEfmt, and &DATXfmt) append the suffix .DATE_LOCALE to the system variable. This allows system variables that are localized to coexist with non-localized system variables.
- This parameter is not supported with FORMAT EXL2K report output.

DATEDISPLAY

The DATEDISPLAY parameter controls the display of a base date. Previously, TABLE always displayed a blank when a date read from a file matched the base date or a field with a smart date format had the value 0. The following shows the base date for each supported date format:

Format	Base Date
YMD and YYMD	1900/12/31
YM and YYM	1901/01
YQ and YYQ	1901/Q1
JUL and YYJUL	00/365 and 1900/365

Note: You cannot set DATEDISPLAY with the ON TABLE command.

The syntax is:

```
SET DATEDISPLAY = {ON|OFF}
```

where:

ON

Displays the base date if the data is the base date value.

OFF

Displays a blank if the date is the base date value. OFF is the default value.

DATEFNS

The DATEFNS parameter activates year 2000-compliant versions of date functions.

The syntax is:

```
SET DATEFNS = {ON|OFF}
```

where:

ON

Loads the year 2000-compliant versions of functions supplied by Information Builders.

OFF

This value is no longer functional, and operates as ON.

DATEFORMAT

The DATEFORMAT parameter specifies the order of the date components (month/day/year) when date-time values are entered in the formatted string and translated string formats. It makes the input format of a value independent of the format of the variable to which it is being assigned.

The syntax is:

```
SET DATEFORMAT = datefmt
```

where:

datefmt

Can be one of the following: MDY, DMY, YMD, or MYD. MDY is the default value for the U.S. English format.

DATETIME

The DATETIME parameter sets time and date in reports. This command is useful for determining (statically or dynamically) exactly when your report was run. You can display the DATETIME value using any FOCUS date variable (for example, YMD, MDY, TOD). If DATETIME is not set, the behavior of the FOCUS date variables remain the same.

The syntax is:

```
SET DATETIME = option
```

where:

option

Is one of the following:

- **STARTUP,** which is the time and date when you began your session. STARTUP is the default value.
- **CURRENT|NOW,** which changes each time it is interrogated. For example, if your batch job starts before midnight at 11:59 P.M., it will not complete until the next day. If DATETIME is set to NOW|CURRENT, any reference to the variable gives the current date, not the date when the job started.

- **RESET,** which freezes the date and time of the current run for the rest of the session or until another SET DATETIME command is issued.

DB_INFILE

The SET DB_INFILE command controls whether the expression generated by the DB_INFILE function for use against a relational data source is optimized.

The syntax is:

```
SET DB_INFILE = {DEFAULT|EXPAND_ALWAYS|EXPAND_NEVER}
```

where:

DEFAULT

Enables DB_INFILE to create a subquery if its analysis determines that it is possible. This is the default value.

EXPAND_ALWAYS

Prevents DB_INFILE from creating a subquery and, instead, expands the expression into IF and WHERE clauses in memory.

EXPAND_NEVER

Prevents DB_INFILE from expanding the expression into IF and WHERE clauses in memory and, instead, attempts to create a subquery. If this is not possible, a FOC32585 message is generated and processing halts.

DBACSENSITIV

When a DBA or user issues the SET USER, SET PERMPASS or SET PASS command, this user ID is validated before they are given access to any data source whose Master File has DBA attributes. The password is also checked when encrypting or decrypting a FOCEXEC.

The SET DBACSENSITIV command determines whether the password is converted to uppercase prior to validation.

The syntax is:

```
SET DBACSENSITIV = {ON|OFF}
```

where:

ON

Does not convert passwords to uppercase. All comparisons between the password set by the user and the password in the Master File or FOCEXEC are case-sensitive.

OFF

Converts passwords to uppercase prior to validation. All comparisons between the password set by the user and the password in the Master File or FOCEXEC are not case-sensitive. OFF is the default value.

DBAJOIN

The DBAJOIN parameter controls where DBA restrictions are treated as WHERE conditions in the report request or are added as join conditions.

```
SET DBAJOIN = {OFF|ON}
```

where:

OFF

Treats DBA restrictions as WHERE filters in the report request. OFF is the default value.

ON

Treats DBA restrictions as join conditions.

DBASOURCE

The DBASOURCE parameter determines which security attributes are used to grant access to multi-file structures. By default, access restrictions are based on the host file in a JOIN structure or the last file in a COMBINE structure. If you set the DBASOURCE parameter to ALL, access restrictions from all files in a JOIN or COMBINE structure will be enforced.

All files in the JOIN or COMBINE structure must have the same DBA password. If the DBA attributes are not the same, there will be no way to access the structure.

The SET DBASOURCE command can only be issued one time in a session or connection. Any attempt to issue the command additional times will be ignored. If the value is set in a profile such as FOCPARM, no user can change it at any point in the session.

When DBASOURCE=ALL:

- In a TABLE request against a JOIN structure, access to a cross-reference file or segment is allowed only if the user has at least read access to each file in the structure.
- In a MODIFY COMBINE structure, the user must have a minimum of READ access to all files in the structure. The user requires WRITE, UPDATE, or READ/WRITE access to a file in the structure when an INCLUDE, DELETE, or UPDATE request is issued against that file.

When DBASOURCE=HOST:

- In a TABLE request, the user needs read access to the host file in the JOIN structure. All security limitations come from the host file. Note that you can create and activate a DBAFILE in order to enforce security restrictions from all files in the structure.
- In a MODIFY procedure, the user needs write access to the last file in the COMBINE structure. All security limitations come from the restrictions in the last file in the structure. Note that you can create and activate a DBAFILE in order to enforce security restrictions from all files in the structure.

The syntax is:

```
SET DBASOURCE = {HOST|ALL}
```

where:

HOST

Enforces access restrictions only from the host file in a JOIN structure or the last file in a COMBINE structure unless a DBAFILE is used to enforce access restrictions to other files in the structure. HOST is the default value.

ALL

Requires the user to have read access to every file in a JOIN or COMBINE structure. The user needs W, U, or RW access to a file in a COMBINE structure when an INCLUDE, UPDATE, or DELETE command is issued against that file.

DEFCENT

The DEFCENT parameter defines a default century globally or on a field-level for an application that does not contain an explicit century. DEFCENT is used in conjunction with YRTHRESH to interpret the current century according to the given values. When assigned globally, the time span created by these parameters applies to every 2-digit year used by the application unless you specify file-level or field-level values. (See *YRTHRESH* on page 149.)

Note: This same result can be achieved by including the FDEFCENT and FYRTHRESH attributes in the Master File.

The syntax is:

```
SET DEFCENT = {cc|19}
```

where:

cc

Is the default century. 19 is the default value if one is not supplied. The value *cc* defaults to 19, for the twentieth century.

DEFECHO

The DEFECHO parameter defines a default value for the &ECHO variable.

The syntax is:

```
SET DEFECHO = {OFF|ON|ALL|NONE}
```

where:

OFF

Establishes OFF as the default value for &ECHO. OFF is the default value.

ON

Establishes ON as the default value for &ECHO.

ALL

Establishes ALL as the default value for &ECHO.

NONE

Prevents procedure code from being displayed (echoed). Once the value of DEFECHO or &ECHO has been set to NONE, it cannot be changed during the session or connection.

DEFINES

The DEFINES parameter increases the speed of calculations in virtual fields by compiling virtual fields into machine code.

The syntax is:

```
SET DEFINES = {COMPILED|OLD}
```

where:

COMPILED

Implements expression compilation at request run time, compiling only those DEFINEs that are used in the request. COMPILED is the default value.

OLD

The value OLD has been deprecated and functions as COMPILED.

DIRECTHOLD

The DIRECTHOLD parameter creates a HOLD file in FOCUS format directly, without an internally generated MODIFY procedure and an intermediate sequential file.

The syntax is:

```
SET DIRECTHOLD = {ON|OFF}
```

where:

ON

Creates a FOCUS HOLD file directly without an intermediate sequential file and MODIFY procedure. ON is the default value and the only value supported. OFF is allowed for backward syntax compatibility, but it operates the same way as ON.

OFF

OFF is allowed for backward syntax compatibility, but it operates the same way as ON.

DMH_LOOPLIM

The DMH_LOOPLIM parameter sets the maximum number of Dialogue Manager loop iterations allowed, using -REPEAT or -GOTO commands.

The syntax is:

```
SET DMH_LOOPLIM = n
```

where:

n

Sets the maximum number of loop iterations allowed. The default value is zero (0), which does not limit the number of loop iterations.

DMH_LOOPLIM should be set high enough to run your existing reports and procedures without error for your entire session. It is recommended that if you set this parameter, you set it in a profile.

DMH_STACKLIM

The DMH_STACKLIM parameter sets the maximum number of lines allowed in FOCSTACK.

The syntax is:

```
SET DMH_STACKLIM = n
```

where:

n

Sets the maximum number of lines allowed in FOCSTACK. The default value is zero (0), which does not limit the number of stacked commands.

DMH_STACKLIM should be set high enough to run your existing reports and procedures without error for your entire session. It is recommended that if you set this parameter, you set it in a profile.

DMPRECISION

The DMPRECISION parameter specifies numeric precision in Dialogue Manager -SET commands.

Without this setting, results of numeric calculations are returned as integer numbers, although the calculations themselves employ double-precision arithmetic. To return a number with decimal precision without this setting, you have to enter the calculation as input into subroutine FTOA, where you can specify the number of decimal places returned.

The syntax is:

```
SET DMPRECISION = {OFF|n}
```

where:

OFF

Specifies truncation without rounding after the decimal point. OFF is the default value

n

Is a positive number from 0-9, indicating the point of rounding. Note that n=0 results in a rounded integer value.

DROPBLNKLINE

The DROPBLNKLINE parameter suppresses blank lines around subtotals, subheadings, and subfootings when formatting a report for output. In addition, certain data lines may be blank and appear as blank lines on the report output. You can eliminate these blank lines from the report output using the SET DROPBLNKLINE=ON command.

This setting does not apply to the following output formats: HOLD/PCHOLD/SAVE formats ALPHA, INTERNAL, BINARY, COM, COMT, COMMA, TAB, TABT, FIX, DFIX, all DBMS, VSAM, LOTUS, SYLK, DIF, FOCUS, and XFOCUS.

The syntax is:

```
SET DROPBLNKLINE = {OFF|ON|BODY|HEADING|ALL}
```

where:

OFF

Inserts system-generated blank lines as well as empty data lines. OFF is the default value.

ON|BODY

Removes system-generated blank lines within the body of the report, for example, before and after subheads. In addition, certain data lines that may be blank and appear as blank lines on the report output will be removed from the output. BODY is a synonym for ON.

HEADING

Removes the blank lines between headings and titles and between the report body and the footing. Works in positioned formats (PDF, PS, DHTML, PPT, PPTX) when a request has a border or backcolor StyleSheet attribute anywhere in the report.

ALL

Provides both the ON and HEADING behaviors.

DTSTRICT

The DTSTRICT parameter controls the use of strict processing. Strict processing checks date-time values when they are input by an end user, read from a transaction file, displayed, or returned by a subroutine to ensure that they represent a valid date and time. For example, a numeric month must be between 1 and 12, and the day must be within the number of days for the specified month.

The syntax is:

```
SET DTSTRICT = {ON|OFF}
```

where:

ON

Invokes strict processing. ON is the default value.

OFF

Does not invoke strict processing. Date-time components can have any value within the constraint of the number of decimal digits allowed in the field. For example, if the field is a two-digit month, the value can be 12 or 99, but not 115.

DUPLICATECOL

The DUPLICATECOL parameter reformats report requests that use multiple display commands, placing aggregated fields in the same column above the displayed field.

The syntax is:

```
SET DUPLICATECOL = {ON|OFF}
```

where:

ON

Displays the report with each field as a column. ON is the default value.

OFF

Displays the report with common fields as a row.

EMBEDDABLE

The EMBEDDABLE parameter controls the generation of document-level HTML tags (such as, <html>, <head>, <body>) in HTML5 chart output. This enables multiple HTML5 charts to be embedded in an HTML page.

The syntax is:

```
SET EMBEDDABLE = {OFF|ON}
```

where:

OFF

Generates complete HTML report output with document-level HTML tags. This is the default value.

ON

Generates report output in HTML format without document-level tags. This setting should be used when creating HTML5 graph output to be used with -HTMLFORM.

Note: SET EMBEDDABLE=ON also affects HTML report output and Java-based graph formats. For those formats, it is the equivalent of using HOLD FORMAT HTMTABLE.

EMPTYREPORT

The EMPTYREPORT parameter controls the output generated when a TABLE request retrieves zero records.

EMPTYREPORT is not supported with TABLEF or Excel. When a TABLEF or Excel request retrieves zero records, an empty report is generated.

Note: Using the IF TOTAL or WHERE TOTAL phrases when EMPTYREPORT is set to OFF may produce an empty report if there is no data that satisfies the TOTAL condition. This occurs because the test for report lines for EMPTYREPORT is applied before the TOTAL condition is applied.

The syntax is:

```
SET EMPTYREPORT={ANSI|ON|OFF}
```

where:

ANSI

Produces a single-line report and displays the missing data character or a zero if a COUNT is requested. In each case, &RECORDS will be 0, and &LINES will be 1.

If the SQL Translator is invoked, ANSI automatically replaces OFF as the default setting for EMPTYREPORT.

ON

Produces an empty report (column headings with no content). This was the default behavior in prior releases.

OFF

Produces no report output. OFF is the default value except for SQL Translator requests. When the SQL Translator is invoked, ANSI replaces OFF as the default setting for the EMPTYREPORT parameter, so the results are the same as for the ANSI setting.

The command can also be issued from within a request using:

```
ON TABLE SET EMPTYREPORT ANSI
```

EQTEST

The EQTEST parameter controls whether the characters $ and $* are treated as wildcard characters or normal characters in IF tests and WHERE tests that can be converted to one or more IF tests.

The syntax is:

```
SET EQTEST = {WILDCARD|EXACT}
```

where:

WILDCARD

Treats the $ and $* characters as wildcard characters. WILDCARD is the default value.

EXACT

Treats the $ and $* characters as normal characters, not wildcards, in IF tests and in WHERE tests that can be translated to IF tests.

ERROROUT

The ERROROUT parameter controls how a batch FOCUS job step responds to an error condition encountered in a procedure. This parameter cannot be set with the ON TABLE SET command.

When ERROROUT is set to ON, any error message generated terminates the job step and issues a return code of 8. Warning messages do not invoke this behavior. When ERROROUT is set to OFF, depending on the specific message, FOCUS determines whether FOCEXEC processing continues. Users can check a Dialogue Manager variable, such as &FOCERRNUM and issue the following command to terminate FOCUS and set *n* as the return code:

```
-QUIT FOCUS n

exit rc
```

Note: The ERROROUT setting is ignored in an interactive session.

The syntax is:

```
SET ERROROUT = {ON|OFF}
```

where:

ON

When an error message is generated in a batch FOCUS job step, ON sets the return code to 8 and terminates the job step.

In addition, the following message displays to inform the user why the program terminated:

```
Exiting due to Exit on Error
```

OFF

Does not set a return code or automatically terminate a job step or procedure in response to any error message. OFF is the default value.

ESTRECORDS

The ESTRECORDS parameter passes the estimated number of records to be sorted in the request.

ESTRECORDS can only be set with the ON TABLE SET command within the TABLE, MATCH, or GRAPH request.

The syntax is:

```
ON TABLE SET ESTRECORDS n
```

where:

n

Is the estimated number of records to be sorted.

EUROFILE

The EUROFILE parameter activates the data source that contains information for the currency you want to convert. This setting can be changed during a session to access a different currency data source. This parameter cannot be issued in a report request.

Note: You cannot set any additional parameters on the same line as EUROFILE. FOCUS ignores any other parameters specified on the same line.

The syntax is:

```
SET EUROFILE = {ddname|OFF}
```

where:

ddname

Is the name of the Master File for the currency data source you want to use. The ddname must refer to a read-only data source accessible by FOCUS. There is no default value.

OFF

Deactivates the current currency data source and removes it from memory.

EXCELSERVURL

The EXCELSERVURL parameter is needed for generating XLSX report output.

The syntax is:

```
SET EXCELSERVURL = LOCAL
```

EXL2KLANG

When included in the member NLSCFG in the ERRNLS PDS, the EXL2KLANG parameter specifies the language used for Microsoft® Excel requests. This language must be the same as the language of Excel on the browser machine in order to correctly display output.

You can code the SET EXL2KLANG command in a profile or procedure to override the setting in the errors file.

The syntax is:

```
EXL2KLANG = {language|ENG}
```

where:

language

Is the Excel language. Valid values are:

- **ENG** for English. ENG is the default value.
- **FRE** for French.
- **GER** for German.
- **JPN** for Japanese.
- **KOR** for Korean.
- **SPA** for Spanish.

EXL2KTXTDATE

The EXL2KTXTDATE parameter allows you to specify that translated dates should be sent as date values with format masks instead of text values.

The syntax is:

```
SET EXL2KTXTDATE = {TEXT|VALUE}
```

where:

TEXT

Passes date values that contain text to Excel 2000 as formatted text. TEXT is the default value.

VALUE

Passes the types of translated date values that contain text and are supported Excel date formats to Excel 2000 as standard date values with text format masks applied.

EXTAGGR

The EXTAGGR parameter uses external sorts to perform aggregation.

The syntax is:

```
SET EXTAGGR = {ON|OFF|NOFLOAT}
```

where:

ON

Allows aggregation by an external sort. ON is the default.

OFF

Does not allow aggregation by an external sort.

NOFLOAT

Allows aggregation if there are no floating data fields present.

EXTENDNUM

The EXTENDNUM parameter controls whether asterisks (*) display on report output when the value to be displayed does not fit in the allotted space on report output or whether the report column is extended to display the number.

The syntax is:

```
SET EXTENDNUM = {ON|OFF|AUTO}
```

where:

ON

Displays all numbers in full, regardless of the USAGE format defined.

OFF

Displays asterisks when the value does not fit in the space allotted by the USAGE format. This is the legacy behavior.

AUTO

Applies an ON or OFF setting based on output format and SQUEEZE settings, as shown in the following table.

Format	SQUEEZE Setting	EXTENDNUM
PDF, PS, DHTML, PPT, PPTX	ON	ON
	OFF	OFF
HTML, EXL2K, XLSX	N/A	ON
WP, other delimited formats	N/A	OFF

AUTO is the default value.

EXTHOLD

The EXTHOLD parameter enables you to create a HOLD file using an external sort.

The syntax is:

```
SET EXTHOLD = {ON|OFF}
```

where:

ON

Creates HOLD files using an external sort. ON is the default value.

OFF

Does not create HOLD files using an external sort.

EXTRACT

The EXTRACT parameter activates Structured HOLD Files for a request.

This parameter is only supported in a TABLE or TABLEF request using an ON TABLE phrase.

The syntax is:

```
ON TABLE SET EXTRACT = {ON|*|OFF}
```

where:

ON

Activates Structured HOLD Files for this request and extracts all fields mentioned in the request.

*

Activates Structured HOLD Files for this request and indicates that a block of extract options follows. For example, you can exclude specific fields from the Structured HOLD File.

OFF

Deactivates Structured HOLD files for this request. OFF is the default value.

EXTSORT

The EXTSORT parameter activates an external sorting feature for use with the TABLE, MATCH, and GRAPH commands.

If the report can be processed entirely in memory, external sorting does not occur.

In order to determine if the report can be processed in memory, issue the ? STAT query after the TABLE, MATCH, or GRAPH command, and check the value of the SORT USED parameter.

When StyleSheets are being used, an external sort does not work.

The syntax is:

```
SET EXTSORT = {ON|OFF}
```

where:

ON

Enables the selective use of an external sorting product to sort report. ON is the default value.

OFF

Uses the internal sorting procedure to sort reports.

FIELDNAME

The FIELDNAME parameter controls whether long and qualified field names are supported.

This command cannot be used with ON TABLE SET.

The syntax is:

```
SET FIELDNAME = {NEW|NOTRUNC|OLD}
```

where:

NEW

Supports long and qualified field names. NEW is the default value.

NOTRUNC

Supports long and qualified field names, but not unique truncations.

OLD

This parameter value is no longer operational. It now functions as the value NEW.

FILE[NAME]

The FILE[NAME] parameter specifies a file to be used, by default, in commands. When you set a default file name, you can use that file without specifying its name.

The syntax is:

```
SET FILE[NAME] = filename
```

where:

filename

Is a default file to be used in commands.

FILTER

The FILTER parameter assigns screening conditions to a data source for reporting purposes. It activates and deactivates filters.

The SET FILTER command is limited to one line. To activate more filters to fit on one line repeat the SET FILTER command.

The syntax is:

```
SET FILTER= {*|xx[yy zz]} IN {file|*} {ON|OFF}
```

where:

*

Denotes all declared filters. * is the default value.

xx, yy, zz

Are the names of filters as declared in the NAME = syntax of the FILTER FILE command.

file

Is the name of the data source you are assigning screening conditions to.

ON

Activates all (*) or specifically named filters for the data source or all data sources (*). The maximum number of filters you can activate for a data source is limited by the number of WHERE/IF phrases the filters contain, not to exceed the limit of WHERE/IF criteria in any single report request.

OFF

Deactivates (*) or specifically named filters for the data source or all data sources (*). OFF is the default value.

FIXRET[RIEVE]

FOCUS HOLD files support keyed retrieval from a fixed format sequential file, which can greatly reduce the I/Os incurred in reading extract files. The performance gains are accomplished by using the SEGTYPE= parameter in the Master File to specify that the BY fields in the request be used as a logical key for sequential files. The FIXRETRIEVE parameter allows you to stop the retrieval process when an equality test on this field holds true. This changes former behavior, as the interface previously read all of the records from the QSAM file and then passed them to FOCUS to apply the screening conditions when creating the final report.

The syntax is:

```
SET FIXRET[RIVE] = {ON|OFF}
```

where:

ON

Enables keyed retrieval. ON is the default value.

OFF

Disables keyed retrieval.

FLOATMAPPING

SET FLOATMAPPING enables you to take advantage of decimal-based precision numbers available in DB2 and Oracle, and extends that functionality to all numeric processing for floating point numbers. With this processing, you gain both precision, including improved rounding, and enhanced performance.

The syntax is

```
SET FLOATMAPPING = {D|M|X}
```

where:

D

Uses the standard double-precision processing. This is the default value.

M

Uses a new internal format that provides decimal precision for double-precision floating point numbers up to 16 digits.

X

Uses a new internal format that provides decimal precision for double-precision floating point numbers up to 34 digits.

Note: If the field is passed to a HOLD file, the internal data types X or M data type will be propagated to the USAGE and ACTUAL formats in the HOLD Master File.

FOC144

The FOC144 parameter suppresses warning message FOC144, which reads:

```
 "Warning: Testing in Independent sets of Data."
```

The syntax is:

```
SET FOC144 = {NEW|OLD}
```

where:

NEW

Displays the FOC144 warning message. NEW is the default value.

OLD

Suppresses the FOC144 warning message.

FOCEXURL

The FOCEXURL parameter is used to generate HTML5 graph output from FOCUS

The syntax is:

```
SET FOCEXURL = path
```

where:

path

Is the location of the WebFOCUS Servlet. The default path for Servlet is /ibi_apps/WFServlet.

FOCFIRSTPAGE

The FOCFIRSTPAGE parameter assigns a page number to the first page of output.

The syntax is:

```
SET FOCFIRSTPAGE = {n|1|&FOCNEXTPAGE}
```

where:

n

Is the number to be assigned to the first page of output. Valid values are integers with one to six characters. 1 is the default value.

&FOCNEXTPAGE

Is a variable whose value is determined by the last page number used by the last report. Its value is one more than that number.

FOCSTACK

This setting is no longer needed, but has been left in the product so that existing applications that still include it continue to work. The FOCSTACK parameter specified the amount of memory, in thousands of bytes, used by FOCSTACK, the stack of FOCUS commands awaiting execution.

This command cannot be used with ON TABLE SET.

The syntax is:

```
SET FOCSTACK [SIZE] = {n|8}
```

where:

n

Is the maximum amount, in thousands of bytes, that can be used by FOCSTACK. The maximum value depends on your region size.

8

Allows 8000 bytes to be used by FOCSTACK. 8 is the default value.

FORMULTIPLE

You can use the same value of a FOR field in many separate rows whether alone, as part of a range, or in a calculation by including the following syntax before or within an FML request:

The syntax is:

```
SET FORMULTIPLE = {ON|OFF}
```

where:

ON

Enables you to reference the same value of a FOR field in more than one row in an FML request.

With FORMULTIPLE set to ON, a value retrieved from the data source is included on every line in the report output for which it matches the tag references.

OFF

Does not enable you to include the same value in multiple rows. OFF is the default value.

With FORMULTIPLE set to OFF, multiple tags referenced in any of these ways (OR, TO, *) are evaluated first for an exact reference or for the end points of a range, then for a mask, and finally within a range. For example, if a value is specified as an exact reference and then as part of a range, the exact reference is displayed. Note that the result is unpredictable if a value fits into more than one row whose tags have the same priority (for example, an exact reference and the end point of a range.)

HDAY

The HDAY parameter specifies the holiday file from which to retrieve dates that are designated as holidays for use with the date functions DATEDIF, DATEMOV, DATECVT, and DATEADD. The file must be named HDAY, followed by two to four characters.

To clear the holiday file, use

```
SET HDAY = OFF
```

The syntax is:

```
SET HDAY = xxxx
```

where:

xxxx

Are the letters in the name of the holiday file, named HDAY*xxxx*. This string must be between two and four characters long.

The default is no setting for this parameter.

HIDENULLACRS

The HIDENULLACRS parameter hides the display of ACROSS groups containing only null columns.

Hiding null ACROSS columns is supported for all styled output formats except for the EXL2K PIVOT and EXL2K FORMULA options. It is not supported for Active Technologies.

The syntax is:

```
SET HIDENULLACRS = {ON|OFF}
```

where:

ON

Hides columns with missing data in ACROSS groups within a BY-generated page break.

OFF

Does not hide columns. OFF is the default value.

HLDCOM_TRIMANV

The HLDCOM_TRIMANV parameter controls whether trailing blanks are retained in AnV fields in delimited output files.

The syntax is:

```
SET HLDCOM_TRIMANV = {OFF|ON}
```

where:

`OFF`

Retains trailing blanks in AnV fields when the output is held in a delimited format. OFF is the default value.

`ON`

Removes trailing blanks in AnV fields when the output is held in a delimited format.

HNODATA

The HNODATA parameter controls the missing data characters that are propagated to fields with the MISSING=ON attribute in HOLD FORMAT ALPHA files. Missing values in fields that do not have the MISSING=ON attribute are propagated to a HOLD file as blank (for alphanumeric fields) or zero (for numeric fields).

The syntax is:

```
SET HNODATA = {charstring|,$}
```

where:

`charstring`

Is a string of up to 12 characters propagated to a HOLD FORMAT ALPHA file for missing values in a field with the MISSING=ON attribute. A period (.) is the default value.

If the string is longer than the length of the field, the value stored in:

- An alphanumeric field is the leftmost character of the string.
- A numeric field is a blank string.

When an alphanumeric string other than the default value (the period) is used to populate a missing numeric field, a blank is inserted in the held field to prevent a format error when displaying the data. If you use the default HNODATA value, it is inserted in numeric fields. In this way, a request against the HOLD file can recognize missing data that was propagated to the HOLD file.

If a number with decimal places is specified for HNODATA and the field with missing data is integer, the value is rounded to a whole number and inserted. In a numeric field that supports decimal places, it is rounded and inserted with the correct number of decimal digits.

,$

Indicates that nothing should be placed in the field when there is missing data. This setting can be used to support null values in non-FOCUS data sources.

HOLDATTR

The HOLDATTR parameter controls which attributes from the original Master File are used in the HOLD Master File. This setting does not affect the way fields are named in the HOLD Master File.

Note: HOLDATTRS is a synonym for HOLDATTR.

The syntax is:

```
SET HOLDATTR = {ON|OFF|FOCUS|CUBE}
```

where:

ON

Includes the TITLE attribute from the original Master File in HOLD Master Files for HOLD files of any format. PROPERTY attributes are also propagated. The ACCEPT attribute is included in the HOLD Master File when the HOLD file is in FOCUS format.

OFF

Does not include the TITLE or ACCEPT attributes from the original Master File in the HOLD Master File.

FOCUS

Includes the TITLE and ACCEPT attributes in HOLD Master Files when the HOLD file is in FOCUS format. PROPERTY attributes are also propagated. FOCUS is the default value.

CUBE

Propagates folders and DV_ROLE attributes, as well as TITLE attributes to the HOLD Master File. It also propagates the field name as the alias value.

HOLDFORMAT

The HOLDFORMAT parameter determines the default format for HOLD files. This value can be overridden for an individual HOLD file by issuing the ON TABLE SET HOLD FORMAT command in a request.

The syntax is:

```
SET HOLDFORMAT = {BINARY|ALPHA}
```

where:

BINARY

Creates HOLD files in binary format. BINARY is the default value.

ALPHA

Creates HOLD files in ALPHA format.

HOLDLIST

The HOLDLIST parameter controls whether only displayed fields or all fields are included in the HOLD or PCHOLD file.

The syntax is:

```
SET HOLDLIST = {PRINTONLY|ALL|ALLKEYS|EXPLICIT}
```

where:

PRINTONLY

Includes only those fields in the HOLD or PCHOLD file that are specified in the report request.

ALL

Includes all fields referenced in a request in the HOLD or PCHOLD file, including both computed fields and fields referenced in a COMPUTE command. ALL is the default value. (OLD may be used as a synonym for ALL.)

Note: Vertical sort (BY) fields specified in the request with the NOPRINT option are not included in the HOLD file, even with SET HOLDLIST=ALL.

ALLKEYS

Includes all fields in the HOLD or PCHOLD file, including NOPRINTed BY fields.

EXPLICIT

Includes fields in the HOLD or PCHOLD file that are explicitly omitted from the report output using the NOPRINT option in the request, but does not include fields that are implicitly NOPRINTed. For example, if a field is reformatted in the request, two versions of the field exist, the one with the new format and the one with the original format, which is implicitly NOPRINTed.

HOLDMISS

The HOLDMISS parameter enables you to distinguish between missing data and default values of blank (for character data) or zero (for numeric data) in a HOLD file.

The syntax is:

```
SET HOLDMISS = {OFF|ON}
```

where:

OFF

Does not allow you to store missing data in a HOLD file. OFF is the default value.

ON

Enables you to store missing data in a HOLD file. When TABLE generates a default value for data not found, it generates missing values.

HOLDSTAT

The HOLDSTAT parameter includes comments and DBA information in HOLD Master Files. This information can be from the HOLDSTAT ERRORS file supplied by Information Builders, or a file specified by the user.

The syntax is:

```
SET HOLDSTAT = {ON|OFF|name}
```

where:

ON

Derives comments and DBA information from a HOLDSTAT file. In z/OS, this information is derived from the member HOLDSTAT in the PDS allocated to the ddname MASTER or ERRORS.

OFF

Does not include information from the HOLDSTAT file in the HOLD Master File. OFF is the default value.

name

Specifies a HOLDSTAT file, created by the end user, whose information is included in the HOLD Master File.

HTMLARCHIVE

The HTMLARCHIVE parameter packages HTML or DHTML reports together with image files into a single web archive document (.mht file). The only browser that supports this format for HTML is Internet Explorer.

The syntax is:

```
SET HTMLARCHIVE = {ON|OFF}
```

where:

ON

Packages HTML or DHTML reports together with image files into a single web archive document (.mht file).

OFF

Does not package multiple files into a single document. OFF is the default value.

HTMLCSS

The HTMLCSS parameter creates an internal Cascading Style Sheets command in the HTML display page.

The syntax is:

```
SET HTMLCSS = {ON|OFF}
```

where:

ON

Creates an internal CSS command in the HTML page that displays the report output.

OFF

Does not create an internal CSS command in the HTML page that displays the report output. OFF is the default value.

HTMLEMBEDIMG

The HTMLEMBEDIMG parameter activates an encoding mechanism that embeds images and graphs directly into an HTML or DHTML .htm file to ensure that all FOCUS reports can be accessed from any browser.

The syntax is:

```
SET HTMLEMBEDIMG =  {OFF|ON|AUTO}
```

where:

OFF

Does not affect the default behavior. If HTMLARCHIVE is set ON, .mht files are generated.

ON

Encodes images within the .htm file.

AUTO

Determines how to handle images based on the browser of the calling client. For clients in Internet Explorer, HTMLARCHIVE will be used to embed the images into an .mht file. For all other browsers, HTMLEMBEDIMG will encode the image information into an .htm file. If the displaying browser is unknown, AUTO will use the HTMLARCHIVE setting that is in effect.

HTMLENCODE

The HTMLENCODE parameter controls whether HTML tags are encoded when these tags are stored within the actual data or created using a DEFINE or COMPUTE command.

In a FOCUS report, HTMLENCODE=ON causes any text set in a string to be encrypted for transportation, and then decrypted to be displayed as written on a report. This is both for security and to ensure that special characters are displayed correctly.

The syntax is:

```
SET HTMLENCODE {ON|OFF}
```

where:

ON

Encodes the HTML output that is data. This setting disables the rendering of HTML tags within a browser when these tags are stored within the actual data or created using a DEFINE or COMPUTE command.

OFF

Disables HTML encoding. OFF is the default value.

Note: Because of the new format of the zipped XLSX files, native HTML symbols, such as a caret (<), cannot be supported as tag characters. For XLSX, unlike other output formats, HTMLENCODE defaults to ON. HTMLENCODE set to OFF will cause any data containing HTML tag characters to be omitted from the cell.

INDEX

The INDEX parameter determines the indexing scheme used for indexes. Indexes are fields specified with FIELDTYPE=I keywords in the Master Files. The OLD setting for INDEX is no longer supported, but has been left in the product so that applications that included it continue to work.

The syntax is:

```
SET INDEX[TYPE] = {NEW|OLD}
```

where:

NEW

Creates a binary tree index. NEW is the default value.

OLD

Creates a hash index.

JOIN_LENGTH_MODE (JOINLM)

The JOIN_LENGTH_MODE (JOINLM) parameter controls processing of equality joined field pairs for the record oriented Adapters (such as VSAM, DFIX, and FIX). There are two supported modes of handling compatible but not identical joined fields:

- **SQL compliance.** The JOIN command processor assures strict value equality of joined fields. Detected truncation of significant characters during host to cross-referenced conversion raises a *target not found* condition. A shorter host field value is extended to the length of cross-referenced field with non-significant characters according to the data type.
- **FOCUS reporting.** The JOIN command processor assures partial value equality of joined fields.
  - When joining a shorter to a longer field, a search range is created to find all cross-referenced values that are prefixed with the host value (partial key join).
  - When joining a longer to a shorter field, the host value is unconditionally truncated to the cross-referenced field length.

The syntax is:

```
SET JOIN_LENGTH_MODE = {SQL|RANGE}
```

where:

SQL

Sets SQL compliant mode. Assures strict equality of host and cross-referenced fields. This is the default value.

RANGE

Sets FOCUS reporting mode. Supports partial key joins.

JOINOPT

The JOINOPT parameter has two functions:

- **Correcting for lagging values with a unique join.** If a parent segment has two or more unique child segments so that each has multiple children, the report may incorrectly display a missing value. The remainder of the child values may then be misaligned in the report. These misaligned values are called lagging values. The JOINOPT parameter ensures proper alignment of your output by correcting for lagging values.
- **Enabling joins with data type conversion.** You can join two or more data sources containing different numeric data types. For example, you can join a field with a short packed decimal format to a field with a long packed decimal format, or a field with an integer format to a field with a packed decimal format. This provides enormous flexibility for creating reports from joined data sources.

The syntax is:

```
SET JOINOPT = {NEW|GNTINT|OLD}
```

where:

`NEW`

Corrects lagging values when a parent segment has multiple unique children. Also, enables joins with data type conversion.

`GNTINT`

Corrects lagging values when a parent segment has multiple unique children. Also, enables joins with data type conversion.

`OLD`

Does not correct lagging values or support joins with data type conversion. This is the default value.

KEEPDEFINES

The KEEPDEFINES parameter controls whether a virtual field created for a host or joined structure is retained after a JOIN command is run. This parameter applies when a DEFINE command precedes the JOIN command.

The syntax is:

```
SET KEEPDEFINES = {ON|OFF}
```

where:

ON

Retains the virtual field after a JOIN command is run.

OFF

Clears the virtual field after a JOIN command is run. OFF is the default value.

KEEPFILTERS

By default, filters defined on the host data source are cleared by a JOIN command. However, filters can be maintained when a JOIN command is issued, by issuing the SET KEEPFILTERS=ON command.

Setting KEEPFILTERS to ON reinstates filter definitions and their individual declared status after a JOIN command. The set of filters and virtual fields defined prior to each join is called a context (see your documentation on SET KEEPDEFINES and on DEFINE FILE SAVE for information about contexts as they relate to virtual fields). Each new JOIN or DEFINE FILE command creates a new context.

If a new filter is defined after a JOIN command, it cannot have the same name as any previously defined filter unless you issue the FILTER FILE command with the CLEAR option. The CLEAR option clears all filter definitions for that data source in all contexts.

When a JOIN is cleared, each filter definition that was in effect prior to the JOIN command and that was not cleared, is reinstated with its original status. Clearing a join by issuing the JOIN CLEAR join_name command removes all of the contexts and filter definitions that were created after the JOIN join_name command was issued.

The syntax is:

```
SET KEEPFILTERS = {OFF|ON}
```

where:

OFF

Does not preserve filters issued prior to a join. This is the default value.

ON

Preserves filters across joins.

LANG[UAGE]

The LANG[UAGE] parameter specifies the National Language Support (NLS) environment. It sets the language of error messages and can also be used to set the language of report titles if the Master File contains alternate language TITLE attributes. For more information, see the *Describing Data* manual.

The syntax is:

```
SET LANG[UAGE] = [LNG| ln]
```

where:

LNG

Is the 3-letter abbreviation used to specify a language.

ln

Is the 2-letter ISO code used to specify a language.

The abbreviations and ISO codes used to specify a language are shown in the following table.

Language Name (Code)	Displayed Language (GUI)	Language Abbreviation	Language ISO code
AMENGLISH or ENGLISH or UKENGLISH	English	AME or ENG or UKE	en
ARABIC	Arabic	ARB	ar
BALTIC	Lithuanian	BAL	lt
CZECH	Czech	CZE	cs
DANISH	Danish	DAN	da
DUTCH	Dutch	DUT	nl
FINNISH	Finnish	FIN	fi
FRENCH	French - Standard or Canadian	FRE	fr fc

Language Name (Code)	Displayed Language (GUI)	Language Abbreviation	Language ISO code
GERMAN	German - Standard or Austrian	GER	de at
GREEK	Greek	GRE	el
HEBREW	Hebrew	HEB or HEW	iw
ITALIAN	Italian	ITA	it
JAPANESE	Japanese-JIS or EUC	JPN or JPE	ja or je
KOREAN	Korean	KOR	ko
POLISH	Polish	POL	po
PORTUGUESE	Portuguese- Brazil or Portugal	POR	br pt
RUSSIAN	Russian	RUS	ru
S-CHINESE	Chinese-Simplified GB	PRC	zh
SPANISH	Spanish	SPA	es
SWEDISH	Swedish	SWE	sv
T-CHINESE	Chinese-Traditional Big-5	ROC	tw
THAI	Thai	THA	th
TURKISH	Turkish	TUR	tr

LAYOUTGRID

Displays a grid in the report output, which enables you to evaluate the correct placement of data and objects during your report design. This option is applicable only when using the PDF, PS, or DHTML report output.

The syntax is:

```
SET LAYOUTGRID = {ON|OFF}
```

where:

ON

Displays a grid in the report output.

OFF

Turns off the grid in the report output. OFF is the default value.

LEADZERO

Leading zeros are truncated in Dialogue Manager strings. The functions in FOCUS, when called in Dialogue Manager, may return a numeric result. If the format of the result is YMD and contains a 00 for the year, the 00 is truncated.

The syntax is:

```
SET LEADZERO = {ON|OFF}
```

where:

ON

Allows the display of leading zeros if present.

OFF

Truncates leading zeros if present. OFF is the default value.

LEFTMARGIN

The LEFTMARGIN parameter sets the StyleSheet left boundary for report contents on a page. This parameter applies to PostScript and PDF reports.

The syntax is:

```
SET LEFTMARGIN = {value|.250}
```

where:

value

Is the left boundary of report contents on a page. 0.250 inches is the default value.

LINES

The LINES parameter sets the maximum number of lines of printed output that appear on a page, from the heading to the footing.

It sets the maximum number of lines that appear on a logical page, from the heading at the top to the footing on the bottom. The value of LINES can range between 1 and 999999. For styled output formats, specify 999 or higher to generate continuous forms. When continuous forms are specified, but the output format has a physical page size (as is the case with PDF output), the column titles repeat at the top of the physical page, without page numbers. For unstyled output formats, specify 999999 for continuous forms.

If this value is less than the value set for PAPER, the difference provides a bottom margin. FOCUS never puts more lines on a page than the LINES parameter specifies, but may put less.

Note: When using SKIP-LINE in a report, always set LINES to at least one less than the value for PAPER. This avoids unintentional page beaks at the bottom of the page.

When the STYLESHEET parameter is in effect, the setting for LINES is ignored.

The syntax is:

```
SET LINES = {n|57}
```

where:

n

Is the maximum number of lines of printed output that appear on a page.

MATCHCOLUMNORDER

The MATCHCOLUMNORDER parameter controls how fields in MATCH FILE requests are sequenced in the MATCH output. The new default method groups fields across files with their sort field values in the resulting HOLD file regardless of their position within the MATCH request. The old legacy method appends them to the HOLD file record as they are referenced in the request.

The syntax is:

```
SET MATCHCOLUMNORDER = {GROUPED|UNGROUPED}
```

where:

GROUPED

Groups verb objects with their highest-level common sort keys. This can result in the fields being propagated to the HOLD file in a different order from the legacy process. This can affect a subsequent request against the MATCH results if that request uses the default alias names generated in the HOLD Master File. This typically occurred when fields with the same name, not used as keys, were merged together with MATCH. The advantage of the new technique is that it supports HOLD file formats such as FORMAT FOCUS that generate an associated Master File.

UNGROUPED

Does not group verb objects with their sort keys across files when laying out the resulting HOLD file record. Fields are appended to the HOLD file record as they are referenced in the request.

MAXDATAEXCPT

Data exceptions occur when data that is supposed to contain a numeric value is manipulated in ways unsupported by the architecture of the operating environment. You can change the number of data exceptions allowed before the session is terminated using the SET MAXDATAEXCPT command.

Note: SET MAXDATAEXCPT is functional on mainframe platforms only. All other platforms allow the syntax, but do not support the functionality.

If this command is issued in a TABLE request using the ON TABLE SET phrase, a new count is established for that request. The running session count is saved and is restored after the request executes.

The syntax is:

```
SET MAXDATAEXCPT={10|maxexcpt}
```

where:

maxexcpt

Is a one to four-digit number that represents how many data exceptions can occur before the session is terminated. 10 is the default value. The value zero (0) allows an unlimited number of data exceptions. The value one (1) terminates the session at the first data exception.

If MAXDATAEXCPT is changed in a request, a new count is established and the session counter is saved and then restored after the request executes. If you issue the command outside of a TABLE request, the running counter is reset to zero.

MAXLRECL

The MAXLRECL parameter defines the maximum record length for an external file that can be read. 0 is the default value. However, FOCUS can read a 12K lrecl by default. This may be set to a maximum of 64K. Note that the maximum length of the internal memory area for data fields is still 32K.

Note: MAXLRECL is character-based, not byte-based. In a Unicode environment, three bytes is used to represent each character on UNIX and Windows, and four bytes is used to represent each character on z/OS. If you are using a double-byte character set, each character uses two bytes.

The syntax is:

```
SET MAXLRECL = {n|0}
```

where:

n

Is the maximum record length for an external file with OCCURS segments. 0 is the default value.

MDICARDWARN

The MDICARDWARN parameter displays a warning message every time the cardinality in a dimension exceeds a specified value, offering you the chance to study the MDI build. When the number of equal values of the data in a dimension reaches a specified percent, a warning message is issued. In order for MDICARDWARN to be reliable, the data source should contain at least 100,000 records.

Note: In addition to the warning message, a number displays in brackets. This number is the least number of equal values for the dimension mentioned in the warning message text.

The syntax is:

```
SET = MDICARDWARN = n
```

where:

n

Is a percentage value from 0 to 50.

MDIENCODING

The MDIENCODING parameter enables retrieval of output from the MDI file without reading the data source.

FOCUS encodes indexed values any time a field or dimension of an MDI has a MAXVALUES attribute specified or is involved in a parent-child relationship. Encoded values are stored in the MDI file at rebuild time and can be retrieved and decoded with a TABLE request that specifies the MDIENCODING command. The MDIENCODING command allows the user to get output from the MDI file itself without having to read the data source.

The following rules apply to fields in a TABLE request that uses MDIENCODING:

- Only one MDI can be referred to at a time.
- Only dimensions that are part of the same parent-child hierarchy can be used simultaneously in a request. A dimension that is not part of a parent-child relationship can be used as the field in a request if it has a MAXVALUES attribute.

The syntax is:

```
SET MDIENCODING = {ON|OFF}
```

where:

`ON`

Enables retrieval of output from the MDI file without reading the data source.

`OFF`

Requires access of the data source to allow retrieval of MDI values.

Note: This command can only be issued in an ON TABLE phrase. It has no default value.

MDIPROGRESS

The MDIPROGRESS parameter displays messages about the progress of an MDI build. The messages show the number of data records accumulated for every *n* records inserted into the MDI as it is processed.

The syntax is:

```
SET MDIPROGRESS = {n|0}
```

where:

n

Is an integer greater than 1000, which displays a progress message for every *n* records accumulated in the MDI build. 100,000 is the default value.

0

Disables progress messages.

MESSAGE

The MESSAGE parameter displays or suppresses informational messages in the view source of your web browser. This parameter cannot be used with ON TABLE SET.

The syntax is:

```
SET {MESSAGE|MSG} = {ON|OFF}
```

where:

ON

Displays informational messages. ON is the default value.

OFF

Suppresses both informational messages and carets that appear when FOCUS executes commands in procedures. Error messages and the carets that prompt for input are still displayed.

MISS_ON

When a virtual field or calculated value can have missing values, you can specify whether all or some of the field values used in the expression that creates the DEFINE or COMPUTE field must be missing to make the result field missing. If you do not specify ALL or SOME for a DEFINE or COMPUTE with MISSING ON, the default value is SOME.

The SET parameter MISS_ON enables you to specify whether SOME or ALL should be used for MISSING ON in a DEFINE or COMPUTE that does not specify which to use.

The syntax is:

```
SET MISS_ON = {SOME|ALL}
```

where:

SOME
: Indicates that if at least one field in the expression has a value, the temporary field has a value (the missing values of the field are evaluated as 0 or blank in the calculation). If all of the fields in the expression are missing values, the temporary field has a missing value. SOME is the default value.

ALL
: Indicates that if all the fields in the expression have values, the temporary field has a value. If at least one field in the expression has a missing value, the temporary field has a missing value.

MISSINGTEST

By default, when an IF-THEN-ELSE expression is used to calculate a result and the IF expression evaluates to zero (for numeric expressions) or blank (for alphanumeric expressions), the left hand side is checked to see if it has MISSING ON. If it does, the result of the expression will be MISSING, not true or false, and the outcome returned will be MISSING, not the result of evaluating the THEN or ELSE expression, if the field only needs some missing values. You can use the SET MISSINGTEST command to eliminate the missing test for the IF expression so that either the THEN expression or the ELSE expression will be evaluated and returned as the result.

The syntax is:

```
SET MISSINGTEST = {NEW|OLD|SPECIAL}
```

where:

NEW
: Excludes the IF expression from the missing values evaluation so that it results in either true or false, not MISSING. If it evaluates to true, the THEN expression is used to calculate the result. If it evaluates to false, the ELSE expression is used to calculate the result. This is the default.

OLD
: Includes the IF expression in the missing values evaluation. If the IF expression evaluates to MISSING, the result is also MISSING, if the missing field only needs some missing values.

SPECIAL
: Is required for passing parameters to RStat.

MULTIPATH

The MULTIPATH parameter controls testing on independent paths.

The syntax is:

```
SET MULTIPATH = {SIMPLE|COMPOUND}
```

where:

SIMPLE

Includes a parent segment in the report output if:

- It has at least one child that passes its screening conditions.
- It lacks any referenced child on a path, but the child is optional (see the *Creating Reports* manual).

The (FOC144) warning message is generated when a request screens data in a multi-path report.

```
(FOC144) WARNING. TESTING IN INDEPENDENT SETS OF DATA:
```

COMPOUND

Includes a parent in the report output if it has *all* of its required children (see the *Creating Reports* manual). The COMPOUND setting does not generate the (FOC144) warning message. COMPOUND is the default value.

The segment rule is applied level by level as FOCUS descends the data source/view hierarchy. The existence of a parent segment depends on the existence of the child segment and the child segment depends on the existence of the grandchild for the full data source tree.

NODATA

The NODATA parameter determines the character string that indicates missing data in a report.

The syntax is:

```
SET {NODATA|NA} = {string|.}
```

where:

string

Is the character string that indicates missing data in reports. A period (.) is the default value.

NULL

The NULL parameter enables you to create a variable-length comma or tab delimited HOLD file that differentiates between a missing value and a blank string or zero value.

The HOLD formats supported for SET NULL=ON are COM, COMT, TAB, and TABT. Missing values in a record are denoted by two consecutive delimiters. A record that starts with a missing value has a delimiter in the first position, and a record that ends with a missing value has a delimiter in the last position.

The syntax is:

```
SET NULL = {ON|OFF}
```

where:

ON

Propagates missing values to a delimited HOLD file when the field has MISSING=ON in the Master File.

OFF

Propagates the value zero for a missing numeric value and blank ("") for a missing alphanumeric value to a delimited HOLD file. OFF is the default value.

OLDSTYRECLEN

The OLDSTYRECLEN parameter determines whether the record length, LRECL, is set to the current setting of LRECL=0, or the older setting of LRECL=512.

The syntax is:

```
SET OLDSTYRECLEN = {ON|OFF}
```

where:

ON

Determines that LRECL=512.

OFF

Determines that LRECL=0. OFF is the default value.

ONFIELD

The ONFIELD parameter determines whether ON phrases that refer to fields not present in the request are ignored or cause the request to terminate. Allowing ON phrases for absent fields enables user selections at run time to determine which elements are included in each execution of the request.

Note that any field used must be present in the Master File for the data source or the following message is generated and execution terminates:

The syntax is:

```
SET ONFIELD = {ALL|IGNORE}

ON TABLE SET ONFIELD {ALL|IGNORE}
```

where:

`ALL`

Issues a message and terminates execution when a field referenced in an ON phrase is not present in the request. ALL is the default value.

`IGNORE`

Ignores ON phrases that reference fields that are not present in the request as well as ON phrases that include options not supported by the type of field specified.

ORIENTATION

The ORIENTATION parameter specifies the page orientation for reports styled with StyleSheets.

The syntax is:

```
SET ORIENTATION = {PORTRAIT|LANDSCAPE}
```

where:

`PORTRAIT`

Displays the page in portrait style. PORTRAIT is the default value.

`LANDSCAPE`

Displays the page in landscape style.

OVERFLOWCHAR

The OVERFLOWCHAR parameter controls the characters displayed in a numeric report column when the column does not provide enough space to display its value. The number of overflow characters displayed is the same as the length assigned to the field. By default, the displayed overflow character is the asterisk (*).

The syntax is

```
SET OVERFLOWCHAR = 'char'
```

where:

char

Is a single byte displayable character. Depending on the character specified, it may not need to be enclosed in single quotation marks (').

The following characters are not supported as the overflow character: numeric digits, comma, period, apostrophe, percent sign, minus sign, space, current currency symbol, dollar sign, Yen symbol, Pound Sterling sign, and Euro symbol. In addition, other symbols may have significance in your operating environment.

PAGE[-NUM]

The PAGE[-NUM] parameter controls the numbering of output pages.

The syntax is:

```
SET PAGE[-NUM] = option
```

where:

option

Is one of the following:

ON displays the page number on the upper left-hand corner of the page. ON is the default value.

OFF suppresses page numbering.

NOPAGE suppresses page breaks, causing the report to be printed as a continuous page. When PAGE is set to NOPAGE, the LINES parameter controls where column headings are printed. You can use NOLEAD in place of NOPAGE.

TOP omits the line at the top of each page of the report output for the page number and the blank line that follows it. The first line of report output contains the heading, if one was specified, or the column titles if there is no heading.

Note: The settings ON, TOP, and OFF include the carriage control character 1 in the first column of each page.

PAGESIZE

The PAGESIZE parameter specifies the page size for StyleSheets. For optimal report appearance, the actual paper size must match your setting for PAGESIZE. If it does not, your report is cropped or contains extra blank spaces.

The syntax is:

```
SET PAGESIZE = size
```

where:

size

Specifies the page size. If the actual paper size does not match the PAGESIZE setting, your report is either cropped or contains extra blank space.

The page size options are:

LETTER sets the page size to 8.5 x 11 inches.
ENVELOPE-PERSONAL sets the page size to 3.625 x 6.5 inches.
ENVELOPE-MONARCH sets the page size to 3.875 x 7.5 inches.
ENVELOPE-9 sets the page size to 3.875 x 8.875 inches.
ENVELOPE-10 sets the page size to 4.125 x 9.5 inches.
ENVELOPE-12 sets the page size to 4.5 x 11 inches.
ENVELOPE-DL sets the page size to 4.3 x 8.6 inches.
ENVELOPE-ITALY sets the page size to 4.3 x 9.1 inches.
ENVELOPE-B4 sets the page size to 9.8 x 13.9 inches.
ENVELOPE-B5 sets the page size to 6.9 x 9.8 inches.
ENVELOPE-B6 sets the page size to 6.9 x 4.9 inches.
ENVELOPE-C3 sets the page size to 12.75 x 18 inches.
ENVELOPE-C4 sets the page size to 9 x 12.75 inches.
ENVELOPE-C5 sets the page size to 6.4 x 9 inches.
ENVELOPE-C6 sets the page size to 4.5 x 6.375 inches.
ENVELOPE-C65 sets the page size to 4.5 x 9 inches.

STATEMENT sets the page size to 5.5 x 8.5 inches.
EXECUTIVE sets the page size to 7.5 x 10.5 inches.
GERMAN-STANDARD-FANFOLD sets the page size to 8.5 x 12 inches.
GERMAN-LEGAL-FANFOLD sets the page size to 8.5 x 13 inches.
FOLIO sets the page size to 8.5 x 13 inches.
LEGAL sets the page size to 8.5 x 14 inches.
10X14 sets the page size to 10 x 14 inches.
TABLOID sets the page size to 11 x 17 inches.
CUSTOM enables you to set a custom page size for a DHTML, PDF, or PPTX report.

A3 sets the page size to 11.7 x 16.8 inches.
A4 sets the page size to 8.25 x 11.7 inches.
A5 sets the page size to 5.8 x 8.25 inches.
B4 sets the page size to 9.8 x 13.9 inches.
B5 sets the page size to 7.2 x 10.1 inches.
C sets the page size to 17 x 22 inches.
D sets the page size to 22 x 34 inches.
E sets the page size to 34 x 44 inches.
US-STANDARD-FANFOLD sets the page size to 14.875 x 11 inches.
LEDGER sets the page size to 17 x 11 inches.
QUARTO sets the page size to 8.5 x 10.8 inches.

PANEL

The PANEL parameter sets the maximum line width, in characters, of a report panel for a screen or printer. If report output exceeds this value, the output is partitioned into several panels. For example, if you set PANEL to 80, the first 80 characters of a record appear on the first panel, the second 80 characters appear on the second panel, and so on.

When printing a report to your screen, the ideal value for the PANEL parameter is the width of your screen (usually 80). When printing to your printer, the ideal value for PANEL is the print width of your printer (usually 132). If PANEL is larger or set to 0, long report lines wrap around the screen or page.

When the BYPANEL parameter is OFF, a report can be divided into a maximum of 4 panels. If SET BYPANEL has a value other than OFF, the report may be divided into 99 panels.

When the STYLESHEET parameter is in effect, PANEL is ignored.

The syntax is:

```
SET PANEL = {0|n}
```

where:

n

Is the maximum line width, in characters, of a report panel.

0

Does not divide the report into panels. Long report lines wrap around the screen or page. 0 is the default value.

PARTITION_ON

When using a statistical function, you must establish the size of the partition on which the function will operate, if the request contains sort fields. You can do this using the PARTITION_ON command.

The syntax is:

```
SET PARTITION_ON = {FIRST|PENULTIMATE|TABLE}
```

where:

`FIRST`
Uses the first (also called the major) sort field in the request to partition the values.

`PENULTIMATE`
Uses the next to last sort field where the COMPUTE is evaluated to partition the values. This is the default value.

`TABLE`
Uses the entire internal matrix to calculate the statistical function.

PASS

The PASS parameter enables user access to a data source or stored procedure protected by Information Builders security.

This command cannot be used with ON TABLE SET.

The syntax is:

```
SET PASS = password [IN filename]
```

where:

password

Is the password that allows access to data sources protected by Information Builders database security.

filename

Is a specific FOCUS data source or stored procedure protected by security.

PCOMMA

The PCOMMA parameter controls the retrieval of comma-delimited files.

By default, when a Master File specifies SUFFIX=COM, incoming alphanumeric values are not enclosed in double quotation marks, and each record is terminated with a comma and dollar sign (,$) character combination. This format does not support retrieval of most comma-delimited files produced by a PC application.

The syntax is:

```
SET PCOMMA = option
```

where:

option
Can be one of the following:

- **ON**, which enables the retrieval of comma-delimited data sources created by a PC application, in which alphanumeric data is enclosed in double quotation marks and each record is completely contained on one line and is terminated with a carriage return and line feed. It can also retrieve comma-delimited data sources in which alphanumeric data is not enclosed in double quotation marks and each record is terminated with a comma and dollar sign.
- **OFF**, which does not enable the retrieval of comma-delimited data sources created by a PC application. It indicates that alphanumeric data is not enclosed in double quotation marks and each record is terminated with a comma and dollar sign. OFF is the default value.
- **DFIX**, which causes delimited files with SUFFIX=COM, COMT, TAB, and TABT to be processed through the Adapter for DFIX. This processing provides more complete and meaningful messages and some changes to the processing of missing values when two delimiters in a row are encountered. With DFIX processing, a missing value is assigned to the field.

  In order to be eligible for DFIX processing, the delimited file must satisfy the following requirements.

  - Each record must be completely contained on one line and terminated with the crlf (carriage return/linefeed) character combination.
  - The ENCLOSURE can be only in the first position after the delimiter for COM (new) and COMT records. Otherwise, it will not be recognized.
  - The number of fields on a line cannot exceed the number of fields defined in the Master File.

PCTFORMAT

The PCTFORMAT parameter controls whether fields prefixed with the operators PCT., RPCT., and PCT.CNT. display with a percent sign or with the format associated with the original field.

The syntax is:

```
SET PCTFORMAT = {OLD|PERCENT}
```

where:

OLD
: Displays columns prefixed with PCT., RPCT., and PCT.CNT. with the format associated with the original field.

PERCENT
: Displays columns prefixed with PCT., RPCT., and PCT.CNT. with a percent sign. It also allows the prefixed fields to be reformatted. This is the default value.

PCT.CNT.*field* will always display with two decimal places, unless reformatted. For PCT.*field* and RPCT.*field*, with SET PCTFORMAT = PERCENT, if the original field has a:

- Precision-based format (F, D, M, X), the column will display with length 7 and two decimal places.
- Packed format, the column will display with its original number of decimal places.
- Integer format, the column will display with no decimal places.

PDFLINETERM

The PDFLINETERM parameter determines if an extra space is appended to each record of a PDF output file to facilitate proper file transfer between Windows and UNIX.

In Windows systems, the end of each PDF file has a table containing the byte offset, including two line termination characters, a carriage return, and a line feed. In UNIX, files are terminated by only one character, a line feed. Transferring files between Windows and UNIX systems requires the proper use of the PDFLINETERM parameter.

The syntax is:

```
SET PDFLINETERM = {STANDARD|SPACE}
```

where:

STANDARD

Creates a PDF file without any extra characters. This file will be a valid PDF file if transferred in text mode to a Windows machine, but not to a UNIX machine. If subsequently transferred from a UNIX machine to a Windows machine in text mode, it will be a valid PDF file on the Windows machine.

SPACE

Creates a PDF file with an extra space character appended to each record. This file will be a valid PDF file if transferred in text mode to a UNIX machine, but not to a Windows machine. If subsequently transferred from an ASCII UNIX machine to a Windows machine in binary mode, it will be a valid PDF file on the Windows machine.

PERMPASS

The PERMPASS parameter establishes a user password that remains in effect throughout a session or connection. You can issue this setting in any supported profile but is most useful when established for an individual user by setting it in a user profile. It cannot be set in an ON TABLE phrase. It is recommended that it not be set in FOCPARM or FOCPROF because it would then apply to all users. In a FOCUS session, SET PERMPASS can be issued in PROFILE, a FOCEXEC, or at the command prompt.

All security rules established in the DBA sections of existing Master Files are respected when PERMPASS is in effect. The user cannot issue the SET PASS or SET USER command to change to a user password with different security rules. Any attempt to do so generates the following message:

```
permanent PASS is in effect. Your PASS will not be honored.
VALUE WAS NOT CHANGED
```

Only one permanent password can be established in a session. After it is set, it cannot be changed within the session.

The syntax is:

```
SET PERMPASS=userpass
```

where:

userpass

Is the user password used for all access to data sources with DBA security rules established in their associated Master Files.

PHONETIC_ALGORITHM

The PHONETIC_ALGORITHM parameter sets a phonetic algorithm to use with the PHONETIC function, which calculates an index for alphanumeric values such as names, based on their pronunciation, so that words that have variations in spelling can be grouped together.

The syntax is:

```
SET PHONETIC_ALGORITHM = {METAPHONE|SOUNDEX}
```

where:

METAPHONE

Uses the Metaphone algorithm for indexing. Metaphone is suitable for use with most English words, not just names. Metaphone algorithms are the basis for many popular spell checkers. METAPHONE is the default algorithm, except on z/OS.

Note: Metaphone is not optimized in the SQL sent to a relational DBMS. Therefore, if you need to optimize the request for an SQL DBMS, the SOUNDEX value should be used.

SOUNDEX

Soundex is a legacy phonetic algorithm for indexing names by sound, as pronounced in English. SOUNDEX is the default algorithm on z/OS.

PRFTITLE

The PRFTITLE parameter generates descriptive column titles for prefixed fields. These column titles have readable and translatable descriptions of the prefix operators.

The syntax is:

```
SET PRFTITLE = {SHORT|LONG}
```

where:

SHORT
: Places the prefix operator name above the field name to generate the column title.

LONG
: Generates descriptive column titles for prefixed fields that can be translated to other languages.

PRINT

The PRINT parameter specifies the report output destination.

It determines whether report output is sent to your screen or to the printer.

You can enter ONLINE and OFFLINE as separate commands that have the same effect as specifying ONLINE and OFFLINE as PRINT settings.

The syntax is:

```
SET PRINT = {ONLINE|OFFLINE}
```

where:

ONLINE

Sends report output to the terminal. ONLINE is the default value.

OFFLINE

Sends report output to the system printer.

PRINTDST

The handling of DST operators has been improved to support multiple DST operators in the same request, and the ability to use DST with ACROSS.

With these improvements, you can control the behavior of requests that use the PRINT command with multiple DST operators to achieve independent DST values. To implement this functionality, set the PRINTDST parameter to NEW.

The syntax is:

```
SET PRINTDST = {OLD|NEW}
```

where:

OLD

Processes multiple DST operators in a PRINT request as nested BY fields, making them dependent on each other. OLD is the default value.

NEW

Processes multiple DST operators in a PRINT request as totally independent objects.

PRINTPLUS

The PRINTPLUS parameter introduces enhancements to the display alternatives offered by the FOCUS Report Writer. To force a break at a specific spot, you must use NOSPLIT. PRINTPLUS is not supported with StyleSheets. Problems may be encountered if HOTSCREEN is set to OFFLINE.

The syntax is:

```
SET {PRINTPLUS|PRTPLUS} = {ON|OFF}
```

where:

ON

Handles the PAGE-BREAK internally to provide the correct spacing of pages, NOSPLIT is handled internally and you can perform RECAPs in cases where pre-specified conditions are met. Additionally, a Report SUBFOOT now prints above the footing instead of below it.

OFF

Does not support StyleSheets. OFF is the default value.

PSPAGESETUP

The PSPAGESETUP parameter causes the paper source used by a PostScript printer to match the PAGESIZE parameter setting.

The syntax is:

```
SET PSPAGESETUP = {OFF|ON}
```

where:

OFF

Does not include PostScript code for the selection of a PostScript printer paper source. OFF is the default value.

ON

Includes PostScript code that automatically tells a PostScript printer to set its paper source to the size specified by PAGESIZE.

QUALCHAR

The QUALCHAR parameter specifies the qualifying character to be used in qualified field names.

The syntax is:

```
SET QUALCHAR = {character|.}
```

where:

character

Is a valid qualifying character. They include:

.	period	(hex 4B)
:	colon	(hex 7A)

!	exclamation point	(hex 5A)
%	percent sign	(hex 6C)
¦	broken vertical bar	(hex 6A)
\	backslash	(hex E0)

A period (.) is the default value. The use of the other qualifying characters listed above is restricted and should not be used with 66-character field names.

If the qualifying character is a period, you can use any of the other characters listed above as part of a field name. If you change the default qualifying character to a character other than the period, then you cannot use that character in a field name.

QUALTITLES

The QUALTITLES parameter uses qualified column titles in report output when duplicate field names exist in a Master File. A qualified column title distinguishes between identical field names by including the segment name.

The syntax is:

```
SET QUALTITLES = {ON|OFF}
```

where:

ON

Uses qualified column titles when duplicate field names exist and FIELDNAME is set to NEW.

OFF

Disables qualified column titles. OFF is the default value.

RANK

The RANK parameter determines how rank numbers are assigned when a request contains the [RANKED] BY [HIGHEST|LOWEST] *n* phrase and multiple data values fall into the same rank category. If the rank number for the next group of values is the next sequential integer, the ranking method is called *dense*. If the rank number for the next group of values is the previous rank number plus the number of multiples, the ranking method is called *sparse*.

The syntax is:

```
SET RANK = {DENSE|SPARSE}
```

where:

DENSE

Specifies dense ranking. With this method, each rank number is the next sequential integer, even when the same rank is assigned to multiple data values. DENSE is the default value.

SPARSE

Specifies sparse ranking. With this method, if the same rank number is assigned to multiple data values, the next rank number will be the previous rank number plus the number of multiples.

RECAP-COUNT

The RECAP-COUNT parameter includes lines containing a value created with RECAP when counting the number of lines per logical page for printed output.

The number of lines per page is determined by the LINES parameter.

The syntax is:

```
SET RECAP-COUNT = {ON|OFF}
```

where:

ON

Counts lines containing a value created with RECAP.

OFF

Does not count lines containing a value created with RECAP. OFF is the default value.

RECORDLIMIT

The RECORDLIMIT parameter limits the number of records retrieved or displayed.

The syntax is:

```
SET RECORDLIMIT = {RECORDLIMIT|OUTPUTLIMIT}
```

where:

RECORDLIMIT

In a request with a RECORDLIMIT filter (WHERE RECORDLIMIT EQ *n* or IF RECORDLIMIT EQ *n*), limits the records displayed on the report output to the number of reads specified in the filter. RECORDLIMIT is the default.

OUTPUTLIMIT
In a request with a RECORDLIMIT filter (WHERE RECORDLIMIT EQ *n* or IF RECORDLIMIT EQ *n*), applies the RECORDLIMIT filter to the number of records displayed in the final output.

RIGHTMARGIN

The RIGHTMARGIN parameter sets the StyleSheet right boundary for report contents on a page. This parameter applies to PostScript and PDF reports.

The syntax is:

```
SET RIGHTMARGIN = {value|.250}
```

where:

value

Is the right boundary of report contents on a page. 0.250 inches is the default value.

RPAGESET

The RPAGESET parameter controls how the number of lines per logical page are determined when output contains text created with SUBFOOT and a field value created with RECAP.

The syntax is:

```
SET RPAGESET = {NEW|OLD}
```

where:

NEW

Sets the number of lines per logical page equal to the LINES value plus two plus the number of the highest BY field with a SUBFOOT.

OLD

Sets the number of lines per logical page equal to the value of the LINES parameter. See *LINES* on page 113 for details. OLD is the default.

SAVEDMASTERS

The SAVEDMASTERS parameter saves a Master File in memory after it is used in a request. Saving a Master File prevents re-parsing the Master File when referenced in subsequent requests, resulting in performance improvement.

Up to 99 Master Files can be saved to memory.

This parameter cannot be set in the ON TABLE SET command.

The syntax is:

```
SET SAVEDMASTERS = n
```

where:

n

Is an integer between 0 and 99 that specifies the maximum number of Master Files on the SAVEDMASTERS list. 10 is the default value.

Note that the most recently used Master File is always stored in memory, even with SAVEDMASTERS set to zero. However, the zero setting does not generate the list of saved Master Files.

SAVEMATRIX

The SAVEMATRIX parameter saves the matrix from your request to protect it from being overwritten when using Dialogue Manager commands.

The syntax is:

```
SET SAVEMATRIX = {ON|OFF}
```

where:

ON

Saves the internal matrix from the last report request, preventing it from being overwritten.

OFF

Overwrites the internal matrix for each request. OFF is the default.

SHADOW

The SHADOW parameter activates the Absolute File Integrity feature for FOCUS files (but not XFOCUS files).

The syntax is:

```
SET SHADOW [PAGE] = {ON|OFF|OLD}
```

where:

ON

Activates the FOCUS Absolute File Integrity feature. The maximum number of pages shadowed is 256K.

OFF

Deactivates the Absolute File Integrity feature. OFF is the default value.

OLD

Indicates that your FOCUS file was created before Version 7.0. This means that the maximum number of pages shadowed is 63,551.

SHIFT

The SHIFT parameter controls the use of *shift* strings.

The syntax is:

```
SET SHIFT = {ON|OFF}
```

where:

ON

Specifies a shift string for Hebrew or DBCS (double-byte character support).

OFF

Indicates that SHIFT is not in effect. OFF is the default value.

SHORTPATH

The SHORTPATH parameter controls how screening conditions against missing cross-referenced segment instances are processed in a left outer join.

In FOCUS, the command SET ALL = ON or JOIN LEFT_OUTER specifies a left outer join. With a left outer join, all records from the host file display on the report output. If a cross-referenced segment instance does not exist for a host segment instance (called a *short path*), the report output displays missing values for the fields from the cross-referenced segment. However, the fields are not assigned missing values for testing purposes.

If there is a screening condition on the dependent segment, those dependent segment instances that do not satisfy the screening condition are omitted from the report output, and so are their corresponding host segment instances. With missing segment instances, tests for missing values fail because the fields in the segment have not been assigned missing values.

When a relational engine performs a left outer join, it processes host records with missing cross-referenced segment instances slightly differently from the way FOCUS processes those records when both of the following conditions apply:

- There is a screening condition on the cross-referenced segment.
- A host segment instance does not have a corresponding cross-referenced segment instance.

When these two conditions are true, FOCUS omits the host record from the report output, while relational engines supply null values for the fields from the dependent segment and then apply the screening condition. If the missing values pass the screening condition, the entire record is retained on the report output. This type of processing is useful for finding or counting all host records that do not have matching records in the cross-referenced file or for creating a DEFINE-based join from the cross-referenced segment with the missing instance to another dependent segment.

If you want FOCUS to assign null values to the fields in a missing segment instance when a left outer join is in effect, you can issue the command SET SHORTPATH=SQL.

```
SET SHORTPATH = {FOCUS|SQL}
```

where:

FOCUS

Omits a host segment from the report output when it has no corresponding cross-referenced segment and the report has a screening condition on the cross-referenced segment.

SQL

Supplies missing values for the fields in a missing cross-referenced segment in an outer join. Applies screening conditions against this record and retains the record on the report output if it passes the screening test.

Note: There must be an outer join in effect, either as a result of the SET ALL=ON command or a JOIN LEFT_OUTER command (either inside or outside of the Master File).

SHOWBLANKS

The SHOWBLANKS parameter preserves leading and internal blanks in HTML and EXL2K report output.

The syntax is:

```
SET SHOWBLANKS = {OFF|ON}
```

where:

OFF

Removes leading and internal blanks in HTML and EXL2K report output. OFF is the default value.

ON

Preserves leading and internal blanks in HTML and EXL2K report output.

SORTMATRIX

The SORTMATRIX parameter controls whether to employ in-memory sorting with decreased use of external memory. The syntax is

```
SET SORTMATRIX = {SMALL|LARGE}
```

where:

SMALL
: Creates a single sort matrix of up to 2048 rows, and uses a binary search based insertion sort with aggregation during retrieval. The maximum number of rows in this matrix has been determined to provide the best performance for this type of sort. If the sort matrix becomes full, it is written to a file called FOCSORT on disk, the in-memory matrix is emptied, and retrieval continues, writing to FOCSORT as many times as necessary. When the end of data is detected, the remaining rows are written to FOCSORT and the merge routine merges all of the sort strings in FOCSORT (which, in extreme cases, may require multiple merge phases), while also completing the aggregation.

LARGE
: Creates a large matrix or multiple small matrices in memory, when adequate memory is available as determined by the SORTMEMORY parameter. LARGE is the default value. The goal of this strategy is to do as much sorting as possible in internal memory before writing any records to disk. Whether disk I/O is necessary at all in the sorting process depends on the amount of memory allocated for sorting and the size of the request output. If the amount of SORTMEMORY is not large enough to meaningfully make use of the LARGE strategy, the sort will default to the SMALL strategy. The LARGE strategy greatly reduces the need for disk I/O and, if disk I/O is required after all (for very large output), it virtually eliminates the need for multiple merge phases.

SORTMEMORY

The SORTMEMORY parameter controls the amount of internal memory available for sorting. The syntax is:

```
SET SORTMEMORY = {n|512}
```

where:

n
: Is the positive number of megabytes of memory available for sorting. The default value is 512.

SPACES

The SPACES parameter sets the number of spaces between columns in a report.

This parameter does not work with HTML, PDF, or styled reports.

The syntax is:

```
SET SPACES = {AUTO|n}
```

where:

AUTO

Automatically places either one to two spaces between columns. AUTO is the default value.

n

Is the number of spaces to place between columns of a report. Valid values are integers between one and eight.

SQLTOPTTF

The SQLTOPTTF parameter enables the SQL Translator to generate TABLEF commands instead of TABLE commands.

The syntax is:

```
SET SQLTOPTTF = {ON|OFF}
```

where:

ON

Generates TABLEF commands when possible. For example, a TABLEREF command is generated if there is no JOIN or GROUP BY command. ON is the default value. ON is the default value.

OFF

Always generates TABLE commands.

SQUEEZE

The SQUEEZE parameter applies only to the StyleSheet feature.

It determines the column width in report output. The column width is based on the size of the data value or column title, or on the field format defined in the Master File.

The syntax is:

```
SET SQUEEZE = {ON|OFF|n}
```

where:

ON

Assigns column widths based on the widest data value or widest column title, whichever is longer.

OFF

Assigns column widths based on the field format specified in the Master File. This value pads the column width to the length of the column title or field format descriptions, whichever is greater. OFF is the default value.

n

Represents a specific numeric value, based on the UNITS parameter setting, to which the column width can be set (valid only in PDF and PS).

STYLE[SHEET]

The STYLE[SHEET] parameter controls the format of report output by accepting or rejecting StyleSheet parameters. The parameters specify formatting options, such as page size, orientation, and margins.

The syntax is:

```
SET STYLE[SHEET] = {stylesheet|ON|OFF}
```

where:

stylesheet

Is the name of the StyleSheet file. For UNIX and Windows, this is the name of the StyleSheet file without the file extension .sty. For z/OS, this is the member name in the PDS allocated to ddname FOCSTYLE.

For a PDF or PostScript report, it uses the page layout settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE. The settings for LINES, PAPER, PANEL, and WIDTH are ignored.

ON

Uses the page layout settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE. The settings for LINES, PAPER, PANEL, and WIDTH are ignored.

For a PDF or PostScript report, uses the page layout settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE; the settings for LINES and WIDTH are ignored.

OFF

This uses the settings for LINES, PAPER, PANEL, and WIDTH. The settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE are ignored. OFF is the default value.

SUBTOTALS

The SUBTOTALS parameter specifies whether summary lines are displayed above or below the detail lines in a report. The summary commands affected include SUBTOTAL, SUB-TOTAL, RECOMPUTE, SUMMARIZE, COMPUTE, RECAP, and COLUMN-TOTAL.

The syntax is:

```
SET SUBTOTALS {ABOVE|BELOW}
```

where:

ABOVE

Places summary lines above the detail lines and displays the sort field values on every detail line of the report output.

BELOW

Places summary lines below the detail lines. BELOW is the default value.

SUMMARYLINES

The SUMMARYLINES parameter allows users to combine fields with and without prefix operators on summary lines in one request. Prefix operator processing is used for all summary lines. Fields without prefix operators are processed as though they were specified with the operator SUM.

This command cannot be used with ON TABLE SET.

The syntax is:

```
SET SUMMARYLINES = {NEW|OLD|EXPLICIT}
```

where:

NEW

Propagates all summary operations to the grand total line. Uses prefix operator processing for all summary commands (all summary fields without prefix operators are processed as though they had a SUM. operator). Fields listed in a summary command are populated only on summary lines created by that summary command and on summary lines created by propagation of that summary command. Supports display of alphanumeric fields on summary lines. NEW is the default value.

OLD

This value is no longer supported. It processes as NEW.

EXPLICIT

Does not propagate SUBTOTAL and RECOMPUTE to the grand total line. Uses prefix operator processing for all summary commands (all summary fields without prefix operators are processed as though they had a SUM. operator). Fields listed in a summary command are populated only on summary lines created by that summary command and on summary lines created by propagation of that summary command. Supports display of alphanumeric fields on summary lines.

Note: This command is not supported in a request using the ON TABLE SET syntax.

SUMPREFIX

The SUMPREFIX parameter allows users to choose the answer set display order when using an external sort to perform aggregation on alphanumeric or smart date formats.

The syntax is:

```
SET SUMPREFIX = {FST|LST|MIN|MAX}
```

where:

FST

Displays the first value when alphanumeric or smart date data types are aggregated.

LST

Displays the last value when alphanumeric or smart date data types are aggregated. LST is the default value.

MIN

Displays the minimum value in the sort order set by your FOCUS code page and configuration when alphanumeric or smart date data types are aggregated.

MAX

Displays the maximum value in the sort order set by your FOCUS code page and configuration when alphanumeric or smart date data types are aggregated.

TESTDATE

The TESTDATE parameter temporarily alters the system date in order to test a dynamic window allowing you to simulate clock settings to determine the behavior of your program.

The syntax is:

```
SET TESTDATE = {yyyymmdd|TODAY}
```

where:

yyyymmdd

Is an 8-digit date in the format YYYYMMDD.

TODAY

Is the current date. TODAY is the default value.

TIME_SEPARATOR

This parameter defines the separator for time components for the&TOD system variable.

The syntax is:

```
SET TIME_SEPARATOR = {DOT|COLON}
```

where:

DOT

Uses a dot (.) to separate time components. This is the default value.

COLON

Uses a colon (:) to separate time components.

TITLELINE

The TITLELINE parameter controls underlining of column titles on report output.

The syntax is:

```
SET {TITLELINE|ACROSSLINE} = {ON|OFF|SKIP}
```

where:

ON

Underlines column titles on report output. ON is the default value.

OFF

Replaces the underline with a blank line.

SKIP

Specifies no underline and no blank line.

TITLES

The TITLES parameter controls whether to use pre-defined column titles in the Master File as column titles in report output.

The syntax is:

```
SET TITLES = {ON|OFF|NOPREFIX}

ON TABLE SET TITLES {ON|OFF|NOPREFIX}
```

where:

ON

Displays the value of the TITLE attribute as the column heading on the report output, if a TITLE attribute exists in the Master File. If the field has a prefix operator in the report request, creates the column heading using both the prefix operator and the TITLE attribute. If there is no TITLE attribute, the field name is used instead. ON is the default value.

OFF

Displays the field name as the column heading on the report output. If the field has a prefix operator in the report request, creates the column heading using both the prefix operator and the field name.

NOPREFIX

Displays the value of the TITLE attribute as the column heading on the report output, if a TITLE attribute exists in the Master File. If there is no TITLE attribute, the field name is used instead. If the field has a prefix operator in the report request, creates the column heading using both the prefix operator and the field name.

TOPMARGIN

The TOPMARGIN parameter sets the top StyleSheet boundary for report contents on a page.

This parameter applies to PostScript and PDF reports.

The syntax is:

```
SET TOPMARGIN = {value|.250}
```

where:

value

Is the top boundary on a page for report output. 0.250 inches is the default value.

UNITS

The UNITS parameter applies to PostScript and PDF reports.

It specifies the unit of measure for page margins, column positions, and column widths.

The syntax is:

```
SET UNITS = {INCHES|CM|PTS}
```

where:

INCHES

Uses inches as the unit of measure. INCHES is the default value.

CM

Uses centimeters as the unit of measure.

PTS

Uses points as the unit of measurement. (One inch = 72 points, one cm = 28.35 points).

USER

The USER parameter enables user access to a data source or stored procedure protected by Information Builders security.

The syntax is:

```
SET USER = user
```

where:

user

Is the user name that, with a password, enables access to a data source or stored procedure protected by Information Builders security.

USERFCHK

The USERFCHK parameter controls the level of verification applied to DEFINE FUNCTION arguments and Information Builders-supplied function arguments. It does not affect verification of the number of parameters. The correct number must always be supplied.

Note that the USERFNS=SYSTEM setting must be in effect. For details, see *USERFNS* on page 146.

Issue the following command in FOCPARM, FOCPROF, on the command line, in a procedure, or in an ON TABLE command.

```
SET USERFCHK = setting
```

where:

setting

Can be one of the following:

ON verifies parameters in requests, but does not verify parameters for functions used in Master File DEFINEs. If a parameter has an incorrect length, an attempt is made to fix the problem. If such a problem cannot be fixed, a message is generated and the evaluation of the affected expression is terminated. ON is the default value.

Because parameters are not verified for functions specified in a Master File, no errors are reported for those functions until the DEFINE field is used in a subsequent request when, if a problem occurs, the following message is generated:

```
(FOC003) THE FIELDNAME IS NOT RECOGNIZED
```

OFF does not verify parameters except in the following cases:

- If a parameter that is too long would overwrite the memory area in which the computational code is stored, the size is automatically reduced without issuing a message.

  Note: The OFF setting will be deprecated in a future release.

- If an alphanumeric parameter is too short, it is padded with blanks to the correct length.

Note: We strongly recommend that you not use this option, as disabling parameter checking can lead to unexpected issues.

FULL is the same as ON, but also verifies parameters for functions used in Master File DEFINEs.

ALERT verifies parameters in a request without halting execution when a problem is detected. It does not verify parameters for functions used in Master File DEFINEs. If a parameter has an incorrect length and an attempt is made to fix the problem behind the scenes, the problem is corrected with no message. If such a problem cannot be fixed, a warning message is generated. Execution then continues as though the setting were OFF.

USERFNS

If your site has a locally written function with the same name as an Information Builders-supplied function, the USERFNS parameter determines which function is used.

Parameter verification can be enabled for DEFINE FUNCTIONs and functions supplied by Information Builders.

The syntax is:

```
SET USERFNS= {SYSTEM|LOCAL}
```

where:

SYSTEM

Gives precedence to functions supplied by Information Builders and to those created with the DEFINE FUNCTION command. SYSTEM is the default value.

This setting is required to enable parameter verification. For details, see *USERFCHK* on page 145.

LOCAL

Gives precedence to locally written functions. Parameter verification is not performed with this setting in effect.

WARNING

The WARNING parameter suppresses (FOC441) warnings. The file exists already. Create will overwrite it.

The syntax is:

```
SET WARNING = {ON|OFF}
```

where:

ON

Turns on warning messages. On is the default.

OFF

Turns off warning messages.

WEEKFIRST

The WEEKFIRST parameter specifies a day of the week as the start of the week. This is used in week computations by the HDIFF, HNAME, HPART, HYYWD, and HSETPT functions, described in the *Using Functions* manual.

The HPART and HNAME subroutines can extract a week number from a date-time value. To determine a week number, they can use ISO 8601 standard week numbering, which defines the first week of the year as the first week in January with four or more days. Any preceding days in January belong to week 52 or 53 of the preceding year.

Depending on the value of WEEKFIRST, these functions can also define the first week of the year as the first week in January with seven days.

The WEEKFIRST parameter does not change the day of the month that corresponds to each day of the week, but only specifies which day is considered the start of the week.

The syntax is:

```
SET WEEKFIRST = {value|7}
```

where:

value
Can be:

1 through 7, representing Sunday through Saturday with non-standard week numbering.

or

ISO1 through ISO7, representing Sunday through Saturday with ISO standard week numbering.

Note: ISO is a synonym for ISO2.

The ISO standard establishes Monday as the first day of the week, so to be fully ISO compliant, the WEEKFIRST parameter should be set to ISO or ISO2.

WPMINWIDTH

If you need the report width for a format WP output file to remain fixed across releases for later processing of the output file, you can set the width you need using the SET WPMINWIDTH command. This parameter specifies the minimum width of the output file. It will be automatically increased if the width you set cannot accommodate the fields propagated to the output file in the request. On z/OS, The LRECL of the output file will be four bytes more than the report width because the file is variable length and needs an additional four bytes to hold the actual length of each record instance. In other operating environments, the length of the record is the value of WPMIDWIDTH.

The syntax is:

```
SET WPMINWIDTH = {0|nnn}

ON TABLE SET WPMINWIDTH {0|nnn}
```

where:

nnn

Is the minimum width of the output file. On z/OS, the LRECL will automatically be *nnn* + 4 bytes. If you specify zero (0) for *nnn*, the width will be calculated automatically based on the report request. If the width you specify cannot accommodate the fields propagated to the output file, it will be automatically increased enough to accommodate them.

XLSXPAGEBRKIGNORE

The XLSXPAGEBRKIGNORE parameter controls whether page breaks in FOCUS format XLSX report output insert Excel page breaks at the same location in the output.

The syntax is:

```
SET XLSXPAGEBRKIGNORE = {OFF|ON}
```

where:

OFF

Synchronizes FOCUS page breaks with Excel page breaks in format XLSX report output. This is the default value.

ON

Does not synchronize FOCUS page breaks with Excel page breaks in format XLSX report output. This value conforms to behavior in prior releases.

XRETRIEVAL

The XRETRIEVAL parameter previews the format of a report without actually accessing any data. This parameter enables you to perform TABLE, TABLEF, or MATCH requests and produce HOLD Master Files without processing the report.

The syntax is:

```
SET XRETRIEVAL = {ON|OFF}
```

where:

ON

Performs retrieval when previewing a report. ON is the default value.

OFF

Specifies that no retrieval is to be performed.

YRTHRESH

The YRTHRESH parameter defines the start of a 100-year window globally or on a field-level. Used with DEFCENT, interprets the current century according to the given values. Two-digit years greater than or equal to YRTHRESH assume the value of the default century. Two-digit years less than YRTHRESH assume the value of one more than the default century. (See *DEFCENT* on page 83.)

Note: This same result can be achieved by including the FDEFCENT and FYRTHRESH attributes in the Master File.

The syntax is:

```
SET YRTHRESH = {[-]yy|0}
```

where:

yy

Is the year threshold for the window. 0 is the default value.

If *yy* is a positive number, that number is the start of the 100-year window. Any 2-digit years greater than or equal to the threshold assume the value of the default century. Two-digit years less than the threshold assume the value of one more than the default century.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and the default century is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

Chapter 2

Managing Applications

An application is a platform-independent repository for a group of related components, such as procedures, Master and Access Files, data files, HTML files, PDF files, and image files.

You can use a variety of application (APP) commands to control the application environment, including the application itself, its component files, and its search paths.

In this chapter:

- What Is an Application?
- Application Commands Overview
- Search Path Management Commands
- Application and File Management Commands
- Output Redirection Commands
- Application Metadata Commands and Catalog Metadata
- APP HELP
- Accessing Metadata and Procedures
- Allocating Temporary Files

What Is an Application?

An application is a platform-independent repository for a group of related components, such as procedures, Master and Access Files, data files, HTML files, PDF files, and image files. It provides a way to confer a unique identity on the application components and facilitates the sharing of components across applications in an organized manner. This construct also simplifies the process of moving a user application from one platform to another.

These components are physically grouped together on an application-by-application basis for run-time execution. This physical grouping can be within an application under a common root or a mapping to an application anywhere in the file system. The physical application or mapped name is referred to as the application name in this document. A comprehensive set of application (APP) commands are provided to control/manipulate the application components, as well as to facilitate applications that can be written and deployed to any platform.

The physical location of an application and its components is determined by a configuration parameter called approot. This parameter is set at installation time and stored in the configuration file, edaserve.cfg. On z/OS, the EDASERVE configuration file must be a member in a PDS allocated to DDNAME ERRORS. The default value is dependent on the platform, relative to the install ID home directory, where applicable.

Application directories can be nested, except on z/OS. A nested application directory is an application created within a higher-level application. For more information. see *Nested Application Directories*.

On z/OS, data sets are created for each component type using the approot value as the high-level qualifier.

On z/OS, the following is a list of all application data sets automatically created for an application, where approot is USERID.APPS and the application name is BASEAPP:

```
APP CREATE BASEAPP
USERID.APPS.BASEAPP.ACCESS.DATA
USERID.APPS.BASEAPP.ETG.DATA
USERID.APPS.BASEAPP.FOCCOMP.DATA
USERID.APPS.BASEAPP.FOCEXEC.DATA
USERID.APPS.BASEAPP.FOCSTYLE.DATA
USERID.APPS.BASEAPP.GIF.DATA
USERID.APPS.BASEAPP.HTML.DATA
USERID.APPS.BASEAPP.MAINTAIN.DATA
USERID.APPS.BASEAPP.MASTER.DATA
USERID.APPS.BASEAPP.SQL.DATA
USERID.APPS.BASEAPP.WINFORMS.DATA
```

To reference an application file in a FOCUS command, use the syntax appname/filename if the file is not first in the application path. For example:

```
TABLE FILE baseapp/GGSALES
...
```

No allocation or definition is needed for referencing the standard file types. Other types of files that you may create within an application must be allocated or defined before being used. FOCUS files can be defined with a USE command.

Application Commands Overview

This topic lists the platform-independent application (APP) commands that enable you to control the application environment.

You can use the following wildcard characters in the file name and file type references for the APP commands COPYFILE, MOVEFILE, DELETEFILE, and RENAMEFILE.

- An asterisk (*) replaces any combination of characters of any length (including zero).

  Note that an asterisk can also be used to replace the entire filename or filetype parameter.

- A question mark (?) replaces zero or one character.

Reference: APP Commands Quick Reference

Click any command in the following charts to access detailed information, including the required syntax.

Search Path Management Commands

Command	Description
APP PATH	Sets or resets the application search path.
APP PREPENDPATH	Temporarily adds application names to the beginning of an existing APP PATH search path.
APP APPENDPATH	Temporarily adds application names to the end of an existing APP PATH search path.
APP MAP	Defines a virtual application that points to a physical location outside of the approot structure or redirects an application to another application. This command makes the virtual or redirected application available for addition to the search path. It does not automatically add it to the APP PATH.
APP SET METALOCATION_SAME	Indicates whether corresponding Master and Access files must be in the same application directory.
APP ? METALOCATION_SAME	Retrieves the value of APP SET METALOCATION_SAME.
APP SHOWPATH	Lists all the currently active applications in the search path.

Application Management Commands

Command	Description
APP COPY	Copies the contents of one application to a second application.
APP CREATE	Creates an application under the approot location.
APP DELETE	Deletes an application.
APP MOVE	Moves the contents of one application to a second application.
APP PROPERTY CODEPAGE	Specifies a code page for files in an application.
APP RENAME	Renames an application.

File Management Commands

Command	Description
APP COPYF[ILE]	*Copies a single component or component type from one application to another.
APP MOVEF[ILE]	*Moves a single component or component type from one application to another.
APP RENAMEF[ILE]	*Renames a single component or component type in an application.
APP DELETEF[ILE]	*Deletes a single component or component type from an application.

* The shortened form of the APP commands is used in the remainder of this document.

Output Redirection Commands

Command	Description
APP HOLD	Controls where output files are created for any HOLD, SAVE, SAVB, CREATE SYNONYM, or APP QUERY HOLD process in the application, unless a FILEDEF command has been used to allocate data files.
APP HOLDDATA	Designates an application as the location for temporary data files created with the HOLD, SAVE, or SAVB command.
APP HOLDMETA	Designates an application as the location for temporary Master and Access Files created with the HOLD command.
APP FI[LEDEF]	This command has been deprecated and aliased to FILEDEF.

Help Commands

Command	Description
APP HELP	Displays a list of APP commands with a brief description of each.

Reference: Application Metadata Commands and Metadata Tables

Click any command in the following chart to access detailed information, including the required syntax.

Command	Description
STATE app/file.extension	Check file existence
APP LIST app [HOLD]	Lists the applications under approot. If the HOLD option is used, it lists the applications under approot and writes the output to a temporary file called focappl.ftm, which you can then use in a report request.

Command	Description
APP QUERY app [HOLD]	Lists all files in the application. If the HOLD option is used, it lists all files in the application and writes the output to a temporary file called focappq.ftm, which you can then use in a report request.
catalog/sysfiles	Table, list of accessible app name objects on path for a given type (default MASTER).
catalog/sysdirs	Table, recursive list of physical files under a physical directory.
catalog/sysapps	Table, metadata for physical objects on path.
catalog/systables	Table, app name of tables (and related metadata) on path.

For information about reporting from system tables, see the *Developing Reporting Applications* manual.

Search Path Management Commands

FOCUS has a default search path for application and system components. You can supplement this search path by using one or more of the following APP commands:

- APP PATH
- APP PREPENDPATH
- APP APPENDPATH
- APP SET METALOCATION_SAME
- APP ? METALOCATION_SAME

Generally, these commands add applications to the beginning of the default search path. The exception is temporary components that are created in the current session. These temporary components are searched first, before the user-defined path.

APP PATH

The APP PATH command sets the search path to a designated list of application names that refer to applications under the approot value. You can specify multiple application names to extend the search path.

Syntax: How to Add an Application to the Search Path Manually

```
APP PATH app1[/] [app2[/] ...]
    [appn[/]]
```

where:

app1...appn

Are application names. If you follow an application name with a slash (/), nested applications (the subtree of applications below the named application) will not be in the search path. If you do not follow the application name with a slash, the *nested_app* parameter in the edaserve.cfg file determines whether nested applications are searched for files referenced in a procedure, and to what level. If you need to specify more application names than can fit on one line, add the continuation character (-) at the end of the first line, and code more application names on the next line.

Note:

- You can use the APP PATH command without an application name to reset the search path to the initial list.
- APP PATH does not validate the application list.

APP PREPENDPATH

The APP PREPENDPATH command enables you to temporarily add application names to the beginning of an existing APP PATH search path.

If you wish to use this command to alter the search path, you must code it manually in your application.

Syntax: How to Add Application Names to the Beginning of a Search Path

```
APP PREPENDPATH app1[/] [app2[/]] ...
   [appn[/]]
```

where:

app1...appn

Are application names. If you follow an application name with a slash (/), nested applications (the subtree of applications below the named application) will not be in the search path. If you do not follow the application name with a slash, the *nested_app* parameter determines whether nested applications are searched for files referenced in a procedure, and to what level. If you need to specify more application names than can fit on one line, add the continuation character (-) at the end of the first line, and code more application names on the next line.

APP APPENDPATH

The APP APPENDPATH command enables you to temporarily add application names to the end of an existing APP PATH search path.

If you wish to use this command to alter the search path, you must code it manually in your application.

Syntax: How to Add Application Names to the End of a Search Path

```
APP APPENDPATH app1[/] [app2[/]] ... [appn[/]]
```

where:

app1...appn

Are application names. If you follow an application name with a slash (/), nested applications (the subtree of applications below the named application) will not be in the search path. If you do not follow the application name with a slash, the *nested_app* parameter determines whether nested applications are searched for files referenced in a procedure, and to what level. If you need to specify more application names than can fit on one line, add the continuation character (-) at the end of the first line, and code more application names on the next line.

APP MAP

The APP MAP command allows you to assign an application name to a non-approot application anywhere in the file system or to redirect an application. The application name becomes a virtual application under approot, which can be referenced in an APP PATH command and any other APP command that takes an application name as a parameter.

Note that mapping does *not* automatically add a directory to the path, it simply makes it available for addition to the search path.

Syntax: How to Map a Physical File Location or Redirect an Application

To map a physical file location outside of approot, the syntax is:

```
APP MAP virtualname real_location
```

where:

virtualname

Is an application name of up to 64 characters that can later be used in an APP PATH command.

real_location

Is a real full path name or DDNAME in the native style of the given operating system. On UNIX and Linux, the location may be a mixed case name, but the MAP virtualname itself is always handle insensitive when used (that is, EX mymap/mytest).

Note that if the real location contains spaces, it must be surrounded by double quotation marks.

To redirect an approot or non-approot application to a different name, the syntax is:

```
APP MAP app1 app2/dir1/dir2/../dirn
```

where:

app1
Can be an existing physical, mapped or linked app. It can also be a new app.

app2
Can be a physical, mapped or linked or non-existent app.

dir1 ... dirn
Are application directory names.

Syntax: How to Map DDNAME Allocations

The syntax for this type of mapping is

```
APP MAP appname file_extension=//dd:ddname;file_extension=//
dd:ddname;...
```

where:

appname

Is the name of the application used to reference this mapping in an APP PATH, APP APPENDPATH, or APP PREPENDPATH command.

file_extension

Is one of the following valid FOCUS file extensions:

```
.mas
.fex
.acx
.htm
.sty
.gif
.psb
```

ddname

Is the ddname of the allocation you wish to map. The allocation can be performed using JCL code or a DYNAM command.

Example: Mapping DDNAME Allocations

```
DYNAM ALLOC FILE MYMAS DA EDAARH.MASTER.DATA SHR REU
APP MAP APP1 MAS=//DD:MYMAS;
APP APPENDPATH APP1
```

By default, FOCUS has an APP MAP command in the EDASPROF file to map the application MVSAPP to the allocations FOCEXEC, MASTER, ACCESS, HTML, FOCSTYLE, GIF, FOCPSB. While allocations of these ddnames are not required for the APP MAP command to be valid, once the ddnames are allocated by JCL or DYNAM commands, they become available for use.

Reference: APP MAP With Universal Naming Convention (UNC)

On platforms that support Universal Naming Convention (UNC), you must use the UNC to designate a network drive to access APP directories. The UNC must:

- Be at least one folder below the initial shared location.
- Not contain spaces unless enclosed in double quotation marks. For example,

```
\\mynode\myshare\accnting
"\\mynode\my share\accnting"
```

APP SET METALOCATION_SAME

The APP SET METALOCATION_SAME command identifies whether Master Files and their corresponding Access Files must be in the same location.

Syntax: How to Control the Location of Synonym Files

```
APP SET METALOCATION_SAME {ON|OFF}
```

where:

ON

Specifies that Master Files and their corresponding Access Files must reside in the same application directory. ON is the default value.

OFF

Specifies that once the Master File for a request is located, FOCUS will use the active search path to find the corresponding Access File.

APP ? METALOCATION_SAME

The APP ? METALOCATION_SAME command queries whether Master Files and their corresponding Access Files must be in the same location.

Syntax: How to Query Whether Synonym Files Must Reside in the Same Location

```
APP ? METALOCATION_SAME
```

If the result of this query command is ON, FOCUS expects to find corresponding Master and Access Files in the same application directory. If the result is OFF, FOCUS uses the active search path to find the Access File that corresponds to a given Master File.

APP SHOWPATH

The APP SHOWPATH command lists all the currently active applications in the search path, including baseapp, which is always last.

Syntax: How to List Active Applications

```
APP SHOWPATH
```

Example: Listing Active Applications in the Search Path

FOCUS is generally installed with two default applications: ibisamp (contains sample files), and baseapp (which can contain any files you create).

The APP SHOWPATH command generates the following output is:

```
ibisamp
baseapp
```

Application and File Management Commands

The APP commands in this section provide management options for applications and their component files.

APP CREATE

In general, the APP CREATE command creates an application under the approot high-level qualifier.

The APP CREATE command can create any number of applications with one command.

Syntax: How to Create an Application Manually

```
APP CREATE app1[/app1a...] [app2[/app2a...] ...
   [appn[/appna...] ] [DROP]
```

where:

app1...appn

Are application names under approot. The application name can be up to 64 characters.

app1a...appna

Are nested application directories, allowed when nested applications are configured. In order to create a nested application, the parent application must already exist.

DROP

Deletes an application if one already exists with the same name as the one to be created, and then creates a new application with that name. Note that any files in the pre-existing application are deleted. Without the DROP option, a message will be generated, and the pre-existing application will not be deleted or changed.

The application name may not contain spaces. If the name contains spaces, each section is understood to be a separate application. If you require a name with spaces, you must create it using another mechanism, such as the Windows Explorer. You can then use the APP MAP command to add it to APPROOT.

If you need to specify more application names than can fit on one line, add the continuation character (-) at the end of the first line, and code more application names on the next line.

The word HOLD cannot be used as an application name.

Syntax:

How to Change Default Characteristics of Component File Types (z/OS Only)

You can change the default characteristics of individual component file types by issuing a DYNAM SET APP command. This command controls the types of component files that are generated for the application when an APP CREATE command is issued. By default, all component file types are generated.

The syntax is

```
DYNAM SET APP FOR filetype [SKIP|CREATE] [POSTFIX aaa.bbb] [parms]
```

where:

filetype

Are the component types that may be affected by this command, in uppercase: FOCEXEC, MASTER, ACCESS, HTML, GIF, FOCSTYLE, MAINTAIN, ETG. You must issue a separate command for each component type you wish to affect.

SKIP

Indicates that the designated file type should not be created when the APP CREATE command is issued.

CREATE

Creates the designated file type when the APP CREATE command is issued. This is the default setting.

POSTFIX

Specifies the lower level qualifier of the DSN (data set name) for the component type. The APPROOT value is used to complete the full DSN, which is expressed as

approotvalue.appname.component_type

The default value for component_type is *filetype.DATA*.

parms

Are the allocation parameters you can set. The default parameter values are:

File Type	Parameter
FOCEXEC	RECFM VB TRKS LRECL 4096 BLKSIZE 27998 SPACE 50 50 DIR 50

File Type	Parameter
MASTER	`RECFM FB TRKS LRECL 80 BLKSIZE 22000 SPACE 50 50 DIR 50`
ACCESS	`RECFM FB TRKS LRECL 80 BLKSIZE 22000 SPACE 50 50 DIR 50`
HTML	`RECFM VB TRKS LRECL 4096 BLKSIZE 27998 SPACE 50 50 DIR 50`
GIF	`RECFM VB TRKS LRECL 4096 BLKSIZE 27998 SPACE 50 50 DIR 50` The GIF file type creates libraries for GIF and JPG files.
FOCSTYLE	`RECFM FB TRKS LRECL 1024 BLKSIZE 27648 SPACE 50 50 DIR 50`
MAINTAIN	`RECFM VB TRKS LRECL 4096 BLKSIZE 27998 SPACE 50 50 DIR 50`
ETG	`RECFM FB TRKS LRECL 80 BLKSIZE 22000 SPACE 50 50 DIR 50`

Example: Changing Default Characteristics of an Application

The following command indicates that GIF files should not be created when the APP CREATE command is issued.

```
DYNAM SET APP FOR GIF SKIP
```

The following command indicates that Procedures (FOCEXECs) should be created when APP CREATE is issued.

```
DYNAM SET APP FOR FOCEXEC TRKS SP 10 20 DIR 30
```

APP COPY

The APP COPY command copies the entire contents of one application to another. The target application must already exist.

Syntax: How to Copy an Application

```
APP COPY app1[/app1a...] app2[/app2a...]
```

where:

app1[/*app1a*...]

Is the application being copied. It can be a nested application name.

app2[/*app2a*...]

Is the application to which the contents of the first application are being copied. It can be a nested application name.

APP COPYF[ILE]

The APP COPYF[ILE] command copies one or more components or component types from one application to another.

Note that if you copy a component manually, you can, optionally, rename it in the process.

If you copy a Master File, the corresponding Access File is also copied. However, copying an Access File (file type FOCSQL) does not automatically copy the corresponding Master File.

Syntax: How to Copy an Application Component Manually

```
APP COPYF[ILE] app1[/app1a...]
  {filename1|*} filetype1   app2 [/app2a...]
  {filename2|*} {filetype2|*} [IFEXIST] DROP
```

where:

app1[/*app1a*...]

Is the application that contains the component to be copied. It can be a nested application name.

filename1

Are the components to be copied. Use an asterisk (*) to copy all components of file type *filetype1*.

You can use the following wildcard characters in the file name and file type references.

- An asterisk (*) replaces any combination of characters of any length (including zero).

  Note that an asterisk can also be used to replace the entire filename or filetype parameter.

- A question mark (?) replaces zero or one character.

filetype1

Is the file type, in uppercase, of the component to be copied.

app2[*/app2a...*]

Is the application to which the named component is being copied. It can be a nested application name.

filename2

Is the component name in the target application, after the copy process. Use an asterisk (*) to propagate the file names from the source application to the target application.

filetype2

Is the component type, in uppercase, in the target application after the copy process. Use an asterisk (*) to propagate the file types from the source application to the target application.

IFEXIST

Ignores any component in the source application that does not exist.

DROP

Overwrites any component already in the target application with the same name and file type as a component being copied.

For a full list of the types of files you can copy with APP commands, see *Designating File Types for APP Commands* on page 171.

APP MOVE

The APP MOVE command moves the entire contents of one application to another. The target application must already exist.

Syntax: How to Move an Application

APP MOVE *app1*[*/app1a...*] *app2*[*/app2a...*]

where:

app1[*/app1a...*]

Is the application being moved. It can be a nested application name.

app2[*/app2a...*]

Is the application to which the contents of the first application are being moved. It can be a nested application name.

APP MOVEF[ILE]

The APP MOVEF[ILE] command moves one or more components or component types from one application to another.

Note that if you move a component manually, you can, optionally, rename it in the process.

If you move a Master File, the corresponding Access File is also moved. However, moving an Access File (file type FOCSQL) does not automatically move the corresponding Master File.

Syntax: How to Move an Application Component Manually

```
APP MOVEF[ILE] app1[/app1a...]
  {filename1|*} filetype1  app2 [/app2a...]
  {filename2|*} {filetype2|*} [IFEXIST] [DROP]
```

where:

app1[*/app1a...*]

Is the application that contains the component to be moved. It can be a nested application name.

filename1

Is the name of the component to be moved. Use an asterisk (*) to move all components of file type *filetype1*.

You can use the following wildcard characters in the file name and file type references.

- An asterisk (*) replaces any combination of characters of any length (including zero).

  Note that an asterisk can also be used to replace the entire filename or filetype parameter.

- A question mark (?) replaces zero or one character.

filetype1

Is the file type, in uppercase, of the component to be moved.

app2[*/app2a...*]

Is the application to which the named component is being moved. It can be a nested application name.

filename2

Is the component name in the target application, after the move process. Use an asterisk (*) to propagate the file names from the source application to the target application.

filetype2

Is the component type, in uppercase, in the target application after the move process. Use an asterisk (*) to propagate the file types from the source application to the target application.

IFEXIST

Ignores any component in the source application that does not exist.

DROP

Overwrites any component already in the target application with the same name and file type as a component being moved.

For a full list of the types of files you can move with APP commands, see *Designating File Types for APP Commands* on page 171.

APP DELETE

The APP DELETE command deletes applications under approot.

Syntax: How to Delete an Application Manually

```
APP DELETE app1[/app1a...] [app2[/app2a...] ...
  [appn[/appna...]]
```

where:

app1[/*app1a...*]... [*appn*[/*appna...*]

Are application names. Nested application names are supported. If you need to specify more application names than can fit on one line, add the continuation character (-) at the end of the first line, and enter additional application names on the next line.

APP DELETEF[ILE]

The APP DELETEF[ILE] command deletes one or more components or component types from an application.

If you delete a Master File, the corresponding Access File is also deleted. However, deleting an Access File (file type FOCSQL) does not automatically delete the corresponding Master File.

Syntax: How to Delete an Application Component Manually

```
APP DELETEF[ILE] app[/appna...] {filename|*} filetype
```

where:

appn[*/appa...*]

Is the application from which the component or component type is being deleted. Nested application names are supported.

filename

Is the name of the component to be deleted. Use an asterisk (*) to delete all files of type *filetype*.

You can use the following wildcard characters in the file name and file type references.

- An asterisk (*) replaces any combination of characters of any length (including zero).

  Note that an asterisk can also be used to replace the entire filename or filetype parameter.

- A question mark (?) replaces zero or one character.

filetype

Is the component type, in uppercase, of the component to be deleted.

For a full list of the types of files you can use with APP commands, see *Designating File Types for APP Commands* on page 171.

APP PROPERTY CODEPAGE

The APP PROPERTY appname CODEPAGE command identifies the codepage to be used for non-data files in the application directory.

Syntax: How to Specify a Code Page for an Application

```
APP PROPERTY app[/appa...] CODEPAGE number
```

where:

app[*/appa...*]

Is an application name. Nested application names are supported.

number

Is the code page number for non-data files in the application.

APP RENAME

The APP RENAME command renames an existing application.

Note: You cannot rename an application if it is active in the search path.

Syntax: How to Rename an Application

```
APP RENAME app1[/app1a...] app2[/app2a...]
```

where:

app1[/*app1a...*]

Is the application name to be renamed. It can be a nested application name.

app2[/*app2a...*]

Is the new application name of up to 64 characters. It can be a nested application name.

Example: Renaming an Application

The following shows app1 being renamed to app2.

```
APP RENAME app1 app2
```

APP RENAMEF[ILE]

The APP RENAMEF[ILE] command renames one or more components in an application.

If you rename a Master File, the corresponding Access File is also renamed. However, renaming an Access File (file type FOCSQL) does not automatically rename the corresponding Master File.

Syntax: How to Rename an Application Component

```
APP RENAMEF[ILE] app[/appa... ] filename1
filename2 filetype [DROP]
```

where:

app[/*appa...*]

Is the name of the application that contains the component being renamed. It can be a nested application name

filename1

Is the file name of the component to be renamed.

You can use the following wildcard characters in the file name and file type references.

- An asterisk (*) replaces any combination of characters of any length (including zero).

Note that an asterisk can also be used to replace the entire filename or filetype parameter.

- A question mark (?) replaces zero or one character.

filename2

Is the new name for the component. The component name may be up to 64 characters.

filetype

Is the file type, in uppercase, of the component to be renamed.

DROP

Overwrites an existing component with the same file name and file type.

For a full list of the types of files you can use with APP commands, see *Designating File Types for APP Commands* on page 171.

Designating File Types for APP Commands

The APP COPYF, APP MOVEF, APP DELETEF, and APP RENAMEF commands enable you to perform their actions on a wide variety of file types.

The following is a comprehensive list of the file types you can use with APP commands and the file extensions associated with the on-disk names for hierarchical file systems.

Note that the file types must be coded in uppercase in any APP command that requires it.

Note: This list reflects file types supported across all Information Builders products and release levels. Particular file types may not be supported in particular releases or with every product.

File Type	File Extension
ACX	.acx
ADR	.adr
AFM	.afm
BMP	.bmp
BST	.bst
cascading style sheet	.css

File Type	File Extension
CONTROL	.ctl
DATA	.dat
DDS	.DDS
DEFAULT	The APP *filename* value is used to derive the physical extension for the APP command, so that unknown user-defined extensions may be supported in an APP command (for example, APP COPYFILE BASEAPP MYFILE.FOO DEFAULT BASEAPP MYFILE FOCEXEC).
DTD	.dtd
EDANLS	.nls
EDAPRFU	.prf
EDAPROF	.prf
EDAPSB	.psb
EPS	.eps
ERRORS	.err
ETG	.etg
ETL	.etl
EXCEL	.xls
FMU	.fmu
FOCCOMP	.fcm
FOCDEF	.def
FOCEXEC OR FEX	.fex
FOCFTMAP	.fmp

File Type	File Extension
FOCPSB	.psb
FOCSQL	.acx
FOCSTYLE	.sty
FOCTEMP	.ftm
FOCUS	.foc
GIF	.gif
HLI	.hli
HTML	.htm
IBICPG	.sl
JPG	.jpg
JS	.js
LSN	.lsn
MAINTAIN	.mnt
MASTER OR MAS	.mas MASTER has a special behavior that any matching Access File (.acx) is also operated upon by the APP command. This is so metadata is operated upon as a matched pair. Use MAS if it is strictly desired to only operate on the Master File and not the Access File.
MHT	.mht
Microsoft Access database	.mdb
MNTPAINT	.mpt
OMI	.omi

File Type	File Extension
PDF	.pdf
PFA	.pfa
PFB	.pfb
PNG	.png
PS	.ps
SMARTLIB	.knb
SQL	.sql
SVG	.svg
TABS	.txt
TDL	.tdl
TRF	.trf
TTEDIT	.tte
TXT	.txt
WINFORMS	.wfm
WSDL	.wsd
XHT	.xht
XLSM	.xlsm
XLSX	.xlsx
XLTM	.xltm
XLTX	.xltx
XML	.xml
XSD	.xsd

File Type	File Extension
XSL	.xsl

Output Redirection Commands

Three APP commands (APP HOLD, APP HOLDDATA, and APP HOLDMETA) along with the FILEDEF and DYNAM commands comprise a class of commands that control where output is stored. In order to redirect output as you wish, it is important to understand the interactions among these commands.

Note: When the same behavior applies for APP HOLD, APP HOLDDATA, and APP HOLDMETA, these commands are referred to collectively as APP HOLD*. Note also that although DYNAM (USS only) and FILEDEF are not members of the APP family of commands, these file allocation commands interact with the APP HOLD* commands. Therefore, where appropriate, these commands are also included in this discussion. APP FI[LEDEF] has been deprecated and aliased to FILEDEF.

The most straightforward of these commands is APP HOLD, which allows you to relocate all output to a particular application. You can use this command with operations that produce output files, such as HOLD, SAVE and SAVB, as well as with CREATE SYNONYM and APP QUERY HOLD. (For details about HOLD, SAVE, and SAVB commands, see the *Creating Reports* manual.)

The APP HOLD* commands are particularly helpful when you are creating permanent files for other applications to use. However, if a command is used at an inappropriate point in the application or if it remains in effect when further steps are performed within the application, the target application may be flooded with intermediate and unintended files. Understanding the behavior of each command and the interactions among them will help you avoid this situation.

Reference: Interactions Among Output Redirection Commands

This chart describes the behavior associated with each redirection command and the interactions among them if multiple commands are used.

Command	Stand Alone	Notes
`APP HOLD`	Redirects all output from HOLD, SAVE, SAVB, CREATE SYNONYM, and APP QUERY HOLD commands to the designated application.	When issued without a specific *appname,* APP HOLD has the effect of turning off the command.
`APP HOLDDATA`	Redirects the *data* from HOLD, SAVE, and SAVB operations to the designated application, but does not redirect the associated *metadata* (see Note 1 after chart).	Overrides APP HOLD.
`APP HOLDMETA`	Redirects the *metadata* from HOLD, SAVE, and SAVB operations to the designated application, but does not redirect the associated *data* (see Note 1 after chart).	Overrides APP HOLD.
`FILEDEF` *ddname* `DYNAM ALLOC` *ddname*	Redirects the *data* from specific HOLD, SAVE, and SAVB operations to the designated target, but does not redirect the associated *metadata* (see Note 1 after chart). The AS phrase must match the ddname. When there is no AS phrase, the *ddname* must match a predefined default name: HOLD for HOLD output files; SAVE for SAVE output files; and SAVB for SAVB output files.	Overrides APP HOLD and APP HOLDDATA.

Command	Stand Alone	Notes
`DYNAM ALLOC HOLDMAST`	Redirects the *metadata* from HOLD, SAVE, and SAVB operations to the designated target (using the HOLDMAST *ddname*), but does not redirect the associated *data* (see Note 1 after chart). The recommended practice is to use this command on a request-by-request basis to avoid overriding previous output. If used as a global setting, previously held output will be overwritten with the same name.	Overrides APP HOLD and APP HOLDMETA.

Note:

- Not all formats have associated metadata. For example, the HOLD FORMAT PDF command does not produce metadata, therefore, there is no metadata to redirect.
- The use of the APP HOLD command to redirect CREATE SYNONYM output is neither necessary nor desirable since the CREATE SYNONYM command directly supports application names using the syntax:

  ```
  CREATE SYNONYM appname/synonym ...
  ```

APP HOLD

The APP HOLD command defines an application in which to hold output data files (and associated Master and Access Files, if applicable) created by a HOLD, SAVE, or SAVB process in the application.

APP HOLD is intended to be used to refresh files that are common for all users of the application. It should not be used for private files since it points to an application area that is used by multiple users. If the same hold name (HOLD or AS *name*, for example) is used, conflicts between users could result.

For related information, see *Interactions Among Output Redirection Commands* on page 176.

Syntax: How to Designate a Storage Location for Temporary Files

```
APP HOLD appname[/appnamea...]
```

where:

appname[/*appnamea*...]

Is the application in which you wish to store output files. It can be a nested application name.

Note: Issuing APP HOLD without an *appname* turns off the effects of the command.

APP HOLDDATA

The APP HOLDDATA command designates an application as the location for storing data files created with the HOLD command. For related information, see *Interactions Among Output Redirection Commands* on page 176.

Syntax: How to Designate a Storage Location for Data Files

```
APP HOLDDATA appname[/appnamea...]
```

where:

appname[/*appnamea*...]

Is the name of the location for the data files created by any write process in the application. It can be a nested application name

APP HOLDMETA

The APP HOLDMETA command designates an application directory as the location for storing Master and Access Files created in the application. For related information, see *Interactions Among Output Redirection Commands* on page 176.

Syntax: How to Designate a Storage Location for Master and Access Files

```
APP HOLDMETA appname[/appnamea...]
```

where:

appname[/*appnamea*...]

Is the name of the location for the Master and Access Files created in the application. It can be a nested application name.

APP FI[LEDEF]

The APP FI[LEDEF] command has been deprecated and aliased to FILEDEF. For information, see the *Stored Procedure Reference* manual.

Application Metadata Commands and Catalog Metadata

Developers may want to write applications that check application metadata and decide a course of action. For example, they may want to check the existence of a file or a file date, and decide on the need for another step, such as recreation of the file. There are multiple ways to accomplish a simple check for file existence or some other attribute, that have evolved over the release history of the product. However, some of these methods have limitations. A good example of this is the STATE command, which uses a native path name for UNIX. This type of path name would not match a Windows file path and, therefore, would require IF THEN ELSE or GOTO logic to issue the correct version of the command for the operating environment, that might be quite cumbersome, depending on how often it is needed.

To solve part of this problem, commands such as STATE, FILEDEF, and DYNAM have been extended to support APP names (that is, issue APP MAP then use STATE mymap/myproc.fex). To deal with more complex issues, such as retrieving a list of available applications (APP names) and files within a particular application, a series of APP commands were developed (APP LIST and APP QUERY). However, as features such as nested applications (sub-directories) were implemented, it became apparent that a much more extended ecosystem for accessing application metadata was needed.

To satisfy this need for extended information, various internal tables were extended or created. Today the catalog/sysapps table is the primary method for accessing application metadata using standard TABLE or SELECT syntax. This is what is used in most internal applications. That is not to say that the prior methods are no longer supported. At times they can provide quick and simple coding for a specific need, but they have limitations (as noted). More complex situations require the use of the newer methods to access information. Additionally, tables such as catalog/systables and catalog/syscolum can provide additional information that is table specific, such as what DBMS a table is using and the data specification of particular columns, but they are beyond the scope of this section. It should also be noted that the newer methods occasionally overlap on how to accomplish a task. For example, a number of the catalog/sys* tables can be used to answer the question of whether a file exists. However, the tables differ from each other in the more detailed information, such as physical or application locations and attributes.

Retrieving Basic Information

The following commands return basic information about files and applications.

STATE

The STATE command lets you check for the existence of a file. The file reference you supply can be the full path native operating system file name, or a file name prefaced with an APP name. This section only described the use of APP-name prefaced files. When an APP name is used, it does not matter if the name was natively created under APPROOT or as an APP MAP name.

If the file does not exist, the STATE command displays a message to that effect. After issuing the STATE command, the &RETCODE system variable contains the value zero (0) if the file exists, or a non-zero value if the file does not exist.

Syntax: How to Check File Existence

```
STATE appname/filename.filetype -TYPE RETCODE &RETCODE
```

where:

appname

Is the application under which the file is stored.

filename

Is the name of the file.

filetype

Is the file type or extension of the file.

If the file exists, the &RETCODE value will be 0 (zero). Otherwise, it will be non-zero and can be used to further direct the logic of the application, typically in a -SET or a -IF command. The STATE command will also output a *not found* message. To suppress this message, use the SET TRMOUT={OFF|ON} command.

For example, the following STATE command checks the existence of the file *myproc.fex* in the *baseapp* application. The STATE command displays a message if the file does not exist. The -TYPE command displays the value zero (0) if the file exists or the value -1 if the file does not exist.

```
STATE baseapp/myproc.fex
-TYPE RETCODE &RETCODE
```

Example: **Checking the Existence of a File With the STATE Command**

The following partial example suppresses the message returned by the STATE command, issues the STATE command to check if the file *myproc.fex* exists in the baseapp application, checks the return code, and creates the file if it does not exist, before continuing with the next step in the application. If the file does exist, the code immediately transfers to the next step in the application, (-RESUME label):

```
SET TRMOUT=OFF
STATE baseapp/myproc.fex
SET TRMOUT=ON
-IF &RETCODE EQ 0 THEN GOTO RESUME;
...
* Some code to create the file goes here
...
-RESUME
```

APP LIST

The APP LIST command alphabetically lists the applications available under the application root, APPROOT, or under an APP MAPped location. It does not care if the APP is on the current application map or not, as it is a raw list of available applications.

Syntax: **How to List the Applications in APPROOT**

```
APP LIST [HOLD]
```

If the HOLD option is used, the output is written to a temporary file called focappl.ftm, (FOCAPPL on z/OS), which can, in turn, be used in a request to drive a report or take an action using the catalog/focappl Master File.

Limitations:

- APP LIST does not display nested application names.
- On operating systems that use case sensitive file names (such as UNIX), uppercase physical directory names are not valid (so are not returned by APP LIST). APP names are case insensitive, but they are created on disk as lowercase, which may in turn be upper-cased by the native operating system. However, APP LIST returns them in lowercase, to be homogenous across operating systems.

Example: **Using APP LIST to List and Work with Applications**

The following request lists applications

```
APP LIST
```

The APP LIST output is:

```
BEGIN-APP-LIST
15/02/2000  13.36.38   baseapp
15/02/2000  13.36.38   ggdemo
15/02/2000  13.36.38   ncp
15/02/2000  13.36.38   template
END-APP-LIST
```

The following request lists applications that have been stored using the HOLD option

```
APP LIST HOLD
SQL SELECT DATE, TIME, APPNAME FROM FOCAPPL;
END
```

The APP LIST output is:

```
DATE        TIME      APPNAME
----        ----      -------
15/02/2000  13.36.38   baseapp
15/02/2000  13.36.38   ggdemo
15/02/2000  13.36.38   ncp
15/02/2000  13.36.38   template
```

The following practical example of using the APP LIST HOLD command issues a TABLE request against the HOLD file to check if any files exist in the application *myapp*. If no lines are returned, the application does not exist, so it is created, and the application continues. Otherwise, the application continues without creating the application.

```
APP LIST HOLD
TABLE FILE FOCAPPL
PRINT * ON TABLE HOLD WHERE APPNAME = 'myapp'
END
-IF &LINES GT 0 THEN GOTO RESUME
APP CREATE myapp
-RESUME
```

APP QUERY

The APP QUERY command lists files within a given application. Applications and specific nested applications can be queried.

Syntax: How to List Components

```
APP QUERY app1[/app1a...]  [app2[/app2a]...] ...
  [appn[/appna]] [HOLD]
```

where:

app1[*/app1a...appn*[*/appna*]

Are application names. They can be nested application names. If you need to specify more application names than can fit on one line, add the continuation character (-) at the end of the first line, and continue more application names on the next line.

If the HOLD option is used, the output is written to a temporary file called focappq.ftm (FOCAPPQ on z/OS), which can, in turn, be used in a request to drive a report or take an action using the catalog/focappq Master File.

Limitations: All files within an APP are listed. On systems like UNIX, this may include files of any case, so files such as MYPROC.FEX and myproc.fex may appear in a listing, but only the lowercase version would be accessed in a request.

Example: Listing Application Files

The following request lists application files.

```
APP QUERY abc
```

The APP QUERY output is:

```
BEGIN-APP-QUERY: abc
24/10/2014 21.38.28        4 F myproc1.fex
24/10/2014 21.38.35        4 F myproc1.fex
24/10/2014 21.37.49        4 F myapp1
24/10/2014 21.32.36        0 D myapp2
END-APP-QUERY
```

The following request lists files that have been stored using the HOLD option.

```
APP QUERY ABC HOLD
SQL SELECT DATE, TIME, GMTTIME, SIZE, OTYPE, FILENAME, APPNAME FROM
FOCAPPQ ;
END
```

The APP QUERY output is:

```
DATE        TIME      GMTTIME     SIZE  OTYPE FILENAME     APPNAME
----        ----      -------     ----  ----- --------     -------
24/10/2014  21.38.28  1414201108  4     F     myproc1.fex  abc
24/10/2014  21.38.35  1414201115  4     F     myproc2.fex  abc
24/10/2014  21.37.49  1414201069  4     D     myapp1       abc
24/10/2014  21.32.36  1414200756  0     D     myapp2       abc
```

Note that APP QUERY ... HOLD returns a slightly extended type of information. Whitespace has selectively been removed from the above output for readability (the FILENAME column is actually 70 characters wide).

The following practical example of using the APP QUERY HOLD command checks the existence of the file *myproc1.fex* in application *abc*. If the file does not exist, the procedure exits. If the file does exist, the procedure continues.

```
APP QUERY abc HOLD
TABLE FILE FOCAPPQ
PRINT * ON TABLE HOLD
WHERE APPNAME = 'abc'
WHERE FILENAME = 'myproc1.fex'
END
-IF &LINES GT 0 THEN GOTO RESUME
-TYPE Procedure Not Found ... exiting!
-EXIT
-RESUME
```

Retrieving Extended Catalog Information

This section provides basic information about querying the FOCUS catalogs.

For more information, see *Reporting Dynamically From System Tables* on page 355.

catalog/sysapps

The catalog/sysapps table contains metadata for physical objects on path.

This section only touches on basic uses typically needed by a developer. The Master File on disk robustly describes more attributes than are described here. You can directly study the Master File in order to understand other uses. The catalog/sys* group of files are subject to change (and are usually upwardly compatible). You should never write applications that have specific dependencies (typically on object size), which tend to cause upward compatibility issues.

Example: Listing Files in an APP

The following request lists the application name, application location, file names, and file extensions in the application named *abc*.

```
TABLE FILE SYSAPPS
PRINT APPNAME APPLOC FNAME FEXT
WHERE APPNAME EQ 'abc' ;
END
```

The output (with whitespace selectively removed for readability) is:

```
APPNAME     APPLOC                    FNAME       FEXT
-------     ------                    -----       ----
abc         /usr/wf/ibi/apps/abc      myproc1     fex
abc         /usr/wf/ibi/apps/abc      myproc2     fex
```

The following practical example of using the SYSAPPS table to check file existence checks the existence of the file *myproc1.fex* in the application *abc*. If it does not exist, the procedure exits. If the file does exist, the procedure transfers to the next step in order to continue:

```
TABLE FILE SYSAPPS
PRINT * ON TABLE HOLD
WHERE APPNAME = 'abc' ;
WHERE FNAME =  'myproc1' ;
WHERE FEXT = 'fex' ;
END
-IF &LINES GT 0 THEN GOTO RESUME
-TYPE Procedure Not Found ... exiting!
-EXIT
-RESUME
```

catalog/sysfiles

The catalog/sysapps table contains metadata for app name objects on a path for a select object type. The default is for file type MASTER (Master Files), but is settable for other types. Unless limited in some way, all objects (of the selected type) are displayed.

This section only touches on basic uses typically needed by a developer. The Master File on disk robustly describes more attributes than are described here. You can directly study it in order to understand other uses. The catalog/sys* group of files are subject to change (and are usually upwardly compatible). You should never write applications that have specific dependencies (typically on object size), which tend to cause upward compatibility issues.

Example: **Listing APP MASTER Objects**

The following request lists file names, file names with their application paths, and extensions of files with file type MASTER (the default):

```
TABLE FILE SYSFILES
PRINT FILENAME LGNAME PHNAME EXTENSION
END
```

The output (with some records and whitespace selectively removed for readability) is:

```
FILENAME         LGNAME    PHNAME                          EXTENSION
--------         ------    ------                          ---------
...
mydata           MASTER    baseapp/mydata.mas              mas
mdschema         MASTER    _edahome/catalog/mdschema.mas   mas
```

Example: Listing APP FOCEXEC Objects

The following request sets the file type to FOCEXEC and then prints the file names, file names with their application paths, and extensions of files with file type FOCEXEC:

```
SQL FMI SET SYSFILES FOCEXEC
TABLE FILE SYSFILES
PRINT FILENAME LGNAME PHNAME EXTENSION
END
```

The output (with some records and whitespace selectively removed for readability) is:

```
FILENAME         LGNAME    PHNAME                          EXTENSION
--------         ------    ------                          ---------
...
myproc1          FOCEXEC   baseapp/myproc1                 fex
myproc2          FOCEXEC   baseapp/myproc2                 fex
...
```

Note: The value for LGNAME will switch to DEFAULT if the data is limited and only one object returns.

A valid value for the SQL FMI SET SYSFILES command is any valid FOCUS file type. Some examples are FOCUS, FOCEXEC, STY, PDF. or ACCESS. For a full list of valid file types, see *Designating File Types for APP Commands* on page 171.

Example: Using the SYSFILES Table to Check File Existence

The following practical example of using the SYSFILES table to check file existence prints the filename *myproc1* with extension *fex* (with the file type set to FOCEXEC). If no lines are returned, the file does not exist and the procedure exits. If the file exists, the procedure transfers to the point at which processing continues.

```
SQL FMI SET SYSFILES FOCEXEC
TABLE FILE SYSFILES
PRINT FILENAME ON TABLE HOLD
WHERE FILENAME = 'myproc1' ;
WHERE EXTENSION = 'fex' ;
END
-IF &LINES GT 0 THEN GOTO RESUME
-TYPE Procedure Not Found ... exiting!
-EXIT
-RESUME
```

APP HELP

The APP HELP command provides help information for all of the APP commands.

Syntax: How to Retrieve Information About APP Commands: APP HELP

```
APP HELP command parameters
```

where:

command

Is any valid APP command.

parameters

Are parameters that are available to or required by the command.

Accessing Metadata and Procedures

Permanent files include metadata and procedures that were either created before the session by another application or remain after the session is over for use by another application.

Search Rules

Unless a file name is fully qualified with the application name, the search sequence is:

1. Applications set using APP HOLDMETA for metadata files, and APP HOLDDATA for hold data files.
2. Applications set in APP PATH (including MVSAPP for z/OS).

3. The *baseapp* application.
4. The EDAHOME/catalog.
5. For stored procedures only: if the file is not found, FOCUS checks to see if the file was allocated with a FILEDEF or DYNAM command, and if so, tries to execute it.

Example: Search Paths

The following commands follow the search path, starting with the application set by the APP HOLDMETA command:

```
APP HOLDMETA APP1
```

When a procedure is executed, and referred to by a one-part name

```
EX ABC
```

the following is executed

```
profile.fex in APP1 application
```

followed by

```
EX APP1/ABC
```

If the procedure ABC is not found in APP1, FOCUS follows the standard search path for procedures to find and execute it.

Creation Rules for Procedure Files

Unless a file name is fully qualified or redirected to another location using an APP HOLD, APP HOLDMETA, APP HOLDDATA, FILEDEF, or DYNAM command, it is created in the temporary application area of the agent and disappears after the agent is released.

For example, on z/OS if DYNAM allocation for HOLDMAST or HOLDACC is present, the metadata files are created in the corresponding PDSs (for example, for a CREATE SYNONYM or TABLE FILE file with HOLD).

For related information, see *Output Redirection Commands* on page 175.

Locating Master Files and Procedures

Once your path is set, you can locate Master Files and procedures using the WHENCE command.

Syntax: How to Locate Files

You can issue the WHENCE command to return the fully-qualified path to the first occurrence of a file in your application path. On z/OS, WHENCE will return the location of the file if it is in an allocated DDNAME, or, if FOCUS is APP-enabled, WHENCE will return the name of the APP PDS. . You can issue the APP WHENCE command to return the name of the first application on your application path in which the file resides.

To return the location of the first occurrence of a Master File, procedure, or other FOCUS file type on your application path, issue the following command

```
WHENCE filename filetype
```

To return the name of the first application on your application path that contains a Master File, procedure, or other FOCUS file type, issue the following command

```
APP WHENCE filename filetype
```

where:

filename

Is the name of the file you are trying to locate.

filetype

Is the type of file you are trying to locate.

Example: Locating Files

The following command returns the location of the first occurrence of the Excel file named pivot_demo.xlsx on the application path.

```
WHENCE pivot_demo xlsx
```

The output is:

```
C:\ibi\apps\retail8205\uploads\pivot_demo.xlsx
```

The following command returns the location of the first occurrence of the Master File GGSALES on z/OS:

```
WHENCE GGSALES MASTER
```

The output is:

```
DD:MASTER(GGSALES)
```

If the command is run in an APP-enabled environment, returns the name of the APP PDS that contains the file. For example:

```
APP PATH IBISAMP
WHENCE GGSALES MASTER
```

The output is:

```
USER1.APPS.IBISAMP.MASTER.DATA(GGSALES)
```

The following command returns the name of the application that contains the first occurrence of the Excel file named pivot_demo.xlsx on the application path.

```
APP WHENCE pivot_demo xlsx
```

The output is:

```
APPNAME: retail
```

Accessing Existing Data Files

You can allocate existing data files using the following methods:

- DATASET keyword in the Master File.
- FILEDEF command for non-FOCUS data sources (FIXED, RMS, VSAM, XML).
- USE command for FOCUS data sources.
- For z/OS, native operating system services, when supported.
- DYNAM command.
- Superseded by JCL DD card.

It is recommended that you use only one method for each allocation.

Creation Rules for Data Files

For a newly created data file, the location is determined as follows:

1. An application set by APP HOLDDATA applies to all HOLD files.
2. For FILEDEF command, one for each data file.
3. For z/OS, native operating system allocations when supported.

The request that caused the file to be created determines the file DCB parameters, such as record length, record format, and so on.

For related information, see *Output Redirection Commands* on page 175.

Example: Sample Allocations by JCL

The following table contains sample allocations by JCL.

VSAM	`//VSAM01 DD DISP=SHR, DSN=qualif.DATA.VSAM` This type of allocation requires the szero = y parameter in the edaserve.cfg file to support sharing of BufferPool Zero.
Fixed	`//FIX01 DD DISP=SHR,DSN=qualif.FIXED.DATA`
PDS	`//MASTER DD DISP=SHR,DSN=qualif.MASTER.DATA`
FOCUS	`//CAR DD DISP=SHR,DSN=qualif.CAR.FOCUS`

Example: Sample DYNAM Commands

The following table contains samples of the DYNAM command.

VSAM	`DYNAM ALLOC FILE QVASM DA qualif.QVSAM.VSAM SHR REUSE`
Fixed	`DYNAM ALLOC FILE FILE1 DA qualif.FILE1.DATA SHR REUSE`
PDS	`DYNAM ALLOC FILE MASTER DA qualif.MASTER.DATA SHR REUSE`
FOCUS	`DYNAM ALLOC FILE CAR DA qualif.CAR.FOCUS SHR REU`

Syntax: How to Issue a FILEDEF Command

```
FI[LEDEF] filedes DISK app/[appa.../]physfile.ftm
```

where:

filedes

Is a file designation (ddname).

app/[appa...]

Is an application name. It can be a nested application name.

physfile.ftm

Is a physical file located in the application.

Syntax: How to Issue a FILEDEF Command to Concatenate Files

```
FI[LEDEF] concatname DISK [app1/]filename1.ext
FI[LEDEF] name2 DISK [app2/]filename2.ext
...
FI[LEDEF] namen DISK [appn/]filenamen.ext
FI[LEDEF] concatname CONCAT name2 ... namen
```

where:

concatname

Is the ddname for one of the files and the name for the concatenated files. Use this name in a request. The individual ddnames will not be available once they are used in a FILEDEF CONCAT command.

name2 ... namen

Are ddnames for the files that will be added to the concatenation.

app1 ... appn

Are application names. They can be nested application names.

filename1.ext ...filenamen.ext

Are the physical file names.

Example: Concatenating Files Using FILEDEF

The following request creates three files, file1.ftm, file2.ftm, and file3.ftm.

```
APP HOLD app1
TABLE FILE WF_RETAIL_LITE
SUM COGS_US REVENUE_US
BY STATE_PROV_NAME
WHERE STATE_PROV_NAME LE 'F'
WHERE COUNTRY_NAME EQ 'United States'
ON TABLE HOLD AS file1 FORMAT ALPHA
END
-RUN
TABLE FILE WF_RETAIL_LITE
SUM COGS_US REVENUE_US
BY STATE_PROV_NAME
WHERE STATE_PROV_NAME GT 'F' AND STATE_PROV_NAME LE 'M'
WHERE COUNTRY_NAME EQ 'United States'
ON TABLE HOLD AS file2  FORMAT ALPHA
END
-RUN
TABLE FILE WF_RETAIL_LITE
SUM COGS_US REVENUE_US
BY STATE_PROV_NAME
WHERE STATE_PROV_NAME GT 'M'
WHERE COUNTRY_NAME EQ 'United States'
ON TABLE HOLD AS file3  FORMAT ALPHA
END
```

The following commands concatenate the three files.

```
FILEDEF FILE1 DISK app1/file1.ftm
FILEDEF FILE2 DISK app1/file2.ftm
FILEDEF FILE3 DISK app1/file3.ftm
FILEDEF FILE1 CONCAT FILE2 FILE3
```

The following procedure issues a request against the concatenated files.

```
TABLE FILE FILE1
SUM COGS_US REVENUE_US
BY STATE_PROV_NAME
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF, SIZE=8,$
END
```

The output is shown in the following image.

STATE_PROV_NAME	COGS_US	REVENUE_US
Alabama	$12,774.00	$18,612.25
Alaska	$20,959.00	$29,991.52
Arizona	$9,915.00	$15,100.19
Arkansas	$10,822.00	$15,000.70
California	$61,009.00	$83,133.90
Colorado	$20,596.00	$28,140.37
Connecticut	$7,044.00	$10,536.34
Delaware	$5,043.00	$6,254.51
District of Columbia	$5,151.00	$7,228.56
Florida	$52,541.00	$72,244.02
Georgia	$27,356.00	$37,988.80
Hawaii	$2,770.00	$3,997.04
Idaho	$10,499.00	$14,785.35
Illinois	$43,464.00	$60,364.26
Indiana	$30,904.00	$42,367.87
Iowa	$8,964.00	$12,316.32
Kansas	$19,279.00	$26,478.06
Kentucky	$9,070.00	$13,290.75
Louisiana	$25,212.00	$35,789.99
Maine	$2,542.00	$3,795.93
Maryland	$23,169.00	$32,319.96
Massachusetts	$16,046.00	$22,463.11
Michigan	$45,308.00	$62,878.30
Minnesota	$35,815.00	$50,591.70
Mississippi	$6,661.00	$10,078.13
Missouri	$30,450.00	$41,776.64
Montana	$9,920.00	$14,315.84
Nebraska	$14,626.00	$19,472.62
Nevada	$10,381.00	$14,380.39
New Hampshire	$3,704.00	$5,576.98
New Jersey	$24,128.00	$33,598.01
New Mexico	$7,590.00	$10,119.98
New York	$49,956.00	$69,800.34
North Carolina	$27,789.00	$39,716.65
North Dakota	$9,478.00	$13,552.55
Ohio	$34,478.00	$49,394.62
Oklahoma	$19,010.00	$26,164.76
Oregon	$15,637.00	$21,709.21
Pennsylvania	$46,154.00	$63,984.09
Puerto Rico	$436.00	$714.97
Rhode Island	$583.00	$818.97
South Carolina	$4,309.00	$7,033.86
South Dakota	$6,684.00	$9,311.33
Tennessee	$19,889.00	$27,732.33
Texas	$101,081.00	$141,345.92
Utah	$7,074.00	$10,599.14
Vermont	$5,212.00	$7,474.33
Virginia	$32,148.00	$44,582.23
Washington	$18,101.00	$24,042.12
West Virginia	$5,001.00	$7,000.03
Wisconsin	$29,173.00	$39,605.50
Wyoming	$5,633.00	$7,136.01

Syntax: How to Issue a FILEDEF Command for a Native MVS Data Set

```
FI filedes DISK "//'NATIVE.MVS.DATASET'"
```

where:

filedes

Is a file designation.

NATIVE.MVS.DATASET

Is a Native MVS data set. It can contain any number of qualifiers, up to 44 characters long.

Syntax: How to Issue a USE Command

The USE command can be issued instead of an allocation command for FOCUS data sources. The USE command is the only mechanism for accessing files on the sink machine.

Example: Sample USE Commands

The USE command supports renaming of Master Files and concatenation of data sets. The USE command is the only mechanism for accessing files on the sink machine.

Renaming a Master File

```
USE
 CAR1 AS CAR
END
```

Concatenating Master Files

```
USE
 CAR1 AS CAR
 CAR2 AS CAR
END
```

Accessing Files on a Sink Machine

```
USE
 CAR1 ON FOCSU01
END
```

Data Set Names

If a data set name satisfies one of the following conditions, FOCUS assumes that it is an MVS file name:

- Data set name starts with "//".

- Data set name contains no "/" and contains at least one "."

Syntax: How to Define a Data Set

The following syntax is supported:

```
DATASET=APP1/physfile.ftm
DATASET='qualif.car.data'
DATASET=qualif.car.data
```

In addition, on z/OS, you can use the following:

GDG files	`FILENAME=CARGDG,SUFFIX=FOCUS,` `DATASET='qualif.CARGDG.FOCUS(0)'`
PDS members	`FILENAME=CARMEMB,SUFFIX=FOCUS,` `DATASET=qualif.CARPDS.DATA(CARMEMB)`
FOCUS, VSAM, Fixed	`FILENAME=CAR,SUFFIX=FOCUS,` `DATASET=//'qualif.CAR.FOCUS'`

Allocating Temporary Files

Temporary files are transient files that disappear after you end a session

For z/OS, you can control the size and location of these temporary metadata files and data files. You can specify that the temporary files reside in MVS data sets, or in hiperspace.

Syntax: How to Allocate Temporary Files

To specify the allocation of your temporary files, issue the following command

```
DYNAM SET TEMP[ALLOC] {MVS|HIPER}
```

where:

`MVS`

Allocates temporary files to MVS data sets.

`HIPER`

Allocates temporary files to hiperspace.

Reference: Usage Notes for Allocating Temporary Files

For z/OS, temporary metadata files can be allocated using a similar procedure to allocating permanent metadata files:

- If DYNAM allocation for HOLDMAST or HOLDACC is present, temporary files are stored in the designated PDSs.
- If DYNAM SET TEMP[ALLOC] MVS is issued; in the default temporary PDSs.
- If DYNAM SET TEMP[ALLOC] HIPER is issued; in the HIPERSPACE.

Syntax: How to Allocate Temporary Files to MVS Data Sets

To alter the default allocation parameters for temporary files for MVS data sets, issue the following command

```
DYNAM SET TEMP[ALLOC] FOR type dynam_parms
```

where:

type

Is one of the following: HOLDACC, HOLDMAST, HOLD SAVE, REBUILD, FOCUS, FOCSORT, OFFLINE, or FOC$HOLD.

dynam_parms

Are regular DYNAM ALLOC parameters to be used as default for that type. Note that DCB parameters, if provided here, will be ignored, since they must be compatible with the file type being written.

This is similar to the functionality of IBITABLA in the SSCTL Server. The defaults should be overwritten for all cases when, in older versions, a private copy of IBITABLA existed containing different values.

Reference: System Defaults for Allocating Temporary Files to MVS Data Sets

System defaults for HOLDMAST and HOLDACC are:

```
TRKS 5  5 DSORG PO DIR 36 NEW REU
```

System defaults for all other types are:

```
CYLS 5 10 DSORG PS NEW REU
```

Syntax: How to Support Long Synonym Names Using DYNAM SET LONGSYNM

FOCUS supports synonym names up to 64 characters. However, PDS member names cannot exceed eight characters. FOCUS accounts for this operating environment limitation with the command DYNAM SET LONGSYNM.

A synonym comprises a Master File and, usually, an Access File. When you create a synonym with a name exceeding eight characters, the LONGSYNM setting currently in effect determines how the long name of the Master File and of the Access File will be handled.

You can issue DYNAM SET LONGSYNM anywhere SET commands are valid, including the global profile (EDASPROF) and a stored procedure (FOCEXEC).

The syntax is

```
DYNAM SET LONGSYNM {MVS|MATCH}
```

where:

`MVS`

Specifies that when you save a synonym with a name exceeding eight characters, FOCUS truncates the name, preserving up to the first six characters, followed by a left curly brace ({) and a suffix number that ensures the name is unique. (FOCUS preserves the original long name within the synonym files.)

For example, if you create a Master File named VERYLONGNAMETEST, it will be saved as VERYLO{0. If you then create a Master File named VERYLONGNAMEPROD, it will be saved as VERYLO{1.

FOCUS chooses a suffix number by taking the next unused number in the sequence for that truncation of a Master File or Access File name. If the next number available for the Master File is different than that available for the Access File, the files will be created with different numbers. For example, if the highest Master File name truncated to VERYLO is VERYLO{8, and the highest Access File name truncated to VERYLO is VERYLO{5, and you create a synonym specifying the name VERYLONGNAMEAGAIN, the new Master File will be saved as VERYLO{9, and the new Access File will be saved as VERYLO{6.

`MATCH`

Works the same as the MVS setting, except that it ensures that the truncated names of a Master File and Access File synonym will always match. That is, they will be named using the same suffix number.

In the example provided for the MVS setting, if SET LONGSYNM had instead been set to MATCH, both the new Master File and the new Access File would have been named VERYLO{9.

Matching names may be a convenience for some people if they manually manage synonym files. It is less efficient than the MVS setting, however.

Syntax: How to Pre-Allocate Temporary Files

You can pre-allocate an individual file for a user, using the following techniques:

- **For UNIX and Linux or to allocate a file stored under USS from FOCUS for Mainframe:**

```
FILEDEF XXX DISK /u/another/area/xxx.dat
```

where:

`/u/another/area`

Has enough free space to hold the file.

- **For FOCUS files:**

You can use the FILEDEF and USE commands to create a FOCUS file.

```
FILEDEF NAME DISK /{pathname}{filename}.foc ......
USE NAME NEW
END
```

Syntax: How to Dynamically Allocate FOCUS Files on z/OS

You can dynamically allocate FOCUS files on z/OS with the USE command. The command is

```
DYNAM ALLOC FILE ddname SPACE
USE ddname AS masterfile
END
```

where:

ddname

Is the DDNAME.

masterfile

Is the Master File name.

If the DDNAME and Master File name are the same, use just the command:

```
DYNAM ALLOC
```

Chapter 3

Managing Flow of Control in an Application

Dialogue Manager is the part of the FOCUS language that controls the execution of your application's components. You can add flexibility to your application design by dynamically managing the flow or control in procedures using Dialogue Manager commands and variables whose values are supplied at run time.

In this chapter:

- Uses for Dialogue Manager
- Dialogue Manager Processing
- Creating a Procedure
- Executing and Terminating a Procedure
- Navigating a Procedure
- Using Variables in a Procedure
- Supplying and Verifying Values for Variables
- Manipulating and Testing Variables
- Using Numeric Amper Variables in Functions
- Debugging a Procedure
- Issuing an Operating System Command
- Dialogue Manager Quick Reference

Uses for Dialogue Manager

The following are ways to use Dialogue Manager to control the flow of your application:

- **Control the execution of a procedure.** Use Dialogue Manager control commands to determine the sequence in which FOCUS commands execute and when and how procedures terminate. For details, see *Executing and Terminating a Procedure* on page 213.

- **Navigate a procedure.** You can conditionally execute requests, repeat execution with program loops, or call another procedure. For details, see *Navigating a Procedure* on page 217.
- **Customize a procedure with variables.** Dynamically change a procedure's execution by including variables whose values depend on user input, developer settings, or system information. You can also test a variable's value, the result of a calculation, the existence of a file, or an operating system condition, and execute or not based on the results of the test. For details, see *Using Variables in a Procedure* on page 230, *Supplying and Verifying Values for Variables* on page 247, and *Manipulating and Testing Variables* on page 275.
- **Issue operating system commands.** You can issue an operating system command to query the environment or load a function and run it. For details on issuing operating system commands, see *Issuing an Operating System Command* on page 307.

You can also use Dialogue Manager commands and variables to:

- **Control passwords.** You can directly assign and change passwords. For details, see *Controlling User Access to Data* on page 211.
- **Send a message to an application user.** You can send a message to the user while a procedure is processing to explain the purpose of the procedure, display results, or present other useful information. For details, see *Sending a Message to the User* on page 210.
- **Test and debug the application.** You can use variables to display command lines as they execute and to test Dialogue Manager command logic. See *Debugging a Procedure* on page 300.

Reference: Overview of Dialogue Manager Commands

For descriptions and syntax, see *Dialogue Manager Quick Reference* on page 307.

Command	Meaning
`-*`	Is a comment line; it has no action.
`-CLOSE ddname`	Closes the specified -READ or -WRITE file.
`-CLOSE *`	Closes all -READ and -WRITE files currently open.
`-CRTCLEAR`	Clears the screen display.

Command	Meaning
-CRTFORM	Initiates full-screen variable data entry.
-DEFAULT -DEFAULTS	Presets initial values for variable substitution.
-EXIT	Executes stacked commands and returns to the FOCUS prompt.
-GOTO	Establishes an unconditional branch.
-HTMLFORM	For use with the Web Interface to FOCUS.
-IF	Tests and branches control based on test results.
-INCLUDE	Dynamically incorporates one procedure in another.
-label	User-supplied name identifying the target for -GOTO or -IF.
-MVS RUN	Same as -TSO RUN.
-PASS	Sets password directly.
-PROMPT	Types a prompt message on the screen and reads a reply.
-QUIT	Exits the procedure without executing stacked commands.
-READ	Reads records from a sequential file.
-READFILE	Reads fields based on a Master File into Dialogue Manager variables.
-REPEAT	Executes a loop.
-RUN	Executes all stacked FOCUS commands and returns to procedure for further processing.
-SET	Assigns a value to a variable.
-TSO RUN	In MVS/TSO, loads and executes a user-written function.

Command	Meaning
`-TYPE`	Types informative messages to the screen or other output device.
`-WINDOW`	Invokes Window Painter, transferring control from the procedure to the specified window file.
`-WRITE`	Writes a record to a sequential file.
`-"..."`	Brackets contents for -CRTFORM display line.
`-? SET parameter &myvar`	Captures the value of a settable parameter in &myvar.
`-? &[variablename]`	Displays the values of currently defined amper variables.

Dialogue Manager Variables Overview

You can write procedures containing variables which values are unknown until run time, allowing a user to customize the procedure by supplying different values each time it executes. Variables fall into two categories:

- **Local and global variables.** Local and global variable values must be supplied at run time. Local variables retain the values only for one procedure. Global variables retain the values across procedures unless you explicitly clear them. They lose the values when you exit from FOCUS. You create a local variable by choosing a name that starts with a single ampersand (&); you create a global variable by choosing a name that starts with a double ampersand (&&).
- **System and statistical variables.** System and statistical variable values are automatically supplied by the system when a procedure references them. System and statistical variables have names that begin with a single ampersand (&). For example, the variable &LINES indicates how many lines of output were produced, and the variable &DATE indicates the current date.

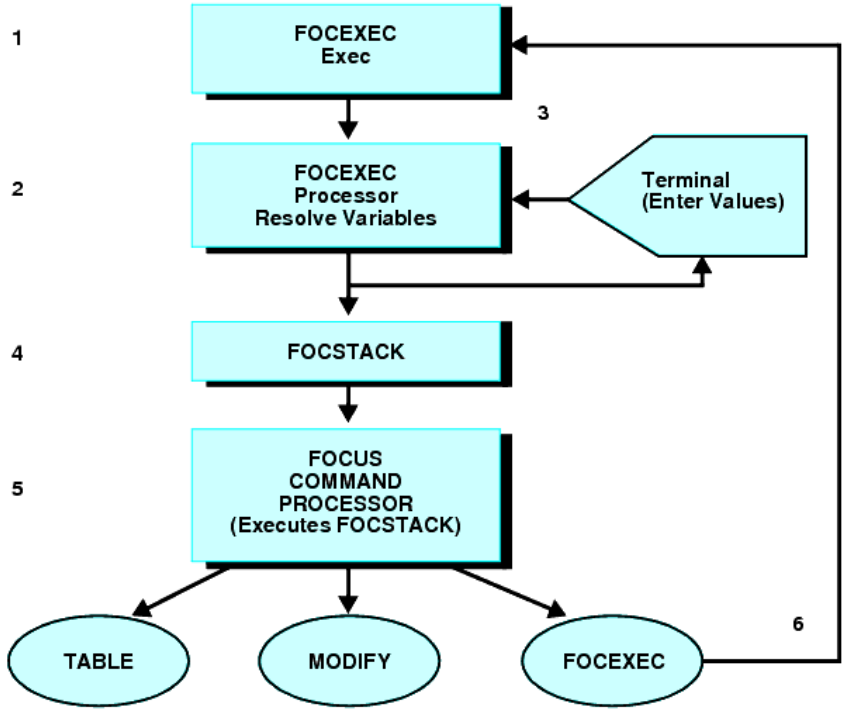
For complete information, see *Using Variables in a Procedure* on page 230, *Supplying and Verifying Values for Variables* on page 247, and *Manipulating and Testing Variables* on page 275.

Dialogue Manager Processing

Modify your application at run time with user input and environment conditions by using Dialogue Manager stored procedures, which include commands and variables.

In the FOCUS community, stored procedures are often referred to as FOCEXECs. In this document they are referred to as *procedures*.

The following diagram illustrates how a Dialogue Manager procedure is processed.



1. Processing begins from the command processor when a procedure is invoked for execution at the FOCUS prompt (for example, EX SLRPT).
2. The FOCEXEC Processor reads each line of the procedure. Any variables on the line are assigned the current values.
3. If a variable is missing a value, FOCUS issues a prompt. The user then supplies the missing value.

   All Dialogue Manager commands execute as soon as Dialogue Manager reads them.
4. When a command line containing no Dialogue Manager commands is fully expanded with any variables resolved (through either a -SET command or prompting), it is placed onto the command execution stack (FOCSTACK).

5. Dialogue Manager execution commands (for example, -RUN) and statistical variables flush the FOCSTACK and route all currently stacked commands to the FOCUS Command Processor.

By the time your FOCSTACK is ready for execution, this has happened:

- All variables have received values and these values have been integrated into the command lines containing variables.
- Dialogue Manager commands have been used to place FOCUS commands into proper sequential order for execution.
- At this point the FOCUS Command Processor no longer sees any Dialogue Manager commands. It only sees FOCUS command lines in the stack.

For an illustration, see *Processing a Procedure*, where the FOCUS Command Processor routes execution to the TABLE module and executes the TABLE request that was stacked.

Note: Any FOCUS command can be placed in a procedure, including the EXEC command. When an EXEC command is processed in a procedure, the commands from the new procedure are first stacked and then executed.

Example: Processing a Procedure

The following example traces the execution process of a procedure. The numbers at the left refer to explanatory notes that follow the example.

```
1. -TOP
2. -PROMPT &WHICHCITY.ENTER NAME OF CITY OR DONE.
3. -IF &WHICHCITY EQ 'DONE' GOTO QUIT;
4. TABLE FILE SALES
   SUM UNIT_SOLD
   BY PROD_CODE
   IF CITY IS &WHICHCITY
   END
5. -RUN
6. -GOTO TOP
7. -QUIT
```

Assume this procedure is stored in a file named SLRPT. To execute it, the user types either of the following:

```
EXEC SLRPT
```

or

```
EX SLRPT
```

The following describes the individual steps of the procedure:

1. `-TOP`

   This is a label, which serves as a target to which -IF ... GOTO or -GOTO commands transfer processing control. Labels call for no special processing, so control passes to the next command.

2. `-PROMPT &WHICHCITY.ENTER NAME OF CITY OR DONE.`

   The prompt "ENTER NAME OF CITY OR DONE" appears on the terminal. Assume the user types "STAMFORD" and the variable value is stored for later use. Processing continues with the next line.

3. `-IF &WHICHCITY EQ 'DONE' GOTO QUIT;`

   Had DONE been entered, control would pass to -QUIT at the bottom of the procedure. This would end processing, cause an immediate exit from this procedure, and return control to the FOCUS prompt. Since STAMFORD was entered, processing continues with the next line.

4. `TABLE FILE SALES`

   ```
   .
   .
   .
   ```

   Without a leading hyphen, this is interpreted as a FOCUS command. Only Dialogue Manager commands execute immediately, so the next five lines are placed in the stack where FOCUS commands are kept until executed; this is referred to as FOCSTACK. Note that the value STAMFORD, entered in response to the prompt, is inserted into the FOCUS command line as the value for &WHICHCITY.

   At this point the FOCSTACK looks like:

   ```
   TABLE FILE SALES
   SUM UNIT_SOLD
   BY PROD_CODE
   IF CITY IS STAMFORD
   END
   ```

   Control passes to the next Dialogue Manager command.

5. `-RUN`

   This command sends the stack to FOCUS, which executes the stored request and returns control to the next Dialogue Manager command.

6. `-GOTO TOP`

   Control is now routed back to -TOP, thus establishing a loop. Execution continues from -TOP with the -PROMPT command.

7. `-QUIT`

   This command is reached when the user types DONE in response to the prompt. The procedure is exited and the FOCUS prompt appears.

Creating a Procedure

You can use the FOCUS integrated text editor, TED, or invoke your system editor from FOCUS with the IEDIT command to create procedures that contain Dialogue Manager functionality. IEDIT is especially useful with variable length files or those whose record lengths are greater than 80 characters.

TED and IEDIT have two valuable features for creating and editing procedures:

- If you issue the TED command, or invoke your system editor using the IEDIT command without specifying a procedure name, the last executed procedure is automatically selected. This is convenient when developing and testing new procedures.
- Test the execution of the procedure by typing RUN on the command line in TED or in a system editor accessed with the IEDIT command. RUN automatically saves the procedure and executes it. If there is an error in your procedure, type TED or IEDIT to bring you back to the editor. It places you directly on the line in which the error was detected.

For details, see *Editing Files With TED* and *Invoking Your System Editor With IEDIT* in the *Overview and Operating Environments* manual.

These options complement the FILE and SAVE options that are common to other editors.

In addition to Dialogue Manager commands and variables that directly affect an application's flow of control, you can use commands to:

- Add comments to a procedure. See *Including Comments in a Procedure* on page 209.
- Send messages to the terminal. See *Sending a Message to the User* on page 210.
- Control user access to data. See *Controlling User Access to Data* on page 211.

You can also create a profile procedure that defines startup conditions and can include Dialogue Manager commands. See *Creating a Startup Procedure* on page 212.

Reference: Rules for Creating Procedures

Follow these general rules when creating procedures:

- Dialogue Manager commands must begin in the first position of the line.

- At least one space must be inserted between the Dialogue Manager command and other text.
- If a Dialogue Manager command exceeds one line, the following line must begin with a hyphen (-). The continuation line must have a space between the hyphen and the rest of the line.
- Procedure names cannot contain special characters.

Including Comments in a Procedure

It is good practice to include comments in a procedure for the benefit of others who may use it. It is particularly recommended that you use comments in a procedure heading to supply the date, the version, and other relevant information. Two styles of comments are available:

- **FOCUS-Style Comments.** A hyphen and an asterisk (-*) mark the beginning of a comment, which can be on a single line.
- **C-Style Comments.** The comment is enclosed within an opening /* tag (slash followed by asterisk) and a closing */ tag (asterisk followed by slash). A C-style comment can appear anywhere and span multiple lines.

Comments do not appear on the terminal nor do they trigger processing. They are visible only when viewing the contents of the procedure through the editor and are strictly for the benefit of the developer. However, you can view comments on the terminal by using the &ECHO variable. For details, see *Debugging a Procedure* on page 300.

Syntax: How to Add a FOCUS-Style Comment in a Procedure

1. Begin the comment line with the command:

   ```
   -*
   ```

2. Type the comment text after the command, optionally with a space before the text.

You can place a comment at the beginning or end of a procedure or in between commands. A comment cannot be on the same line as a command.

The following entry is *valid*:

```
.
.
.
-*Version 2 06/10/00
-RUN
```

The following is *invalid*:

```
-RUN -*Version 2 06/10/00
```

Example: Placing a FOCUS-Style Comment in a Procedure

The following example places a comment at the beginning of a procedure.

```
-* Version 1  08/26/02  HRINFO  Procedure
TABLE FILE CENTHR
   .
   .
   .
```

Example: Placing C-Style Comments in a Procedure

The following example places C-style comments in a procedure.

```
TABLE FILE GGSALES /* this is a multi-line comment
that will not interfere with processing and will be ignored
until the comment is closed with */
SUM /* Another comment */ DOLLARS
   .
   .
   .
```

Sending a Message to the User

You can use the -TYPE command to send a message to the terminal while a procedure is processing. Typically, the message serves the following purposes:

- Explains the purpose of the procedure.
- Displays the results of a procedure or calculation during testing of a procedure.
- Presents other useful information.
- Indicates what type of information to supply in response to a prompt.

Syntax: How to Send a Message to the User

-TYPE sends the message to the terminal as soon as it is encountered in the processing of a procedure. The syntax is

```
-TYPE[+|0|1] text
```

or

```
-label TYPE text
```

where:

text

Is the message to be sent. The message is sent to the screen, followed by a line feed. It remains on screen until scrolled off or replaced by a new screen.

If you include quotation marks around the text, they are displayed as part of the message. (This differs from the use of TYPE in MODIFY, where quotation marks are used as delimiters and must enclose informative text.)

-label

Is the target of a -GOTO or -IF.

+|0|1

Are optional entries that pass printer control characters to the output device. They are particularly useful for character printers. Options + and 1 do not work on IBM 3270-type terminals.

+ suppresses the line feed following the printing of text.

0 forces a line feed before the message text is displayed.

1 forces a page eject before the message text is printed.

If supplied, these values must follow -TYPE without a space.

Example: Sending a Message

The following example illustrates the use of -TYPE to inform a user about the content of a report:

```
-* Version 1  06/26/00  SLRPT  Procedure
-* Component of Retail Sales Reporting Module
-TYPE This report calculates percentage of returns.
TABLE FILE SALES
 .
 .
 .
END
```

Controlling User Access to Data

You can issue and control passwords with the -PASS command. This is especially useful for specifying a password for a particular file or set of files that a given user can read from or write to. Passwords have detailed sets of functions associated with them through the DBA facility.

The procedure that sets passwords can be encrypted so that it and the passwords that it sets cannot be typed and made known.

A variable can also be associated with -PASS so that you can prompt for and assign a password value. You can also check the value of the password and skip or execute a portion of the procedure depending on the value.

Syntax: How to Set a Password in a Procedure

```
-PASS password
```

where:

password

Is a password or a variable containing a password.

Since -PASS is a Dialogue Manager command, it executes immediately and is not sent to the FOCSTACK. This means that the user need not issue the password with the SET command.

Creating a Startup Procedure

You can establish startup conditions in a profile that executes its content immediately upon entry into FOCUS. Using this procedure you can:

- Establish standard conditions that apply throughout the subsequent working session. For example, you can predefine environment parameters or automatically compute variables and make them available for later use.
- Provide a menu of subsequent user options.
- Control use of an application.

You can create a profile using any text editor or the FOCUS editor TED. The file is a procedure (FOCEXEC) named PROFILE.

Note: It is possible to use an alternate procedure as a profile or not to execute a profile at all. For more information, see the *Overview and Operating Environments* manual.

Example: Creating a Startup Profile

The following example creates a startup profile):

```
USE
SALES
EMPLOYEE
END
DYNAM ALLOC DD MYSAV DA USER1.SAVE.TEMP SHR REU
DEFINE FILE SALES
RATIO/D5.2 = (RETURNS/UNIT_SOLD);
END
-TYPE FOCUS SESSION ON &DATE MDYY &TOD
```

```
LET WORKREPORT=TABLE FILE EMPLOYEE
SET LINES=57, PAPER=66, PAGE=OFF
OFFLINE
```

Upon entering FOCUS, the profile is executed and a message, introduced by the -TYPE command, displays the current date and time.

Executing and Terminating a Procedure

You can use Dialogue Manager commands to manage the execution and termination of a procedure. The commands used for these purposes are EXEC, -RUN, -EXIT, -QUIT, and -QUIT FOCUS.

- EXEC executes the named procedure.
- -RUN causes immediate execution of all stacked commands, closes any external files, and continues the procedure. See *Executing Stacked Commands and Continuing the Procedure* on page 214 for more information.
- -EXIT forces the execution of stacked commands, and closes the procedure. For more information, see *Executing Stacked Commands and Exiting the Procedure* on page 215.
- -QUIT cancels execution of any stacked commands and causes an immediate exit from the procedure. For more information, see *Canceling the Execution of a Procedure* on page 216.
- -QUIT FOCUS terminates a procedure and exits FOCUS. For more information, see *Canceling the Execution of a Procedure* on page 216.

Executing Procedures

Procedures are generally initiated from the FOCUS prompt (>). Type the EXEC command followed by the name of the procedure to run.

If you wish to supply arguments for the procedure, see *How to Supply a Variable Value on the Command Line* on page 267.

You can execute a single procedure or call and execute one procedure from within another one. For details, see *Calling Another Procedure With EXEC* on page 228.

Syntax: How to Execute a Procedure

```
EX[EC] procedure
```

where:

procedure

Is the name of the procedure.

Example: Executing a Procedure

To summon a procedure named SLRPT for execution, enter either:

```
EXEC SLRPT
```

or

```
EX SLRPT
```

Executing Stacked Commands and Continuing the Procedure

You can execute stacked commands and continue the procedure with the -RUN command.

The -RUN command causes immediate execution of all stacked commands and closes any external files opened with -READ or -WRITE. For related information, see *Reading Variable Values From and Writing Variable Values to an External File* on page 256.

Following execution of the stacked commands, processing of the procedure continues with the line that follows -RUN.

Example: Executing Stacked Commands and Continuing the Procedure

The following illustrates the use of -RUN to execute stacked code and then return to the procedure. The numbers to the left correspond to the notes explaining the code.

```
1. TABLE FILE SALES
    PRINT PROD_CODE UNIT_SOLD
    BY CITY
   END
2. -RUN
   TABLE FILE EMPLOYEE
    PRINT LAST_NAME FIRST_NAME
    BY DEPARTMENT
   END
```

The procedure processes as follows:

1. The first four lines are the report request. Each line is placed on a stack to be executed later.
2. -RUN causes the stacked commands to be executed and the output returned to the terminal. Processing continues with the line following -RUN.

Executing Stacked Commands and Exiting the Procedure

You can execute stacked commands then exit a procedure with the -EXIT command. -EXIT forces the execution of stacked commands as soon as it is encountered.

-EXIT closes all external files, terminates the procedure, and returns to the FOCUS prompt unless the procedure was called by another procedure, in which case control returns to the calling procedure. For related information, see *Calling Another Procedure With EXEC* on page 228.

Example: Executing Stacked Commands and Exiting the Procedure

In this example, the first report request or the second report request executes, but not both.

```
1. -SET &PROC = 'SALES';
2. -IF &PROC EQ 'EMPLOYEE' GOTO EMPLOYEE;
   -SALES
3. TABLE FILE SALES
   SUM UNIT_SOLD
    BY PROD_CODE
   END
4. -EXIT
   -EMPLOYEE
   TABLE FILE EMPLOYEE
   PRINT LAST_NAME
    BY DEPARTMENT
   END
```

The procedure processes as follows:

1. Dialogue Manager assigns SALES to &PROC.
2. An -IF test is done, and since the value for &PROC is not EMPLOYEE, the test fails and control is passed to the next line, -SALES.

   If the value for &PROC had been EMPLOYEE, control would pass to -EMPLOYEE.
3. The FOCUS code is processed, and stacked to be executed later.
4. -EXIT executes the stacked commands. The output is sent to the terminal and the procedure is terminated.

   The request under the label -EMPLOYEE is not executed.

This example also illustrates an implicit exit. If the value of &PROC was EMPLOYEE, control would pass to the label -EMPLOYEE after the -IF test, and the procedure would never encounter -EXIT. The TABLE FILE EMPLOYEE request would execute and the procedure would automatically terminate.

Canceling the Execution of a Procedure

You can cancel the execution of a procedure with the -QUIT command. -QUIT cancels execution of any stacked commands and causes an immediate exit from the procedure. Control returns directly to the application regardless of whether the procedure was called by another procedure.

This command is useful if tests or computations generate results that make additional processing unnecessary.

You can use a variation, -QUIT FOCUS, to cancel the execution of a procedure and terminate the FOCUS session. It returns you to the operating system and sets a return code.

Syntax: How to Cancel the Execution of a Procedure

```
-QUIT
```

Syntax: How to Cancel the Execution of a Procedure and Exit FOCUS

```
-QUIT FOCUS [n|8]
```

where:

n

Is the operating system return code number. It can be a constant or variable. A variable should be an integer. If you do not supply a value or if you supply a non-integer value, the return code posted to the operating system is 8 (the default).

A major function of user-controlled return codes is to detect processing problems. The return code value determines whether to continue or terminate processing. This is particularly useful for batch processing. For related information, see *Testing the Status of a Query* on page 306.

Example: Canceling the Execution of a Procedure

The following example illustrates the use of -QUIT to cancel execution based on the results of an -IF test:

```
1. -DEFAULT &CODE='B11';
2. -IF &CODE EQ '0' OR &CODE EQ 'DONE' GOTO QUIT;
3. TABLE FILE SALES
   SUM UNIT_SOLD
   WHERE PROD_CODE EQ &CODE
   END
4. -QUIT
```

The procedure processes as follows:

1. The -DEFAULT command sets the default value for &CODE to B11.
2. The value B11 is passed to &CODE.
3. The FOCUS code is processed, and stacked to be executed later.
4. -QUIT cancels the execution of stacked commands and exits the procedure.

Locking Procedure Users Out of FOCUS

Users can respond to a Dialogue Manager value request with QUIT and return to the FOCUS command level or the prior procedure. In situations where it is important to prevent users from entering native FOCUS or QUIT from a particular procedure, the environment can be locked and QUIT deactivated.

Syntax: How to Lock Procedure Users Out of FOCUS

Enter the following command within the procedure:

```
-SET &QUIT=OFF;
```

With QUIT deactivated, any attempt to return to native FOCUS produces an error message indicating that "quit" is not a valid value. The user is prompted for another value.

A user can terminate the FOCUS session from inside a locked procedure by responding to a prompt with

```
QUIT FOCUS
```

to return to the operating system, not the FOCUS command level.

Note: The default value for &QUIT is ON.

Navigating a Procedure

You can navigate a procedure in the following ways:

- **Unconditional branching.** Transfers control to a label. For details, see *Branching Unconditionally* on page 218.
- **Conditional branching.** Transfers control to a label depending on the outcome of a test. For details, see *Branching Conditionally* on page 219.
- **Looping.** Performs a function repeatedly in your procedure. For details, see *Looping in a Procedure* on page 222.

- **Calling another procedure.** Incorporates a whole or partial procedure into your procedure. For details, see *Incorporating Another Procedure With -INCLUDE* on page 225 and *Calling Another Procedure With EXEC* on page 228.

Branching Unconditionally

You can perform unconditional branching, which transfers control to a label with the -GOTO command.

The first time through a procedure, Dialogue Manager notes the addresses of all the labels so they can be found immediately if needed again. If Dialogue Manager hasn't stored the address of the label in the -GOTO command, it searches forward through the procedure for the target label. If no label is found, it begins searching at the top of the procedure.

Dialogue Manager takes no action on labels that do not have a corresponding -GOTO. If a -GOTO does not have a corresponding label, execution halts and an error message is displayed.

Syntax: How to Branch Unconditionally

```
-GOTO label
  .
  .
  .
-label [TYPE text]
```

where:

`-label`

Is a user-defined name of up to 12 characters. Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use arithmetic or logical operations, words that can be confused with functions, or reserved words.

Note: The word CONTINUE can be used as a label in a -GOTO that is not part of a -IF command, but CONTINUE will not be recognized as a label in a -IF command, where it always transfers to the command immediately following the -IF.

The *label* text may precede or follow the -GOTO command in the procedure.

Note: When the *label* is specified in the -GOTO command, a dash does not precede it.

`TYPE text`

Sends a message to the terminal.

Example: Branching Unconditionally

The following example "comments out" all the FOCUS code using an unconditional branch. This is more efficient that placing -* in front of every line:

```
-GOTO DONE
TABLE FILE SALES
PRINT UNIT_SOLD RETURNS
BY PROD_CODE,CITY
END
-RUN
-DONE
```

Branching Conditionally

Conditional branching performs a test of the values of variables and, based on the test, transfers control to a label in the procedure with the -IF... GOTO command. This helps control the execution of requests and builds a dynamic procedure by choosing to execute or not execute parts of a procedure.

For example, you can check whether an extract file was created from a production data source. If the extract file exists, the program runs a set of reports against the extract. If it does not exist, the program branches around the reports and writes a message to a log file.

Note: Generally, an -IF test does not require that each test specify a target label. However, in a compound IF test, where a series of tests are nested within each other, a specified target label is required for each test.

Syntax: How to Branch Conditionally

```
-IF expression [THEN] {GOTO label1|CONTINUE} [ELSE IF...] [ELSE {GOTO
label2|CONTINUE}] ;
```

where:

expression

Is a valid expression. Literals do not need to be enclosed in single quotation marks unless they contain embedded blanks or commas.

THEN

Is an optional word that increases readability of the command.

label1

Is a user-defined name of up to 12 characters to which to pass control if the -IF test is true. Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use arithmetic or logical operations, words that can be confused with functions, or reserved words. The word CONTINUE can be used as a label in a -GOTO that is not part of a -IF command, but CONTINUE will not be recognized as a label in a -IF command, where it always transfers to the command immediately following the -IF.

The *label* text may precede or follow the -IF criteria in the procedure.

CONTINUE

Continues to the command that follows the semicolon of the -IF command.

Note: CONTINUE cannot be used as a label in a -IF statement.

ELSE IF

Specifies a compound -IF test. The command -IF must end with a semicolon to signal that all logic has been specified. For more information, see *Conditional Branching Based on a Compound -IF Test* on page 222.

ELSE GOTO *label2*

Passes control to *label2* when the -IF test fails.

If a command spans more than one line, continuation lines must begin with a hyphen and one or more spaces.

Example: Performing Conditional Branching

The following example passes control to the label -PRODSALES if &OPTION is equal to S. Otherwise, control passes to the label -PRODRETURNS, the next line in the procedure.

```
-IF &OPTION EQ 'S' GOTO PRODSALES;
-PRODRETURNS
TABLE FILE SALES
PRINT PROD_CODE UNIT_SOLD
BY STORE_CODE
END
-EXIT
-PRODSALES
TABLE FILE SALES
SUM UNIT_SOLD
BY PROD_CODE
END
-EXIT
```

The following command specifies both transfers explicitly:

```
-IF &OPTION EQ 'S' GOTO PRODSALES ELSE
- GOTO PRODRETURNS;
```

Notice that the continuation line begins with a hyphen and includes a space after the hyphen.

Example: Conditional Branching Based on Testing of System and Statistical Variables

In the following example, if data (&LINES) is retrieved with the request, then the procedure branches to the label -PRODSALES; otherwise, it terminates.

```
TABLE FILE SALES
SUM UNIT_SOLD
BY PROD_CODE BY CITY
WHERE TOTAL UNIT_SOLD GE 50
ON TABLE HOLD
END
-RUN
-IF &LINES NE 0 GOTO PRODSALES;
-EXIT
-PRODSALES
TABLE FILE SALES
SUM UNIT_SOLD
BY PROD_CODE ACROSS CITY
END
```

Example: Conditional Branching Based on User Input

In the following example, the first report request or the second report request, but not both, executes. Suppose that for the procedure to run a user must supply a value for a variable named &PROC. The user may enter SALES or EMPLOYEE.

```
1. -IF &PROC EQ 'EMPLOYEE' GOTO EMPLOYEE;
2. -SALES
   TABLE FILE SALES
   SUM UNIT_SOLD
    BY PROD_CODE
   END
3. -EXIT
   -EMPLOYEE
   TABLE FILE EMPLOYEE
   PRINT PLANT_NAME
    BY DEPARTMENT
   END
```

The procedure processes as follows:

1. The user enters the value SALES for &PROC. An -IF test is done, and since the value for &PROC is not EMPLOYEE, the test fails and control is passed to the next line, -SALES

   If the value for &PROC had been EMPLOYEE, control would pass to -EMPLOYEE.

2. The FOCUS code is processed, and stacked to be executed later.
3. -EXIT executes the stacked commands. The output is sent to the terminal and the procedure is terminated.

   The request under the label -EMPLOYEE is not executed.

Example: Conditional Branching Based on a Compound -IF Test

A compound -IF test is a series of nested -IF tests nested. In a compound -IF test, each test must specify a target label.

In this example, if the value of &OPTION is neither R nor S, the procedure terminates (-GOTO QUIT). -QUIT serves both as a target label for the GOTO and as an executable command. For the procedure to run, a user must supply a value for a variable named &OPTION.

```
-IF &OPTION EQ 'R' THEN GOTO PRODRETURNS ELSE IF
- &OPTION EQ 'S' THEN GOTO PRODSALES ELSE
- GOTO QUIT;
-PRODRETURNS
TABLE FILE SALES
PRINT PROD_CODE UNIT_CODE
BY STORE_CODE
END
-EXIT
-PRODSALES
TABLE FILE SALES
SUM UNIT_SOLD
BY PROD_CODE
END
-RUN
-QUIT
```

Looping in a Procedure

You can perform an action repeatedly by looping in your procedure with the -REPEAT command. Looping can be used for many tasks. For example, you can populate an indexed variable using a loop or use the output of a request in a second request.

A process loop can be executed a designated number of times or until a condition is met. A loop ends when any of the following occurs:

- It is executed in its entirety.
- A -QUIT or -EXIT command is issued.
- A -GOTO is issued to a label outside of the loop.

  Note: If you issue another -GOTO later in the procedure to return to the loop, the loop proceeds from the point at which it left off.

Note that the -SET command provides another method for implementing loops. See *Controlling Loops With -SET* on page 225.

Tip: During loop processing, the search for labels that indicate the target of a -REPEAT or a -GOTO command takes longer in a procedure with variable, rather than fixed (80 character), record lengths. To speed execution in this situation, consider replacing loops with EX or -INCLUDE commands. See *Incorporating Another Procedure With -INCLUDE* on page 225 and *Calling Another Procedure With EXEC* on page 228.

Syntax: How to Specify a Loop

```
-REPEAT label n TIMES
```

or

```
-REPEAT label WHILE condition;
```

or

```
-REPEAT label FOR &variable [FROM fromval] [TO toval] [STEP s]
```

where:

label

Identifies the code to be repeated (the loop). A label can include another loop if the label for the second loop has a different name than the first.

n TIMES

Specifies the number of times to execute the loop. The value of *n* can be a local variable, a global variable, or a constant. If it is a variable, it is evaluated only once, so you cannot change the number of times to execute the loop. The loop can only be ended early using -QUIT or -EXIT.

WHILE *condition*

Specifies the condition under which to execute the loop. The condition is any logical expression that can be true or false. The loop executes if the condition is true.

&variable

Is a variable that is tested at the start of each execution of the loop and incremented by s with each execution. It is compared with the value of *fromval* and *toval*, if supplied. The loop is executed only if *&variable* is greater than or equal to *fromval* or less than or equal to *toval*.

fromval

Is a constant that is compared with *&variable* at the start of the execution of the loop. The default value is 1.

toval

Is a value that is compared with *&variable* at the start of the execution of the loop. The default value is 1,000,000.

STEP *s*

Is a constant used to increment *&variable* at the end of the execution of the loop. It may be positive or negative. The default increment is 1.

Note: The parameters FROM, TO, and STEP can appear in any order.

Example: Repeating a Loop

These examples illustrate each syntactical element of -REPEAT.

```
-REPEAT label n TIMES
```

For example:

```
-REPEAT LAB1 2 TIMES
-TYPE INSIDE
-LAB1 TYPE OUTSIDE
```

The output is:

```
INSIDE
INSIDE
OUTSIDE
```

```
-REPEAT label WHILE condition;
```

For example:

```
-SET &A = 1;
-REPEAT LABEL WHILE &A LE 2;
-TYPE &A
-SET &A = &A + 1;
-LABEL TYPE END: &A
```

The output is:

```
1
2
END: 3
```

```
-REPEAT label FOR &variable FROM fromval TO toval STEP s
```

For example:

```
-REPEAT LABEL FOR &A STEP 2 TO 4
-TYPE INSIDE &A
-LABEL TYPE OUTSIDE &A
```

The output is:

```
INSIDE 1
INSIDE 3
OUTSIDE 5
```

Example: Controlling Loops With -SET

The following example illustrates the use of -SET to control a loop:

```
1. -DEFAULT &N=0
2. -START
3. -SET &N=&N+1;
4.   EX SLRPT
   -RUN
5. -IF &N GT 5 GOTO NOMORE;
6. -GOTO START
5. -NOMORE TYPE EXCEEDING REPETITION LIMIT
   -EXIT
```

The procedure executes as follows:

1. The -DEFAULT command gives &N the initial value of 0.
2. -START begins the loop. This is also the target of an unconditional -GOTO.
3. The -SET command increments the value of &N by one each time the loop executes.
4. The FOCUS command EX SLRPT is stacked. -RUN then executes the stacked command.
5. The -IF command tests the current value of the variable &N. If the value is greater than 5, control passes to the label -NOMORE, which displays a message for the end user and forces an exit. If the value of &N is 5 or less, control goes to the next Dialogue Manager command.
6. -GOTO passes control to the -START label, and the loop continues.

Incorporating Another Procedure With -INCLUDE

You can insert a whole or partial procedure in another procedure with the -INCLUDE command. A partial procedure might contain heading text, or code that should be included at run time based on a test in the calling procedure. It executes immediately when encountered.

A calling procedure cannot branch to a label in a called procedure, and vice versa. When a procedure is included using the -INCLUDE command, the procedure being included has full access to variables defined in the calling procedure.

The -INCLUDE command can be used for the following:

- Controlling the environment. For example, the included procedure may set variables such as server name or user name before the calling procedure continues execution.
- As a security mechanism. The included procedure can be encrypted and a direct password set.
- Shortening the code when there are several possible procedures that may be called. For example, the command -INCLUDE &NEWLINES could be used to determine the called procedure, reducing the number of GOTO commands.
- Continuing sections of code used throughout the application such as standard headings and footings. This enables changes made in a single module effect the entire application.

Syntax: How to Incorporate a File

```
-INCLUDE filename [filetype]
```

where:

filename

Is the name of a FOCUS procedure.

filetype

Is the procedure's DDNAME. If none is included, FOCEXEC is assumed.

Example: Incorporating Another Procedure With -INCLUDE

In the following example, Dialogue Manager searches for a procedure named DATERPT as specified by the -INCLUDE command.

```
-IF &OPTION EQ 'S' GOTO PRODSALES ELSE GOTO PRODRETURNS;
   .
   .
   .
-PRODRETURNS
-INCLUDE DATERPT
-RUN
   .
   .
   .
```

Assume that DATERPT contains the following code, which Dialogue Manager incorporates into the original procedure. Dialogue Manager substitutes a value for the variable &PRODUCT as soon as the -INCLUDE is encountered. -RUN executes the request.

```
TABLE FILE SALES
PRINT PROD_CODE UNIT_SOLD
WHERE PROD_CODE EQ '&PRODUCT';
END
```

Example: Incorporating a Procedure With a Heading

The following incorporates a heading, which is stored as a procedure:

```
TABLE FILE SALES
-INCLUDE SALEHEAD
SUM UNIT_SOLD AND RETURNS AND COMPUTE
   .
   .
   .
```

The file SALEHEAD contains:

```
HEADING
"THE ABC CORPORATION"
"RETAIL SALES DIVISION"
"MONTHLY SALES REPORT"
```

This heading is included in the report request.

Example: Incorporating a Procedure for a Virtual Field

The following incorporates a virtual field from a procedure:

```
-INCLUDE DEFRATIO
   TABLE FILE SALES
-INCLUDE SALEHEAD
   SUM UNIT_SOLD AND RETURNS AND RATIO
   BY CITY
   .
   .
   .
```

The file DEFRATIO creates a virtual field:

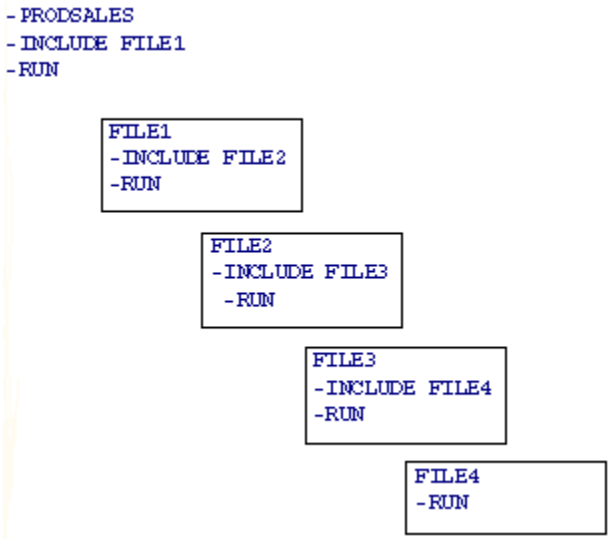
```
DEFINE FILE SALES
RATIO/D5.2=(RETURNS/UNIT_SOLD);
END
```

This virtual field is dynamically included before the report request executes.

Nesting Procedures With -INCLUDE

Any number of different procedures can be invoked from a single calling procedure.

You can also nest a procedure within itself, or recursively. Recursive -INCLUDE commands cannot exceed four levels. For non-recursive -INCLUDE commands, the level of nesting is limited only by the available memory.



Files 1 through 4 are incorporated into the original procedure. All of the included files are viewed as part of the original procedure.

A procedure cannot branch to a label in an included file.

Calling Another Procedure With EXEC

You can call a procedure from another procedure with the EXEC command. The called procedure must be fully executable. It behaves as a completely separate procedure with its own content. It cannot use any local variables (&variables) defined by the *calling* procedure (unless they are explicitly passed to the *called* procedure on the command line). However, the executed (called) procedure can use any global variables (&&variables) that have been defined in the calling procedure.

When an EXEC command is encountered, it is stacked and executed when the appropriate Dialogue Manager command is encountered.

Syntax: How to Call a Procedure With the EXEC Command

```
EX[EC] procedure
```

where:

procedure

Is the name of the procedure.

You can include arguments for the procedure. See *Supplying Variable Values on the Command Line* on page 267.

Note: This syntax is identical to execution syntax for any stored procedure. However, in this context the EXEC command is included within another procedure.

Example: Calling a Procedure With EXEC

In the following example, a procedure calls DATERPT:

```
-IF &OPTION EQ 'S' GOTO PRODSALES ELSE GOTO PRODRETURNS;
    .
    .
    .
-PRODRETURNS
    EX DATERPT
    .
    .
    .
-RUN
```

Note: If the last executable command in the called procedure is a -CRTFORM, control is not returned to the calling procedure unless another Dialogue Manager command is included to terminate the -CRTFORM, such as -RUN or a -label.

Developing an Open-Ended Procedure

A file of stored FOCUS commands without variables looks and executes exactly as though it had been typed interactively into FOCUS from the terminal. However, if there is an error in your procedure file, it will be rejected. If you make an error while typing interactively from the terminal, FOCUS issues prompts to help you correct the error.

If you store a procedure without the END command, you can execute all of the procedure lines. The terminal opens to allow interactive completion of the procedure. You can add additional command lines and enter the END command from the terminal to complete the procedure.

Note that you cannot use amper variables when typing online at a terminal. Open-ended procedures do not support variable substitution in lines entered after the terminal is opened. Variable substitution is supported in the stored portion of the procedure.

Example: Developing and Running an Open-Ended Procedure

Assume the following open-ended procedure is stored as SLRPT:

```
-TYPE ENTER REST OF PROCEDURE
 TABLE FILE SALES
 HEADING CENTER
 "MONTHLY REPORT"
 SUM UNIT_SOLD AND RETURNS AND COMPUTE
 RATIO/D5.2 = 100 * RETURNS/UNIT_SOLD;
```

You can invoke the procedure by typing EX SLRPT. It executes normally but fails to encounter an END command in the file. It then opens up the terminal displaying the FOCUS prompt. You could supply:

```
BY STORE_CODE
END
```

Or, alternatively:

```
IF CITY IS STAMFORD
BY STORE_CODE
END
```

Using Variables in a Procedure

Interactive variable substitution is at the heart of Dialogue Manager. You can create procedures that include variables (also called amper variables) and supply values for them at run time. These variables store a string of text or numbers and can be placed anywhere in a procedure. A variable can refer to a field, a command, descriptive text, a file name—literally anything.

Note: A Dialogue Manager variable contains only alphanumeric data. If a function or expression returns a numeric value to a Dialogue Manager variable, the value is truncated to an integer and converted to alphanumeric format before being stored in the variable, unless you specify the precision to use as described in *How to Specify Precision for Dialogue Manager Calculations* on page 253.

Variables fall into two categories:

- Local and global variables have values supplied at run time. Local variable values remain in effect for the respective procedure, while global variable values remain in effect for all procedures executed during an entire FOCUS session (that is, from the time you enter FOCUS until you exit with the FIN command).

  Leading double ampersands (&&) denote global variables. All other Dialogue Manager variables begin with a single ampersand (&). For this reason, in the FOCUS community they are known as amper variables.

  For details, see *Local Variables* on page 232 and *Global Variables* on page 233.

- System, statistical, and special variables have values that the system automatically resolves whenever you request them.

  For details, see *System Variables* on page 234, *Statistical Variables* on page 242, and *Special Variables* on page 245.

The maximum number of local, global, system, statistical, special, and index variables available in a procedure is 1024. Approximately 40 are reserved for use by FOCUS.

Variables can be used only in procedures. They are ignored if you use them while creating reports live at the terminal.

You can query the values of each type of variable you use. For details, see *Querying the Values of Variables and Parameters* on page 245.

The values for variables may be supplied in a variety of ways. For details, see *Supplying and Verifying Values for Variables* on page 247.

Reference: Naming Conventions for Local and Global Variables

Local and global variable names are user-defined, while system and statistical variables have predefined names. The following rules apply to the naming of local and global variables:

- A local variable name is always preceded by an ampersand (&). The variable can be named or positional.

  A positional variable consists of a single ampersand followed by a numeric string (for example, &1). The value of a positional variable is passed to a procedure when it is executed.

- A global variable name is always preceded by a double ampersand (&&).

- Embedded blanks are not permitted in a variable name.

- If a value for a variable might contain an embedded blank, comma, or equal sign, enclose the variable in single quotation marks when referred to.
- A variable name may be any combination of the characters A through Z, 0 through 9, and the underscore. The first character of the name should be a letter.
- You can assign a number instead of a name to a variable to create a positional variable.
- The underscore may be included in a variable name, but the following special characters are not permitted: plus sign, minus sign, asterisk, slash, period, ampersand, and semicolon.

Syntax: How to Specify a Variable Name

`&[&]`*`name`*

where:

`&`

Denotes a local variable. A single ampersand followed by a numeric string denotes a positional variable.

`&&`

Denotes a global variable.

`name`

Is the variable name. The name you assign must follow the rules outlined in *Naming Conventions for Local and Global Variables* on page 231.

Local Variables

Local variables are identified by a single ampersand (&) preceding the name of the variable. They remain in effect throughout a single procedure.

Example: Using Local Variables

Consider the following procedure, SALESREPORT, in which &CITY, &CODE1, and &CODE2 are local variables:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &CITY"
"PRODUCT CODES FROM &CODE1 TO &CODE2"
" "
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY CITY
IF CITY EQ &CITY
BY PROD_CODE
IF PROD_CODE IS-FROM &CODE1 TO &CODE2
END
```

Assume you supply the following values when you call the procedure:

```
EX SLRPT CITY = STAMFORD, CODE1=B10, CODE2=B20
```

Dialogue Manager substitutes the values for the variables as follows:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR STAMFORD"
"PRODUCT CODES FROM B10 TO B20"
" "
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY CITY
IF CITY EQ STAMFORD
BY PROD_CODE
IF PROD_CODE IS-FROM B10 TO B20
END
```

After the procedure executes and terminates, the values STAMFORD, B10, and B20 are lost.

Global Variables

Global variables differ from local variables in that once a value is supplied, it remains current throughout the FOCUS session unless set to another value with -SET or cleared by the LET CLEAR command. For information on LET CLEAR, see *Defining a Word Substitution* on page 371. Global variables are useful for gathering values at the start of a work session for use by several subsequent procedures. All procedures that use a particular global variable receive the current value until you exit from FOCUS.

Global variables are specified through the use of a double ampersand (&&) preceding the variable name. It is possible to have a local and global variable with the same name. They are distinct and may have different values.

Example: Using Global Variables

The following example illustrates the use of three global variables: &&CITY, &&CODE1, &&CODE2. The values are substituted in the first procedure, PROC1, and the values are retained and passed to the second procedure, PROC2.

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &&CITY"
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY CITY
IF CITY EQ &&CITY
BY PROD_CODE
IF PROD_CODE IS-FROM &&CODE1 TO &&CODE2
END
EX PROC2

TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &&CITY AND PRODUCT &&CODE1"
PRINT UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY CITY
IF CITY EQ &&CITY
IF PROD_CODE EQ &&CODE1
END
```

System Variables

FOCUS automatically substitutes values for system variables encountered in a Dialogue Manager request. For example, you can use the system variable &DATE to automatically incorporate the system date in your request.

System-supplied variables should not be overridden. To avoid this possibility, user-supplied variables should not be given system variables names.

Reference: Summary of System Variables

A list of Dialogue Manager system variables follows:

Variable	Format or Value	Description
&DATE	MM/DD/YY	Returns the current date.

Variable	Format or Value	Description
&DATE*fmt* &DATX*fmt*	Returns the current date or date-time value, where *fmt* can be any valid date or date-time format. &DATE*fmt* retains trailing blanks in the returned value. &DATX*fmt* suppresses trailing blanks in the returned value. **Note:** Using the concatenation symbol (|) to remove punctuation between components is not supported. To return a value without punctuation between the components, use &YYMD or &DATEHYYMDN. For information about date and date-time formats, see Chapter 4, *Describing an Individual Field*, in the *Describing Data* manual.	Returns the current date or date-time value, where *fmt* can be any valid date or date-time format. Because many date format options can be appended to the prefix *DATE* to form one of these variable names, you should avoid using DATE as the prefix when creating a variable name.
&DMY	DDMMYY	Returns the current date.
&DMYY	DDMMCCYY	Returns the current (four-digit year) date.
&ECHO	ON,OFF, ALL, or NONE	Displays command lines as they execute in order to test and debug procedures.
&EXITRC	Any value returned by a command is valid, but zero is considered normal (successful) execution.	Return code value from execution an operating system command. Referencing &EXITRC forces the execution of all stacked commands, like the command -RUN.

Variable	Format or Value	Description
&FOCCODEPAGE		Returns the code page being used by FOCUS.
&FOCCPU	milliseconds	Calculates the OS CPU time.
&FOCEXTTRM	ON OFF	Indicates the availability of extended terminal attributes.
&FOCFEXNAME		Returns the name of the FOCEXEC running even if it was executed using an EX command or a -INCLUDE command from within another FOCEXEC. This variable differs from the &FOCFOCEXEC variable because &FOCFOCEXEC returns the name of the calling FOCEXEC only.
&FOCFIELDNAME	NEW OLD NOTRUNC	Returns a string indicating whether long and qualified field names are supported. A value of OLD means that they are not supported; NEW means that they are supported; and NOTRUNC means that they are supported, but unique truncations of field names cannot be used.
&FOCFOCEXEC		Manages reporting operations involving many similarly named requests that are executed using EX. &FOCFOCEXEC enables you to easily determine which procedure is running. &FOCFOCEXEC can be specified within a request or in a Dialogue Manager command to display the name of the currently running procedure.

Variable	Format or Value	Description
&FOCINCLUDE		Manages reporting operations involving many similarly named requests that are included using -INCLUDE. &FOCINCLUDE can be specified within a request or in a Dialogue Manager command to display the name of the current included procedure.
&FOCMODE	CRJE MSO OS TSO	Identifies the operating environment.
&FOCNEXTPAGE		Establishes consecutive page numbering across multiple reports. When a report is processed, the variable &FOCNEXTPAGE is set to the number following the last page number in the report. This value can then be used as the first page number in a subsequent report, making the report output from multiple requests more useful and readable.
&FOCPRINT	ONLINE OFFLINE	Returns the current print setting.
&FOCPUTLVL	FOCUS PUT level number.	(For example, 9306 or 9310.) &FOCPUTLVL is no longer supported.
&FOCQUALCHAR	. : ! % | \	Returns the character used to separate the components of qualified field names.
&FOCREL	release number	Identifies the FOCUS Release number (for example, 6.5 or 6.8).

Variable	Format or Value	Description
&FOCSBORDER	ON OFF	Whether solid borders are used in full-screen mode.
&FOCTRMSD	24 27 32 43	Indicates terminal height. (This can be any value; the examples shown are common settings.)
&FOCTRMSW	80 132	Indicates terminal width. (This can be any value; the examples shown are common settings.)
&FOCTRMTYP	3270 TTY UNKNOWN	Identifies the terminal type.
&FOCUSER		Returns the connected user ID. Similar to the GETUSER function.
&HIPERFOCUS	ON OFF	Returns a string showing whether HiperFOCUS is on.
&IORETURN		Returns the code set by the last Dialogue Manager -READ or -WRITE operation. (0 = successful; 1= unsuccessful.)
&MDY	MMDDYY	Returns the current date. The format makes this variable useful for numerical comparisons.
&MDYY	MMDDCCYY	Returns the current (four-digit year) date.

Variable	Format or Value	Description
&RETCODE	Any value defined by the FOCUS command. Any value returned by a command is valid, but zero is considered normal (successful) execution. The one exception is the &RETCODE value of dash operating system commands, such as -DOS, -UNIX, and -WINNT, represent the success, not of the command they are running, but of the ability of the server to spawn out to the OS and run the command. In this case, the &RETCODE value is normally zero because it reflects that the spawn executes normally regardless of the results of the specific command. For this case, the amper variable &EXITRC should be used to check the command result or the non-dash version of the command should be used.	numeric &RETCODE executes all stacked commands, like the command -RUN.
&SETFILE	alphanumeric	Contains the value from the SET FILE command.

Variable	Format or Value	Description
&TOD	HH.MM.SS	Returns the current time. When you enter FOCUS, this variable is updated to the current system time only when you execute a MODIFY, SCAN, or FSCAN command. To obtain the exact time during any process, use the HHMMSS function.
&YMD	YYMMDD	Returns the current date.
&YYMD	CCYYMMDD	Returns the current (four-digit year) date.

Example: Retrieving the Date Using the System Variable &DATE

The following example incorporates the system variable &DATE into a request. The footing uses the system variable &DATE to insert the current system date at the bottom of the report.

```
TABLE FILE SALES
SUM UNIT_SOLD
BY PROD_CODE
FOOTING
"CALCULATED AS OF &DATE"
END
```

Example: Retrieving the Procedure Name Using the System Variable &FOCFOCEXEC

This example illustrates how to use the system variable &FOCFOCEXEC in a request to display the name of the currently running procedure:

```
TABLE FILE EMPLOYEE
"REPORT: &FOCFOCEXEC -- EMPLOYEE SALARIES"
PRINT CURR_SAL BY EMP_ID
END
```

If the request is stored as a procedure called SALPRINT, when executed it produces the following:

```
REPORT: SALPRINT -- EMPLOYEE SALARIES
EMP_ID             CURR_SAL
------             --------
071382660        $11,000.00
112847612        $13,200.00
117593129        $18,480.00
119265415         $9,500.00
119329144        $29,700.00
123764317        $26,862.00
126724188        $21,120.00
219984371        $18,480.00
326179357        $21,780.00
451123478        $16,100.00
543729165         $9,000.00
818692173        $27,062.00
```

&FOCFOCEXEC and &FOCINCLUDE can also be used in -TYPE commands. For example, you have a procedure named EMPNAME that contains the following:

```
-TYPE &|FOCFOCEXEC is: &FOCFOCEXEC
```

When EMPNAME is executed, the following output is produced:

```
&FOCFOCEXEC IS: EMPNAME
```

Example: Displaying a Date Using the System Variable &YYMD

You can display a date variable containing a 4-digit year without separators. The variables are &YYMD, &MDYY, and &DMYY.

The following example shows a report using &YYMD:

```
TABLE FILE EMPLOYEE
HEADING
"SALARY REPORT RUN ON DATE &YYMD"
" "
PRINT DEPARTMENT CURR_SAL
BY LAST_NAME BY FIRST_NAME
END
```

The resulting output for May 19, 1999 is:

```
SALARY REPORT RUN ON DATE 19990319

LAST_NAME       FIRST_NAME  DEPARTMENT         CURR_SAL
---------       ----------  ----------         --------
BANNING         JOHN        PRODUCTION       $29,700.00
BLACKWOOD       ROSEMARIE   MIS              $21,780.00
CROSS           BARBARA     OIS              $27,062.00
DAVIS           ELIZABETH   MIS                   $.00
GARDNER         DAVID       PRODUCTION            $.00
GREENSPAN       MARY        MIS               $9,000.00
IRVING          JOAN        PRODUCTION       $26,862.00
JONES           DIANE       MIS              $18,480.00
MCCOY           JOHN        MIS              $18,480.00
MCKNIGHT        ROGER       PRODUCTION       $16,100.00
ROMANS          ANTHONY     PRODUCTION       $21,120.00
SMITH           MARY        MIS              $13,200.00
                RICHARD     PRODUCTION        $9,500.00
STEVENS         ALFRED      PRODUCTION       $11,000.00
```

Statistical Variables

FOCUS posts many statistics concerning overall operations while a procedure executes in the form of statistical variables. As with system variables, FOCUS automatically supplies values for these variables on request.

Reference: Summary of Statistical Variables

A list of Dialogue Manager statistical variables follows:

Variable	Description
&ACCEPTS	Indicates the number of transactions accepted. This variable applies only to MODIFY requests.
&BASEIO	Indicates the number of input/output operations performed.
&CHNGD	Indicates the number of segments updated. This variable applies only to MODIFY requests.

Variable	Description
&DELTD	Indicates the number of segments deleted. This variable applies only to MODIFY requests.
&DUPLS	Indicates the number of transactions rejected as a result of duplicate values in the data source. This variable applies only to MODIFY requests.
&FOCDISORG	Indicates the percentage of disorganization for a FOCUS file. You can use the ? FILE command to display or test this variable, even if the value is less than 30% (the level at which ? FILE displays the amount of disorganization).
&FOCERRNUM	Indicates the last error number, in the format FOCnnnn, displayed after the execution of a procedure. If more than one occurred, &FOCERRNUM holds the number of the most recent error. If no error occurred, &FOCERRNUM has a value of 0. This value can be passed to the operating system with the line -QUIT FOCUS &FOCERRNUM. It can also be used to control branching from a procedure to execute an error-handling routine.
&FORMAT	Indicates the number of transactions rejected as a result of a format error. This variable applies only to MODIFY requests.
&INPUT	Indicates the number of segments added to the data source. This variable applies only to MODIFY requests.
&INVALID	Indicates the number of transactions rejected as a result of an invalid condition. This variable applies only to MODIFY requests.
&LINES	Indicates the number of lines printed in last report. This variable applies only to report requests.
&NOMATCH	Indicates the number of transactions rejected as a result of not matching a value in the data source. This variable applies only to MODIFY requests.

Variable	Description
&READS	Indicates the number of records read from a non-FOCUS file.
&RECORDS	Indicates the number of records retrieved in last report. This variable applies only to report requests.
&REJECTS	Indicates the number of transactions rejected for reasons other than the ones specifically tracked by other statistical variables. This variable applies only to MODIFY requests.
&TRANS	Indicates the number of transactions processed. This variable applies only to MODIFY requests.

Example: Controlling Execution of a Request With the Statistical Variable &LINES

In the following example, the system calculates the value of the statistical variable &LINES. If &LINES is 0, control passes to the TABLE FILE EMPLOYEE request identified by the label -RPT2. If the value is not 0, control passes to the label -REPTDONE, and processing is terminated.

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &CITY"
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY CITY
IF CITY EQ &CITY
BY PROD_CODE
IF PROD_CODE IS-FROM &CODE1 TO &CODE2
END
-RUN
-IF &LINES EQ 0 GOTO RPT2 ELSE GOTO REPTDONE;
-RPT2
TABLE FILE EMPLOYEE
   .
   .
   .
END
-RUN
-QUIT
-REPTDONE
-EXIT
```

Special Variables

FOCUS provides special variables that apply to the cursor, function keys, windows, and other features.

Reference: Summary of Special Variables

A list of special variables follow:

Variable	Description
&CURSOR	Holds the cursor position.
&CURSORAT	Reads the cursor position.
&ECHO	Controls the display of commands for debugging purposes.
&PFKEY	Holds the PF Key function that was pressed or entered.
&QUIT	Controls whether the response QUIT, or PF1 in - CRTFORM, to a prompt causes an exit from the procedure.
&STACK	Controls whether the entire procedure, or only the Dialogue Manager commands are executed.
&WINDOWNAME	Holds the name of the last window activated by the most recently executed -WINDOW command (see *Designing Windows With Window Painter* on page 449).
&WINDOWVALUE	Holds the return value of the last window activated by the most recently executed -WINDOW command (see *Designing Windows With Window Painter* on page 449).

Querying the Values of Variables and Parameters

Two Dialogue Manager commands enable you to:

- Display the values of all types of local, global, and system variables. See *How to Display the Value of a Variable* on page 246.
- Store the value of a parameter in a variable. The stored value can then be queried with the ? SET command. See *How to Store Parameter Value Settings* on page 246.

In addition, you can issue two QUERY (?) commands from the FOCUS prompt to display the values of:

- **Global variables.** Since global variable values remain current throughout the FOCUS session, it is helpful to be able to display the values on demand. The syntax is

  ```
  ? &&
  ```

- **Statistics stored in variables.** You can query the current value of all statistical variables (except &FOCDISORG and &FOCERRNUM). The syntax is:

  ```
  ? STAT
  ```

For details about these commands, see *Testing and Debugging With Query Commands* on page 327.

Syntax: How to Display the Value of a Variable

You can query all Dialogue Manager variables (local, global, system, and statistical) from a stored procedure. The syntax is

```
-? &[&variablename]
```

where:

`&`

Issued alone, displays variables of all types.

`variablename`

Is a complete amper variable or a partial string of up to 12 characters. Only amper variables starting with the specified string are displayed.

The command displays the following message, followed by a list of currently defined amper variables and the values:

```
CURRENTLY DEFINED & VARIABLES:
```

Since local variables do not exist outside a procedure, no similar query is available from the FOCUS command line.

Syntax: How to Store Parameter Value Settings

You can store the current value of a SET parameter in a variable and use the value in a procedure. The syntax is

```
-? SET parameter &[&]variablename
```

where:

parameter

Is any valid FOCUS setting that may be queried with the ? SET or ? SET ALL command. For details about these commands, see *#unique_350*.

variablename

Is the name of the variable where the value is to be stored.

Example: Storing a Parameter Value Setting

If you enter

```
-? SET ASNAMES &ABC
-TYPE &ABC
```

the value stored in &ABC becomes the value of ASNAMES. If you omit &ABC from the command, then a variable called &ASNAMES is created that contains the value of ASNAMES.

Supplying and Verifying Values for Variables

When you design a Dialogue Manager procedure with variables, you must decide how the variables in the procedure acquires values at run time. You can use and/or combine the following techniques.

You can supply variable values directly in procedures, without prompting users for input, using the following methods:

- **-DEFAULT[S]** or **-DEFAULTH** to supply default variable values. See *Supplying a Default Variable Value* on page 250.
- **-SET** to compute a variable value in an expression or to assign a literal value. See *Supplying Variable Values in an Expression* on page 251.
- **-READ** to supply variable values from an external file. See *Reading Variable Values From and Writing Variable Values to an External File* on page 256.
- **EXEC** to supply values on the command line when running a procedure. See *Supplying Variable Values on the Command Line* on page 267.

You can prompt users for variable values using the following methods:

- **-PROMPT** to prompt directly for user input. You can request a set of values before they are needed. You can write your own text for these prompts and validate the entered values to confirm that they fit a preset list of acceptable items or match a predefined format. See *Prompting Directly for Values With -PROMPT* on page 270.

- **-CRTFORM** to prompt for user input on screens. The -CRTFORM command gathers variable values through full-screen data entry. Many values can be input and manipulated at the same time. Several screens can be included in a single procedure and used for a variety of purposes, including the development of menu-driven applications. See *Prompting for Values on Screens With -CRTFORM* on page 271.

  -CRTFORM invokes FIDEL, the FOCUS Interactive Data Entry Language, and incorporates most of its functions. You can also use Screen Painter to design and paint -CRTFORM data entry screens directly on your terminal screen.

  Note that the Dialogue Manager command -CRTFORM is used for entering Dialogue Manager amper variable values. The equivalent MODIFY command, CRTFORM (without a hyphen), is used in MODIFY requests to enter field values.

- **-WINDOW** to prompt for user input in windows you design. You can create a series of menus and windows using the Window Painter facility and display them on the screen using the -WINDOW command. When displayed, the menus and windows can collect data by prompting users to select a value, enter a value, or press a program function (PF) key. See *Prompting for Values on Menus and Windows With -WINDOW* on page 271.

- **Implicit prompting.** FOCUS recognizes variables in a procedure by the leading ampersand (&). If a value has not been provided by some other means, FOCUS automatically requests a value from the terminal when needed. See *Prompting for Values Implicitly* on page 271.

Verifying user input: For values supplied by users, you can also verify input by comparing it against:

- Format specifications. See *Verifying User-Supplied Values Against a Set of Format Specifications* on page 272.

- A pre-defined list of acceptable values. See *Verifying User Input Against a Pre-Defined List of Values* on page 273.

Reference: Rules for Supplying Variable Values

The following rules apply to values for variables:

- If a value contains an embedded comma, equal sign, or blank, you must enclose the variable name in single quotation marks when you use it in an expression. For example, if the value for &LOCATION is BOS, MA, you must refer to the variable as '&LOCATION' in any expression.

- Once a value is supplied for a local variable, it is used throughout the procedure, unless it is changed by -CRTFORM, -PROMPT, -READ, -SET, or -WINDOW.

- Once a value is supplied for a global variable, it is used throughout the FOCUS session in all procedures, unless it is changed by -CRTFORM, -PROMPT, -READ, -SET, or -WINDOW, or cleared by LET CLEAR.
- Dialogue Manager automatically prompts the terminal if a value has not been supplied for a variable.
- The lengths of values stored in Dialogue Manager (amper) variables vary by context:
  - When used with the commands -READ, -TYPE, and WRITE, the maximum length of a variable is approximately 32,000 characters (32K).
  - When used with other Dialogue Manager commands or the EX command, a variable value cannot exceed 4,096 characters (4K).

Example: Supplying Variable Values in a Procedure

This example illustrates the use of the -DEFAULT and -SET commands to supply values for variables. The end user supplies the value B10 for &CODE1, B20 for &CODE2, and SMITH for &REGIONMGR, as prompted by Dialogue Manager.

The numbers to the left of the example apply to the notes that follow:

```
1. -DEFAULT &VERB=SUM
2. -SET &CITY=IF &CODE1 GT 'B09' THEN 'STAMFORD' ELSE 'UNIONDALE';
3. -TYPE REGIONAL MANAGER FOR &CITY
    SET PAGE=OFF
5.      TABLE FILE SALES
        HEADING CENTER
        "MONTHLY REPORT FOR &CITY"
        "PRODUCT CODES FROM &CODE1 TO &CODE2"
        " "
        &VERB UNIT_SOLD AND RETURNS AND COMPUTE
        RATIO/D5.1 = 100 * (RETURNS/UNIT_SOLD);
        BY PROD_CODE
        IF PROD_CODE IS-FROM &CODE1 TO &CODE2
        FOOTING CENTER
4.      "REGION MANAGER: &REGIONMGR"
        "CALCULATED AS OF &DATE"
        END
6. -RUN
```

The procedure executes as follows:

1. The -DEFAULT command sets the value of &VERB to SUM.
2. The -SET command supplies the value for &CITY depending on the value the end user entered in the form for &CODE1. Because the end user entered B10 as the value for &CODE1, &CITY becomes STAMFORD.

3. When the user runs the report, FOCUS writes a message that incorporates the value for &CITY:

```
REGIONAL MANAGER FOR STAMFORD
```

4. The user supplied the value for &REGIONMGR in response to an implicit prompt. FOCUS supplies the current data at run time.
5. The FOCUS stack contains the following lines:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR STAMFORD"
"PRODUCT CODES FROM B10 TO B20"
" "
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.1 = 100 * (RETURNS/UNIT_SOLD);
BY PROD_CODE
IF PROD_CODE IS-FROM B10 TO B20
FOOTING CENTER
"REGION MANAGER: SMITH"
"CALCULATED AS OF 06/11/03"
END
```

6. The -RUN command causes execution of all commands in the stack. The output from the report request is as follows:

```
    MONTHLY REPORT FOR STAMFORD
   PRODUCT CODES FROM B10 TO B20

PROD_CODE  UNIT_SOLD  RETURNS  RATIO
---------  ---------  -------  -----
B10              103       13   12.6
B12               69        4    5.8
B17               49        4    8.2
B20               40        1    2.5

       REGION MANAGER: SMITH
     CALCULATED AS OF 06/11/03
```

Supplying a Default Variable Value

-DEFAULT commands set default values for local or global variables. This technique ensures that a value is passed to a variable so that the user is not prompted for the value.

You can issue multiple -DEFAULT commands for a variable. If the variable is global, these -DEFAULT commands can be issued in separate FOCEXECs. At any point before another method is used to establish a value for the variable, the most recently issued -DEFAULT command will be in effect.

However, as soon as a value for the variable is established using any other method (for example, by issuing a -SET command, retrieving a value input by the user, or reading a value from a file), subsequent -DEFAULT commands issued for that variable are ignored.

Note that -DEFAULTS and -DEFAULTH are synonyms for -DEFAULT.

Syntax: How to Supply a Default Value

```
-DEFAULT[S|H] &[&]name=value [...] [;]
```

where:

name

Is the name of the variable.

value

Is the default value assigned to the variable.

;

Is an optional punctuation character.

Note: -DEFAULTS and -DEFAULTH are synonyms for -DEFAULT.

Example: Supplying a Default Value

In the following example, -DEFAULT sets the default value for &PLANT to Boston (BOS):

```
-DEFAULT &PLANT=BOS
TABLE FILE CENTHR
     .
     .
     .
```

Supplying Variable Values in an Expression

You can assign a variable's value by computing the value in an expression or assigning a literal value to a variable with the -SET command. You can also use the IN FILE phrase to test whether a character value exists in a file and populate a variable with the result. The value of the variable is set to 1 if the test value exists in the file and 0 (zero) if it does not.

You can use this technique to supply dates to Dialogue Manager as variable values. A date supplied to Dialogue Manager in a variable cannot be more than 20 characters long, including spaces. Dialogue Manager variables only accept full-format dates (that is, MDY or MDYY, in any order).

If you are working with cross-century dates that do not include a four-digit year, you can use the SET parameters DEFCENT and YRTHRESH variables to identify the century. For details, see *Working With Cross-Century Dates* on page 397.

If you want to set a variable value to a number, the only supported characters you can use are numeric digits, a leading minus sign, and a period to represent following decimal places. These are the only valid characters that Dialogue Manager supports in a number, regardless of EDIT options or the value of CDN.

Syntax: How to Assign a Value in an Expression

```
-SET &[&]name= {expression|value};

-SET &[&]var3= &var1 IN FILE filename1 [OR &var2 IN FILE filename2 ...];
```

where:

name

Is the name of the variable.

expression

Is a valid expression. Expressions can occupy several lines, so you must end the command with a semicolon.

value

Is a literal value, or arithmetic or logical expression assigned to the variable. If the literal value contains commas or embedded blanks, you must enclose the value in single quotation marks.

&[&]*var3*

Is a variable that is populated with the value 1 if the result of the expression on the right side of the equal sign is true, or with the value 0 if the result is false.

&*var1*

Is the variable that contains the value to be searched for in filename1.

&*var2*

Is the variable that contains the value to be searched for in filename2.

Reference: Usage Notes for IN FILE

- The result of the IN FILE phrase is an alphanumeric value (1 or 0) that can be used in a logical expression connected with AND and OR operators within same –SET command. This value cannot be used as an argument in an alphanumeric operation such as concatenation within the same -SET command.
- In order for IN FILE to return the value 1, the values in the file and the search string must match exactly, starting with the leftmost byte in the file.

- The file can be in any order and have duplicate values. The search stops when either the first match is found or the end of the file is reached. If the file is allocated but does not exist, the value 0 is returned. If the file is not allocated, a FOC351 message displays.
- The length of the variable used in the IN FILE phrase determines the number of bytes from the beginning of each record in the file used for comparison. Only an exact match on that number of bytes will return a 1. Trailing blanks in the variable will require the same number of trailing blanks in the file in order to match. For example, the following will match only the value 'ABC ' (ABC with three trailing blanks):

```
-SET &VAR1 = 'ABC   ';
-SET &VAR2 = &VAR1 IN FILE FILE1;
```

Syntax:

How to Specify Precision for Dialogue Manager Calculations

The DMPRECISION setting enables Dialogue Manager -SET commands to calculate accurate numeric variable values without using the FTOA function.

Without this setting, results of numeric calculations are returned as integer numbers, although the calculations themselves employ double-precision arithmetic. To return a number with decimal precision without this setting, you have to enter the calculation as input into subroutine FTOA, where you can specify the number of decimal places returned.

The SET DMPRECISION command gives users the option of either accepting the default truncation of the decimal portion of output from arithmetic calculations, or specifying up to nine decimal places for rounding.

```
SET DMPRECISION = {OFF|n}
```

where:

`OFF`

Specifies truncation without rounding after the decimal point. OFF is the default value.

n

Is a positive number from 0-9, indicating the point of rounding. Note that n=0 results in a rounded integer value.

- When using SET DMPRECISION, you must include -RUN after the SET DMPRECISION command to ensure that it is set prior to any numeric -SET commands.
- As the actual conversion to double precision follows the rules for the operating system, the values may vary from platform to platform.

Example: Setting Precision for Dialogue Manager Calculations

The following table below shows the result of dividing 20 by 3 with varying DMPRECISION (DMP) settings:

SET DMPRECISION =	Result
OFF	6
0	7
1	6.7
2	6.67
9	6.666666667

Example: Setting a Variable Value in an Expression

In the following example, -SET assigns the value 14Z or 14B to the variable &STORECODE, as determined by the logical IF expression. The value of &CODE is supplied by the user.

```
-SET &STORECODE = IF &CODE GT C2 THEN '14Z' ELSE '14B';
   TABLE FILE SALES
   SUM UNIT_SOLD AND RETURNS
   BY PROD_CODE
   IF PROD_CODE GE &CODE
   BY STORE_CODE
   IF STORE_CODE IS &STORECODE
   END
```

Example: Setting a Literal Value

The use of single quotation marks around a literal is optional unless the literal contains embedded blanks, commas, or equal signs. In these cases, you must include them as illustrated below:

```
-SET &NAME='JOHN DOE';
```

In prior releases, to assign a literal value that included a single quotation mark, you had to place two single quotation marks where you wanted one to appear:

```
-SET &NAME='JOHN O''HARA';
```

Although this technique still works, it is no longer required. However, to start or end a string with a single quotation mark, you must specify two single quotation marks.

Example: Setting the Difference Between Two Dates

This example supplies dates to Dialogue Manager as variables. The variable &DELAY is set to the difference in days between &LATER and &NOW and the result is returned to your terminal.

```
-SET &NOW = 'JUN 30 2002';
-SET &LATER = '2002 25 AUG';
-SET &DELAY = &LATER - &NOW;
-TYPE &DELAY
```

Example: Testing Whether a Variable Value Is in a File

The following FOCEXEC creates an alphanumeric HOLD file called COUNTRY1 with the names of countries from the CAR file. It then sets the variable &C equal to FRANCE. The IN FILE phrase returns the value 1 to &IN1 if FRANCE is in the HOLD file and 0 if it is not:

```
TABLE FILE CAR
PRINT COUNTRY
ON TABLE HOLD AS COUNTRY1 FORMAT ALPHA
END
-RUN
-SET &C = 'FRANCE';
-SET &IN1 = &C IN FILE COUNTRY1;
-TYPE THE VALUE IS &IN1
```

The output shows that FRANCE is in the file COUNTRY1:

```
THE VALUE IS 1
```

Example: Initializing a Variable to a Long String

To set the value of a variable with -SET, you need to specify a character string on the right side of the SET command. Since the character string cannot span multiple lines, if necessary, you can concatenate shorter strings or variables to compose the long string.

The following procedure creates a variable named &LONG that contains a long string:

```
-SET &LONG = 'THIS IS A LONG AMPER VARIABLE. NOTE THAT IN ORDER '
- |'TO SET ITS VALUE USING -SET, YOU MUST CONCATENATE SHORTER STRINGS, '
- |'EACH OF WHICH MUST FIT ON ONE LINE.';
-TYPE &LONG
END
```

The output is:

```
THIS IS A LONG AMPER VARIABLE.NOTE THAT IN ORDER TO SET ITS VALUE USING
-SET, YOU MUST CONCATENATE SHORTER STRINGS, EACH OF WHICH MUST FIT ON
ONE LINE.
```

Reading Variable Values From and Writing Variable Values to an External File

You can read variable values from an external file, or write variable values to an external file with the -READ and -WRITE commands.

- You can supply variable values with the -READ command. For example, an external file may contain the start and end dates of a reporting period. Dialogue manager can read these values from an external file and use them in a variable in a WHERE command that limits the range of data selected in a report request.
- You can save variable values in an external file with the -WRITE command. For example, a request can store the summed total of sales for the day in an external file so that it can be compared to the following day's total sales.

The external file can be a fixed-format file (in which the data is in fixed columns) or a free-format file (in which the data is comma delimited).

You can also read a file using the -READFILE command. The -READFILE command reads a file by first reading its Master File and creating Dialogue Manager amper variables based on the ACTUAL formats for each field in the Master File. It then reads the file and, if necessary, converts the fields from numeric values to alphanumeric strings before returning them to the created variables. Display options in the USAGE formats are not propagated to the variables. The names of the amper variables are the field names prefixed with an ampersand (&).

Syntax: How to Retrieve a Variable Value From an External File

```
-READ ddname[,] [NOCLOSE] &name[.format.][,] ...
```

where:

ddname

Is the logical name of the file as defined to FOCUS using ALLOCATE or DYNAM ALLOCATE.

A space after the ddname denotes a fixed format file while a comma denotes a comma-delimited file.

NOCLOSE

Indicates that the file should be kept open even if a -RUN is encountered. The file is closed upon completion of the procedure or when a -CLOSE or subsequent -WRITE command is encountered.

name

Is the variable name. You may specify more than one variable. Using commas to separate variables is optional.

If the list of variables is longer than one line, end the first line with a comma and begin the next line with a dash followed by a blank (-) for comma-delimited files or a dash followed by a comma followed by a blank (-,) for fixed format files. For example:

Comma-delimited files

```
-READ EXTFILE, &CITY,&CODE1,
- &CODE2
```

Fixed format files

```
-READ EXTFILE &CITY.A8. &CODE1.A3.,
-, &CODE2.A3
```

format

Is the format of the variable. It may be Alphanumeric (A) or Numeric (I). Note that format must be delimited by periods. The format is ignored for comma-delimited files.

Note: -SET provides an alternate method for defining the length of a variable using the corresponding number of characters enclosed in single quotation marks ('). For example, the following command defines the length of &CITY as 8:

```
-SET &CITY='        ';
```

Example: Reading a Value From an External File

Assume that EXTFILE is a fixed-format file containing the following data:

```
STAMFORDB10B20
```

To detect the end of a file, the following code tests the system variable &IORETURN. When no records remain to be read, a value equal to zero is not found.

```
-READ EXTFILE &CITY.A8. &CODE1.A3. &CODE2.A3.
-IF &IORETURN NE 0 GOTO RESUME;
    TABLE FILE SALES
    SUM UNIT_SOLD
    BY CITY
    IF CITY IS &CITY
    BY PROD_CODE
    IF PROD_CODE IS-FROM &CODE1 TO &CODE2
    END
-RESUME
 .
 .
 .
```

Syntax: How to Write a Variable Value to an External File

```
-WRITE ddname [NOCLOSE] text
```

where:

ddname

Is the logical name of the file as defined to FOCUS using ALLOCATE or DYNAM ALLOCATE. For information about file allocations, see the *Overview and Operating Environments* manual.

NOCLOSE

Indicates that the file should be kept open even if a -RUN is encountered. The file is closed upon completion of the procedure or when a -CLOSE or subsequent -READ command is encountered.

text

Is any combination of variables and text. To write more than one line, end the first line with a comma (,) and begin the next line with a hyphen followed by a space (-).

-WRITE opens the file to receiving the text and closes it upon exit from the procedure. When the file is reopened for writing, the new material overwrites the old. To reopen to add new records instead of overwriting existing ones, use the attribute DISP MOD when you define the file to the operating system.

Example: Writing to a File

The following example reopens the file PASS to add new text:

```
DYNAM ALLOC DD PASS DA USER1.PASS.DATA MOD
-WRITE PASS &DIV &RED &TEST RESULT IS,
- &RECORDS AT END OF RUN
```

Example: Reading From and Writing to an External File

The following example illustrates reading from and writing to sequential files. It also illustrates the use of operating system commands. The numbers in the margin refer to notes that follow the example.

```
    SET HOLDLIST=PRINTONLY
    -RUN
1.  -TOP
2.  -PROMPT &CITY.ENTER NAME OF CITY -- TYPE QUIT WHEN DONE.
3.  DYNAM ALLOC DD PASS DA USER1.PASS.DATA LRECL 80 RECFM FB
    -RUN
4.  -WRITE PASS &CITY
        TABLE FILE SALES
        HEADING CENTER
        "LOWEST MONTHLY SALES FOR &CITY"
        " "
        PRINT DATE PROD_CODE
        BY LOWEST 1 UNIT_SOLD
        BY STORE_CODE
        BY CITY
        IF CITY EQ &CITY
        FOOTING CENTER
        "CALCULATED AS OF &DATE"
        ON TABLE SAVE AS INFO
        END

5.  -RUN
6.  DYNAM ALLOC DD LOG DA USER1.LOG.DATA LRECL 80 RECFM FB
        -RUN
        MODIFY FILE SALES
        COMPUTE
        TODAY/I6=&YMD;
        CITY='&CITY';
        FIXFORM X5 STORE_CODE/A3 X15 DATE/A4 PROD_CODE/A3
        MATCH STORE_CODE DATE PROD_CODE
        ON MATCH TYPE ON LOG
        "<STORE_CODE><DATE><PROD_CODE><TODAY>"
        ON MATCH DELETE
        ON NOMATCH REJECT
        DATA ON INFO
        END
7.  -RUN
        EX SLRPT3
8.  -RUN
11. -GOTO TOP
12. -QUIT
```

The procedure SLRPT3, which is invoked from the calling procedure, contains the following lines:

```
9.  -READ PASS &CITY.A8.
        TABLE FILE SALES
        HEADING CENTER
        "MONTHLY REPORT FOR &CITY"
        "LOWEST SALES DELETED"
        " "
        PRINT PROD_CODE UNIT_SOLD RETURNS DAMAGED
        BY STORE_CODE
        BY CITY
        IF CITY EQ &CITY
        FOOTING CENTER
        "CALCULATED AS OF &DATE"
        END
10. -RUN
```

The following annotations explain the logic and show the dialogue between the user and the screen. User entries are in lowercase:

1. -TOP marks the beginning of the procedure.
2. -PROMPT sends the following prompt to the screen after the procedure is executed:

   ```
   ENTER NAME OF CITY -- TYPE QUIT WHEN DONE<STAMFORD
   ```

3. DYNAM defines and opens a file named PASS.
4. -WRITE writes the value of &CITY to the sequential file named PASS. In this case the value written is STAMFORD.
5. -RUN executes the stacked TABLE request. In this case, a sequential file named INFO is created with the SAVE command. This is a sequential file, containing the result of the report request as shown below.

   ```
   NUMBER OF RECORDS IN TABLE=         8  LINES=      8

   ALPHANUMERIC RECORD NAMED  INFO
   FIELDNAME                          ALIAS          FORMAT         LENGTH

   UNIT_SOLD                          SOLD           I5               5
   STORE_CODE                         SNO            A3               3
   CITY                               CTY            A15             15
   DATE                               DTE            A4MD             4
   PROD_CODE                          PCODE          A3               3

   TOTAL                                                              30
   SAVED...
   ```

6. DYNAM defines a log file for the subsequent MODIFY request.
7. -RUN executes the stacked MODIFY request. The data comes directly from the INFO file created in the prior TABLE request and is entered using FIXFORM. Hence, the product with the lowest UNIT_SOLD is deleted from the file, and logged to a log file.

   ```
   SALES    FOCUS    A1 ON 09/04/2003 AT 10.04.35
   ```

```
TRANSACTIONS:          TOTAL =     1  ACCEPTED=     1  REJECTED=     0
SEGMENTS:              INPUT =     0  UPDATED =     0  DELETED =     1
```

8. The next -RUN executes another procedure called SLRPT3.
9. -READ reads the value for &CITY from the sequential file PASS. In this case the value passed is STAMFORD.
10. The -RUN executes the TABLE request and control is routed back to the calling procedure.

```
                    MONTHLY REPORT FOR STAMFORD
                       LOWEST SALES DELETED

STORE_CODE  CITY             PROD_CODE  UNIT_SOLD  RETURNS  DAMAGED
----------  ----             ---------  ---------  -------  -------
14B         STAMFORD         B10               60       10        6
                             B12               40        3        3
                             B17               29        2        1
                             C7                45        5        4
                             D12               27        0        0
                             E2                80        9        4
                             E3                70        8        9

                          CALCULATED AS OF 09/04/03
```

11. -GOTO TOP routes control to the top.
12. When the user types QUIT, processing ends.

Syntax: How to Read Master File Fields Into Dialogue Manager Variables

```
-READFILE mastername
```

where:

mastername

Is the name of the Master File to be read.

Reference: Usage Notes for -READFILE

- A -RUN command does not close the file. You must issue a -CLOSE command to close the file. You must be careful not to delete, change, or re-allocate the file before closing it.
- If multiple fields have the same field name, only one variable is created, and it contains the value of the last field in the Master File.
- -READFILE does not work if the Master File contains DBA restrictions. The following message is generated:

```
(FOC339) DIALOGUE MANAGER -READ FAILED: CHECK FILEDEF OR ALLOCATION FOR:
-READFILE filename
```

- -READFILE is not supported with text fields. The following message is generated:

```
(FOC702) THE OPTION SPECIFIED IS NOT AVAILABLE WITH TEXT FIELDS:
fieldname
```

- -READFILE cannot read XFOCUS data sources. The file to be read must have an associated Master File.

Example: Reading Fields From a Data Source Into Dialogue Manager Variables Using -READFILE

The following request creates a binary HOLD file, then uses -READFILE to read the first record from the HOLD file and type the values that were retrieved into Dialogue Manager variables. Note that the names of the variables are the field names prefixed with an ampersand:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME DEPARTMENT CURR_SAL
BY EMP_ID
ON TABLE HOLD AS READF1 FORMAT BINARY
END
-RUN
-READFILE READF1
-TYPE LAST_NAME  IS &LAST_NAME
-TYPE FIRST_NAME IS &FIRST_NAME
-TYPE DEPARTMENT IS &DEPARTMENT
-TYPE CURR_SAL   IS &CURR_SAL
-TYPE EMP_ID     IS &EMP_ID
```

The output is:

```
>   NUMBER OF RECORDS IN TABLE=       12  LINES=     12
 HOLDING BINARY FILE...
LAST_NAME  IS STEVENS
FIRST_NAME IS ALFRED
DEPARTMENT IS PRODUCTION
CURR_SAL   IS     11000.00
EMP_ID     IS 071382660
```

Syntax: How to Close an External File

The -CLOSE command closes an external file opened with the -READ or -WRITE command. The NOCLOSE option keeps a file open even when -RUN is encountered.

```
-CLOSE {ddname|*}
```

where:

ddname

Is the ddname of the open file described to FOCUS via an allocation.

*

Closes all -READ and -WRITE files that are currently open.

Reading or Writing an Entire File

Using the EDAGET and EDAPUT commands, you can read or write an entire file of a specified type.

EDAGET: Reading a File of a Specified Type

Using the EDAGET command, you can retrieve and display an entire file.

The -READ command only reads one line at a time, so this is a way to avoid issuing repeated commands for reading a single file.

Syntax: How to Read a File of a Specified Type

```
EX EDAGET filetype,[app/]filename,content-type
```

where:

filetype

Is the type of file. On z/OS, this is a DDNAME. The following table lists some of the most common FOCUS file types. Other FOCUS-readable files use the standard extensions, and the file types match the extensions:

File Type	Extension
MASTER	.mas
ACCESS	.acx
FOCEXEC	.fex
FOCSQL	.acx
FOCSTYLE	.sty
DATA	.dat
FOCCOMP	.fcm
FOCTEMP	.ftm

File Type	Extension
FOCUS	.foc
HOLDMAST	.mas
HOLDACC	.acx
HTML	.htm
EXCEL	.xls
MAINTAIN	.mnt
FOCPSB	.psb
TTEDIT	.tte

app

Is an optional application name where the file resides. On z/OS, you can specify an app if you have enabled application logic in the EDASERVE configuration file and created the data sets associated with the application.

filename

Is the file name.

content-type

Is the type of data in the file. Valid values are:

- **T**, which means a text file.
- **B**, which means a binary file.

Note: EDAGET cannot retrieve a file that was written to memory by EDAPUT.

Example: **Reading a File Using EDAGET**

The following EDAGET command reads a Master File named tempmas.mas in the app1 application. The content type is text:

```
EX EDAget MASTER,app1/tempmast,T
```

Using the tempmas.mas file created in *EDAPUT: Writing a File of a Specified Type* on page 265, the output is:

```
FILENAME=TEMPMAST, SUFFIX=FIX,$ SEGNAME=ONE, SEGTYPE=S1 ,$
FIELD=FIELD1 ,ALIAS= ,A10 ,A10 ,$ FIELD=FIELD2 ,ALIAS= ,P18 ,A18 ,$
```

EDAPUT: Writing a File of a Specified Type

Using the EDAPUT command, you can write any number of lines and save them as common FOCUS file types, either in memory or on disk.

The -WRITE command only writes one line at a time, so this is a way to avoid issuing repeated commands for writing a single file.

Syntax: How to Write a File of a Specified Type

```
EX -LINES n EDAPUT filetype,[app/]filename,type,location
```

where:

n

Is the number of lines that will be written, including the EDAPUT line.

filetype

Is the type of file. You specify the file type and, on Windows and UNIX, the file is saved with the associated extension. On z/OS, this is a DDNAME and the file is stored in the PDS associated with the DDNAME. The following table lists some of the most common FOCUS file types. Other FOCUS-readable files use the standard extensions, and the file types match the extensions:

File Type	Extension
MASTER	.mas
ACCESS	.acx
FOCEXEC	.fex
FOCSQL	.acx
FOCSTYLE	.sty
DATA	.dat

File Type	Extension
FOCCOMP	.fcm
FOCTEMP	.ftm
FOCUS	.foc
HOLDMAST	.mas
HOLDACC	.acx
HTML	.htm
EXCEL	.xls
MAINTAIN	.mnt
FOCPSB	.psb
WINFORMS	.wfm
TTEDIT	.tte

app

Is an optional application name under which to store the file. On z/OS, you can specify an app if you have enabled application logic in the EDASERVE configuration file and created the data sets associated with the application.

filename

Is the file name.

type

Is the creation type. Valid values are:

- **CV**, create variable.
- **C**, create fixed
- **A**, append to file.

location

Is the location for the created file. Valid values are:

- **FILE**, write to current location (this will overwrite an existing file of the same name and extension in the same location).

- **MEM**, write to memory only (this will not overwrite an existing file on disk). Files written to memory are first in the path.

Example: Writing a Master File to Disk

The following EDAPUT command writes a Master File named tempmast.mas to the app1 application directory in variable format. On z/OS, it writes member TEMPMAST to the APP1.MASTER.DATA data set under the high-level qualifier assigned as approot in the EDASERVE configuration file:

```
EX -LINES 5 EDAPUT MASTER,app1/tempmast,CV,FILE,
FILENAME=TEMPMAST, SUFFIX=FIX,$
SEGNAME=ONE, SEGTYPE=S1 ,$
  FIELD=FIELD1  ,ALIAS=  ,A10 ,A10 ,$
  FIELD=FIELD2  ,ALIAS=  ,P18 ,A18 ,$
```

Supplying Variable Values on the Command Line

When a user knows the values required by a procedure, some or all of the values can be typed on the command line using the EXEC command following the name of the procedure. This saves time since FOCUS now has values to pass to each local or global variable so the user is not prompted to supply them.

Syntax: How to Supply a Variable Value on the Command Line

```
EX[EC] procedure [[&&][variable=]value, ...]
```

where:

procedure

Is the name of the procedure that contains the name/value values.

variable

Is the name of the variable for which you are supplying a value. Omit for a positional variable.

For a local variable, do not include the ampersand in the variable name.

For a global amper variable, you must supply the double ampersand in the variable name:

```
EX SLRPT &&GLOBAL=value, CITY = STAMFORD, CODE1=B10, CODE2=B20
```

value

Is the value you are giving to the variable.

Name/value pairs must be separated by commas.

When the list of values to be supplied exceeds the width of the terminal, insert a comma as the last character on the line and enter the balance of the list on the following line(s), as shown:

```
EX SLRPT AREA=S, CITY = STAMFORD, VERB=COUNT, FIELDS = UNIT_SOLD,
CODE1=B10, CODE2=B20
```

Reference: Rules for Using Named and Positional Variables With EXEC

You can mix named and positional variables freely in the EXEC command. Positional variables are unnamed values passed to a procedure when it is invoked.

Follow these rules:

- Names must be associated with values for named variables.

  It is not necessary to enter the name=value pairs in the order encountered in the procedure.

- Values for positional variables must be supplied in the order that those variables are numbered within the procedure.

  If the variable is positional (it is a numbered variable), you do not need to specify the variable name in the EXEC command. FOCUS matches the EXEC values to the positional variables as they are encountered in the procedure. For an example, see *Using Positional Variables*.

Example: Supplying Values on the Command Line

Consider the following procedure named SLRPT:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &CITY"
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY PROD_CODE
IF PROD_CODE IS-FROM &CODE1 TO &CODE2
BY CITY
IF CITY EQ &CITY
END
```

You can supply values for the variables as parameters using the EX command as follows:

```
EX SLRPT CITY=STAMFORD, CODE1=B10, CODE2=B20
```

Example: Using Positional Variables

Consider the following example:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &1"
SUM UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY PROD_CODE
IF PROD_CODE IS-FROM &2 TO &3
BY CITY
IF CITY EQ &1
END
```

The EX command that calls the procedure is as follows:

```
EX SLRPT STAMFORD, B10, B20
```

This command substitutes STAMFORD for the first positional variable, B10 for the second, and B20 for the third.

Example: Mixing Named and Positional Variables

The report request SLRPT includes named and positional variables:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &CITY"
&VERB UNIT_SOLD AND RETURNS AND COMPUTE
RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
BY PROD_CODE
IF PROD_CODE IS-FROM &1 TO &2
BY CITY
IF CITY EQ &CITY
END
```

The following EX command executes SLRPT and populates the named and positional variables:

```
EX SLRPT CITY=STAMFORD, B10, B20, VERB=COUNT
```

&CITY is a named variable whose value is STAMFORD.

&1 is a positional variable whose value is B10.

&2 is a positional variable whose value is B20.

&VERB is a named variable whose value is COUNT.

Prompting Directly for Values With -PROMPT

The Dialogue Manager command -PROMPT solicits values before the variables to which they refer are used in the procedure. The user is prompted for a value as soon as -PROMPT is encountered. If a looping condition is present, -PROMPT requests a new value for the variable, even if a value exists already. Thus, each time through the loop, the user is prompted for a new value.

With -PROMPT you can specify format, text, and lists in the same way as all other variables.

Example: Prompting for Variable Values

The following is an example of the use of -PROMPT:

```
-PROMPT &CODE1
-PROMPT &CODE2
-SET &CITY = IF &CODE1 GT B09 THEN STAMFORD ELSE UNION;
-TYPE REGIONAL MANAGER FOR &CITY
-PROMPT &REGIONMGR
    TABLE FILE SALES
    HEADING CENTER
    "MONTHLY REPORT FOR &CITY"
    "PRODUCT CODES FROM &CODE1 TO &CODE2"
    SUM UNIT_SOLD AND RETURNS AND COMPUTE
    RATIO/D5.2 = 100 * (RETURNS/UNIT_SOLD);
    BY CITY
    IF CITY EQ &CITY
    BY PROD_CODE
    IF PROD_CODE IS-FROM &CODE1 TO &CODE2
    FOOTING CENTER
    "REGION MANAGER: &REGIONMGR"
    "CALCULATED AS OF &DATE"
    END
```

-PROMPT sends the following prompts to the screen. User input is shown in lowercase:

```
PLEASE SUPPLY VALUES REQUESTED

CODE1=  > b10
CODE2=  > b20
REGIONAL MANAGER FOR STAMFORD
REGIONMGR=  > smith
```

Note how the sequence of supplied values determines the overall flow of the procedure. The value of &CODE1 determines the value of &CITY that gives meaning to the -TYPE command. -TYPE gives the user the necessary information to make the correct choice when supplying the value for &REGIONMGR.

By default, all user input is automatically converted to uppercase.

Prompting for Values on Screens With -CRTFORM

-CRTFORM sets up full-screen menus for entering values. The -CRTFORM command in Dialogue Manager and the CRTFORM command in MODIFY are two versions of FIDEL for use in different contexts. The syntax, functions and features are fully outlined in the *Maintaining Databases* manual.

Prompting for Values on Menus and Windows With -WINDOW

You can create a series of menus and windows using Window Painter, and then display those menus and windows on the screen using the -WINDOW command. When displayed, the menus and windows collect data by prompting a user to select a value, to enter a value, or to press a program function (PF) key For details, see *Designing Windows With Window Painter* on page 449.

Prompting for Values Implicitly

If a value for a variable is not supplied by any other means, FOCUS automatically prompts the user for the value. This is known as an implicit prompt. These prompts occur sequentially as each variable is encountered in the procedure.

Example: Automatically Prompting for Variable Values

Consider the following example:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &CITY"
.
.
.
BY PROD_CODE
IF PROD_CODE IS-FROM &CODE1 TO &CODE2
.
.
.
FOOTING CENTER
"REGION MANAGER: &REGIONMGR"
"CALCULATED AS OF &DATE"
END
```

When you execute the procedure, FOCUS prompts for the values for the variables one at a time. The terminal dialogue is as follows. User input is in lowercase:

```
PLEASE SUPPLY VALUES REQUESTED

CODE1=  > b10
CODE2=  > b20
REGIONAL MANAGER FOR STAMFORD
REGIONMGR=  > smith
```

At the point when all variables have values, FOCUS processed the report request.

Verifying User-Supplied Values Against a Set of Format Specifications

You can specify variables with format conditions against which entered values can be compared. If the entered values do not have the specified format, FOCUS prints error messages and prompts the user again for the value(s).

Reference: Format Specifications for Variables

Alphanumeric formats are described by the letter A followed by the number of characters. The number of characters can be from 1 to 3968.

Numeric formats are described by the letter I, followed by the number of digits to be entered. The number of digits can be from 1 to 10 (value must be less than 2^{31}-1), and the value supplied for the number can contain a decimal point.

The description of the format must be enclosed by periods.

If you test field names against input variable values, specify formats of the input variables. If you do not, and the supplied value exceeds the format specification from the Master File, the procedure is ended and error messages are displayed. To continue, the procedure must be executed again. However, if you do include the format, and the supplied value exceeds the format, Dialogue Manager rejects the value and the user is prompted again.

Note: FOCUS internally stores all Dialogue Manager variables as alphanumeric codes. To perform arithmetic operations, Dialogue Manager converts the variable value to double-precision floating point decimal and then converts the result back to alphanumeric codes, dropping the decimal places. For this reason, do not perform tests that look for the decimal places in the numeric codes.

Example: Using a Format Specification to Verify User Input

Consider the following format specification:

```
&STORECODE.A3.
```

No special message is sent to the screen detailing the specified format. However, if in the above example the user enters more than three alphanumeric characters, the value is rejected, the error message FOC291 is displayed and the user is prompted again.

Note the following example detailing the dialogue between FOCUS and the user:

```
PLEASE SUPPLY VALUES REQUESTED

STORECODE=  > cc14
(FOC291) THE VALUE IN THE PROMPT REPLY EXCEEDS THE MAXIMUM LENGTH: 03
CHARS:CC14
STORECODE=
```

Verifying User Input Against a Pre-Defined List of Values

You can define values that constitute acceptable responses to prompts. If the user does not enter one of the available options, the terminal displays the list and re-prompts the user. This is an excellent way to limit the values supplied and to provide help information to the screen while prompting.

In addition, you can supply text that either explains what type of value is needed or lists choices of acceptable values on the screen.

Example: Providing a List of Valid Values With -PROMPT

The following lists acceptable responses for &CITY:

```
-PROMPT &CITY.(STAMFORD,UNIONDALE,NEWARK).
```

A message is printed if the user does not respond with one of the values on the list. This is followed by a display of the values list. Then, another prompt is issued for the needed value. For example:

```
PLEASE SUPPLY VALUES REQUESTED

CITY=  > union
PLEASE CHOOSE ONE OF THE FOLLOWING:
 STAMFORD,UNIONDALE,NEWARK
CITY=  >
```

Syntax: How to Create a Reply List as a Variable

You can provide a reply list as a variable, then prompt for the values you have defined for that variable. The syntax is

```
-SET &list='value,...';
-PROMPT &variable.(&list)[.text.]
```

where:

list

Is the name of the reply list variable. Note that in the -PROMPT command, the value is substituted between the parentheses and delimited by periods. If the prompt text has parentheses, enclose that text in single quotation marks (').

value

Is the desired value. You may list more than one value, separated by commas. Enclose the value(s) in single quotation marks ('). A semicolon is required when using -SET.

variable

Is the name of the variable for which you are prompting the user for values.

.text

Optionally provides prompting text.

Example: Using a Variable to Provide a Reply List

In this example, three acceptable values are defined for &CITY:

```
-SET &CITIES='STAMFORD,UNIONDALE,NEWARK';
-PROMPT &CITY.(&CITIES).'(ENTER CITY)'.
```

The resulting screen is exactly the same as when the list itself is provided in the parentheses. See *Providing a List of Valid Values With -PROMPT*.

You can also create more complex combinations. For example:

```
-SET &CITIES=IF &CODE1 IS B10 THEN 'STAMFORD, NEWARK'
- ELSE 'STAMFORD, UNIONDALE, NEWARK';
```

Example: Supplying Text for Variable Prompting

This example uses customized text to prompt for a values for &CITY, &CODE1, &CODE2, and &REGIONMGR:

```
TABLE FILE SALES
HEADING CENTER
"MONTHLY REPORT FOR &CITY.ENTER CITY. "
    .
    .
    .
BY PROD_CODE
IF PROD_CODE IS-FROM &CODE1.A3.BEGINNING CODE. TO
       &CODE2.A3.ENDING CODE.
    .
    .
    .
"REGION MANAGER: &REGIONMGR.REGIONAL SUPERVISOR."
"CALCULATED AS OF &DATEMDYY"
END
```

Notice that text has been specified for &CITY and &REGIONMGR without specification of a format.

Based on the example, the terminal displays the following prompts one by one:

```
ENTER CITY
stamford
BEGINNING CODE
b10
ENDING CODE
b20
REGIONAL SUPERVISOR
smith
```

Manipulating and Testing Variables

You can use a variety of techniques to manipulate and test Dialogue Manager variables.

- You can screen a value by adding a suffix to the variable value:
  - The .LENGTH suffix tests the length of a value.
  - The .TYPE suffix tests the type of a value.
  - The .EXIST suffix tests the presence of a value.
  - The .EVAL suffix replaces a variable with its value.
- You can use the -SET command alone or in conjunction with other commands and functions to manipulate the values for variables in order to:
  - Concatenate variables and/or literals. See *Concatenating Variables* on page 283.
  - Create an index for variables. See *Creating an Indexed Variable* on page 284.

- Perform calculations on a variable. See *Performing a Calculation on a Variable* on page 289.
- Change variable values. See *Changing a Variable Value With the DECODE Function* on page 289.
- Extract and insert characters. See *Extracting Characters From a Variable Value With the EDIT Function* on page 290.
- Remove trailing blanks. See *Removing Trailing Blanks From Variables With the TRUNCATE Function* on page 291.
- Call other functions. See *Calling a Function* on page 293.

- You can determine the command structure of a procedure based on the value of a variable. See *Using Variables to Alter Commands* on page 295.

Testing Variables for Length, Type, and Existence

To ensure that a supplied value is valid and being used properly in a procedure, you can test it for presence, type, and length. For example, you would not want to perform a numerical computation on a variable for which alphanumeric data has been supplied.

Syntax: How to Screen a Variable Value for Length and TYPE

```
-IF &name{.LENGTH|TYPE} rest_of_expression GOTO label...;
```

where:

&name

Is a user-supplied variable.

`.LENGTH`

Tests for the length of a value. If a value is not present, a zero (0) is passed to the expression. Otherwise, the number of characters in the value is passed.

`.TYPE`

Tests for the type of a value. The letter N (numeric) is passed to the expression if the value can be interpreted as a number up to 2^{31}-1 and stored in four bytes as a floating point format. In Dialogue Manager, the result of an arithmetic operation with numeric fields is truncated to an integer after the whole result of an expression is calculated. If the value could not be interpreted as numeric, the letter A (alphanumeric) is passed to the expression. If the value is not defined, the letter U is passed to the expression.

rest_of_expression

Is the remainder of an expression that uses *&name* with the specified suffix.

`GOTO label`

Specifies a label to branch to.

Example: Testing for Variable Length

If the length of &OPTION is more than one character, control passes to the label -FORMAT, which informs the client application that only a single character is allowed.

```
-IF &OPTION.LENGTH GT 1 GOTO FORMAT ELSE
-GOTO PRODSALES;
      .
      .
      .
-PRODSALES
    TABLE FILE SALES
      .
      .
      .
    END
-EXIT
-FORMAT
-TYPE ONLY A SINGLE CHARACTER IS ALLOWED.
```

Example: Storing the Length of a Variable

The following example sets the variable &WORDLEN to the length of the string contained in the variable &WORD.

```
-PROMPT &WORD.ENTER WORD.
-SET &WORDLEN = &WORD.LENGTH;
```

You can use this technique when you want to use one variable to populate another.

Example: Testing for Variable Type

If &OPTION is not alphanumeric, control passes to the label -NOALPHA, which informs the client application that only alphanumeric characters are allowed.

```
-IF &OPTION.TYPE NE A GOTO NOALPHA ELSE
- GOTO PRODSALES;
       .
       .
       .
-PRODSALES
    TABLE FILE SALES
       .
       .
       .
    END
-EXIT
-NOALPHA
-TYPE ENTER A LETTER ONLY.
```

Syntax: How to Test for the Presence of a Variable Value

```
-IF &name.EXIST GOTO label...;
```

where:

&name

Is a user-supplied variable.

.EXIST

Tests for the presence of a value. If a value is not present, a zero (0) is passed to the expression. Otherwise, a non-zero value is passed.

GOTO *label*

Specifies a label to branch to.

Example: Testing for the Presence of a Variable

If no value is supplied, &OPTION.EXIST is equal to zero and control is passed to the label -CANTRUN. The procedure sends a message to the client application and then exits. If a value is supplied, control passes to the label -PRODSALES.

```
-IF &OPTION.EXIST GOTO PRODSALES ELSE GOTO CANTRUN;
      .
      .
      .
-PRODSALES
    TABLE FILE SALES
      .
      .
      .
```

```
    END
-EXIT
-CANTRUN
-TYPE TOTAL REPORT CAN'T BE RUN WITHOUT AN OPTION.
-EXIT
```

Replacing a Variable Immediately

The .EVAL operator enables you to replace a variable with its value immediately, making it possible to change a procedure dynamically. The .EVAL operator is particularly useful in modifying code at run time.

Reference: Usage Notes for .EVAL

A Dialogue Manager variable is a placeholder for a value that will be substituted at run time. In some situations, the value of the variable may not be resolved at the point where the command containing the variable is encountered, unless evaluation is forced by using the .EVAL operator. One example where .EVAL Is required is in a -IF statement, when the variable is embedded in a label (for example, GOTO AB&*label*.EVAL). The .EVAL operator is also required any time a variable is included within single quotation marks (').

Syntax: How to Replace a Variable Immediately

`[&]&`*`variable`*`.EVAL`

where:

`variable`

Is a local or global variable.

When the command procedure is executed, the expression is replaced with the value of the specified variable before any other action is performed. The command that contains this value is then re-evaluated.

Without the .EVAL operator, a variable cannot be used in place of some commands.

Example: Replacing a Variable Immediately

The following example illustrates how to use the .EVAL operator in a record selection expression. The numbers to the left apply to the notes that follow the procedure:

```
1. -SET &R='IF SALARY GT 100000';
2. -IF &Y EQ 'YES' THEN GOTO START;
3. -SET &R = '-*';
   -START
4.   TABLE FILE CENTHR
     SUM SALARY
     BY PLANT
5.   &R.EVAL
     END
```

The procedure executes as follows:

1. The procedure sets the value of &R to 'IF SALARY GT 100000'.
2. If &Y is YES, the procedure branches to the START label, bypassing the second -SET command.
3. If &Y is NO, the procedure continues to the second -SET command, which sets &R to '-*', which is a comment.

   The report request is stacked.
4. The procedure evaluates the value of &R. If the end user wanted a record selection test, the value of &R is 'IF SALARY GT 100000' and this line is stacked.
5. If the end user does not want a record selection test, the value of &R is '-*' and this line is ignored.

Example: Using .EVAL to Interpret a Variable

Without .EVAL, Dialogue Manager interprets a variable only once. Therefore, in the following example,

```
-SET &A='-TYPE';
&A HELLO
```

Dialogue Manager does not recognize that &A is the -TYPE command so it does not display the word HELLO and generates the error message:

```
UNKNOWN FOCUS COMMAND -TYPE
```

Appending the .EVAL operator to the &A variable enables Dialogue Manager to interpret the variable correctly. The code

```
-SET &A='-TYPE';
&A.EVAL HELLO
```

produces the following output:

```
HELLO
>>
```

Validating Variable Values Without Data File Access: REGEX

You can validate a parameter value without accessing the data by using the REGEX mask. The REGEX mask specifies a regular expression to be used as the validation string. A regular expression is a sequence of special characters and literal characters that you can combine to form a search pattern.

Many references for regular expressions exist on the web. For a basic summary, see the section *Summary of Regular Expressions* in Chapter 2, *Security*, of the *Server Administration* manual.

The following messages display in case of an error:

```
(FOC2909) INVALID REGULAR EXPRESSION:
(FOC2910) RESPONSE DOES NOT MATCH THE REGULAR EXPRESSION:
```

Syntax: How to Validate a Variable Value Using a REGEX Mask

```
&variable.(|VALIDATE=REGEX,REGEX='regexpression').
```

where:

`&variable`

Is the variable to validate.

regexpression

Is the regular expression that specifies the acceptable values.

Example: Using a REGEX Mask to Validate a Social Security Number

The following request validates a Social Security number in either *xxxxxxxxx* or *xxx-xx-xxxx* format:

```
-REPEAT NEXTFMT FOR &FMTCNT FROM 1 TO 2
-SET &EMPID1=DECODE &FMTCNT(1 '071382660' 2 '818-69-2173');
-SET &EMPID=IF
     &EMPID1.(|VALIDATE=REGEX,REGEX='^\d{3}\-?\d{2}\-?\d{4}$').Employee ID.
CONTAINS '-'
-       THEN EDIT(&EMPID1,'999$99$9999') ELSE &EMPID1;
TABLE FILE EMPLOYEE
HEADING
" "
"Testing EMPID = &EMPID1</
1"
PRINT EID CSAL
WHERE EID EQ '&EMPID.EVAL'
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
END
-RUN
-NEXTFMT
```

The output is

```
Testing EMPID = 071382660


EMP_ID CURR_SAL
071382660 $11,000.00

Testing EMPID = 818-69-2173


EMP_ID CURR_SAL
818692173 $27,062.00
```

Example: Using REGEX With an Incorrect Value

In the following request, the second value for &EMPID1 is invalid because it does not conform to the REGEX mask:

```
-REPEAT NEXTFMT FOR &FMTCNT FROM 1 TO 2
-SET &EMPID1=DECODE &FMTCNT(1 '071382660' 2 '818-69-2173');
-TYPE EMPID1 = &EMPID1
-SET &EMPID=&EMPID1.(|VALIDATE=REGEX,REGEX='^\d{3}\d{2}\d{4}$').Employee
ID.;
-TYPE EMPID = &EMPID
-NEXTFMT
```

The FOC2910 message in the output shows that the second value for &EMPID1 was rejected:

```
EMPID1 = 071382660
EMPID = 071382660
EMPID1 = 818-69-2173
 ERROR AT OR NEAR LINE      7  IN PROCEDURE __WCFEX FOCEXEC *
(FOC2910)  RESPONSE DOES NOT MATCH THE REGULAR EXPRESSION: 818-69-2173
 ERROR AT OR NEAR LINE      7  IN PROCEDURE __WCFEX FOCEXEC *
(FOC295) A VALUE IS MISSING FOR: &EMPID1
```

Example: Using REGEX With an Invalid Regular Expression

In the following request, the REGEX mask is not a valid regular expression:

```
-SET &EMPID1='071382660';
-SET &EMPID=&EMPID1.(|VALIDATE=REGEX,REGEX='^\d{3}\d{2)}\d{4}$').Employee
ID.;
```

The FOC2909 message in the output shows that the regular expression is not valid:

```
ERROR AT OR NEAR LINE      5  IN PROCEDURE __WCFEX FOCEXEC *
(FOC2909)  INVALID REGULAR EXPRESSION: ^\d{3}\d{2)}\d{4}$
 ERROR AT OR NEAR LINE      5  IN PROCEDURE __WCFEX FOCEXEC *
(FOC295) A VALUE IS MISSING FOR: &EMPID1
```

Concatenating Variables

You can append a variable to a character string or combine two or more variables and/or literals. See the *Creating Reports* manual for complete information on concatenation. When using variables, it is important to separate each variable from the concatenation symbol (||) with a space.

Syntax: How to Concatenate Variables

```
-SET &name3 = &name1 || &name2;
```

where:

&name3

Is the name of the concatenated variable.

&name1 || &name2

Are the variables, separated by a space and the concatenation symbol.

Note: The example shown uses strong concatenation, indicated by the || symbol. Strong concatenation moves any trailing blanks from *&name1* to the end of the result. Conversely, weak concatenation, indicated by the symbol |, preserves any trailing blanks in *&name1*.

Creating an Indexed Variable

You can append the value of one variable to the value of another variable, creating an *indexed variable*. This feature applies to both local and global variables.

If the indexed value is numeric, the effect is similar to that of an array in traditional computer programming languages. For example, if the value of index &K varies from 1 to 10, the variable &AMOUNT.&K refers to one of ten variables, from &AMOUNT1 to &AMOUNT10.

A numeric index can be used as a counter; it can be set, incremented, and tested in a procedure.

Syntax: How to Create an Indexed Variable

```
-SET &name.&index[.&index...] = expression;
```

where:

&name

Is a variable.

.&index

Is a numeric or alphanumeric variable whose value is appended to *&name*. The period is required.

When more than one index is used, all index values are concatenated and the string appends to the name of the variable.

For example, &V.&I.&J.&K is equivalent to &V1120 when &I=1, &J=12, and &K=0.

expression

Is a valid expression. For information on the kinds of expressions you can write, see the *Creating Reports* manual.

Example: Using an Indexed Variable in a Loop

An indexed variable can be used in a loop. The following example creates the equivalent of a DO loop used in traditional programming languages:

```
-SET &N = 0;
-LOOP
-SET &N = &N+1;
-IF &N GT 12 GOTO OUT;
-SET &MONTH.&N=&N;
-TYPE &MONTH.&N
-GOTO LOOP
-OUT
```

In this example, &MONTH is the indexed variable and &N is the index. The value of the index is supplied through the command -SET; the first -SET initializes the index to 0, and the second -SET increments the index each time the procedure goes through the loop.

If the value of an index is not defined prior to reference, a blank value is assumed. As a result, the name and value of the indexed variable do not change.

Indexed variables are included in the system limit of 1024, which includes variables reserved by FOCUS.

Creating a Standard Quote-Delimited String

Character strings must be enclosed in single quotation marks to be handled by most database engines. In addition, embedded single quotation marks are indicated by two contiguous single quotation marks. FOCUS, WebFOCUS, and iWay require quotes around variables containing delimiters, which include spaces and commas.

The QUOTEDSTRING suffix on a Dialogue Manager variable applies the following two conversions to the contents of the variable:

- Any single quotation mark embedded within a string is converted to two single quotation marks.
- Single quotation marks are added around the string.

Dialogue Manager commands differ in their ability to handle character strings that are not enclosed in single quotation marks and contain embedded blanks. An explicit or implied -PROMPT command can read such a string. The entire input string is then enclosed in single quotation marks when operated on by .QUOTEDSTRING.

Note: When using the -SET command to reference a character string, ensure the character string is enclosed in single quotes to prevent errors.

Syntax: How to Create a Standard Quote-Delimited Character String

```
&var.QUOTEDSTRING
```

where:

&var

Is a Dialogue Manager variable.

Example: Creating a Standard Quote-Delimited Character String

The following example shows the results of the QUOTEDSTRING suffix on input strings.

```
-SET &A = ABC;
-SET &B = 'ABC';
-SET &C = O'BRIEN;
-SET &D = 'O'BRIEN';
-SET &E = 'O''BRIEN';
-SET &F = O''BRIEN;
-SET &G = OBRIEN';
-TYPE  ORIGINAL = &A QUOTED = &A.QUOTEDSTRING
-TYPE  ORIGINAL = &B QUOTED = &B.QUOTEDSTRING
-TYPE  ORIGINAL = &C QUOTED = &C.QUOTEDSTRING
-TYPE  ORIGINAL = &D QUOTED = &D.QUOTEDSTRING
-TYPE  ORIGINAL = &E QUOTED = &E.QUOTEDSTRING
-TYPE  ORIGINAL = &F QUOTED = &F.QUOTEDSTRING
-TYPE  ORIGINAL = &G QUOTED = &G.QUOTEDSTRING
```

The output is:

```
ORIGINAL = ABC         QUOTED = 'ABC'
ORIGINAL = ABC         QUOTED = 'ABC'
ORIGINAL = O'BRIEN     QUOTED = 'O''BRIEN'
ORIGINAL = O'BRIEN     QUOTED = 'O''BRIEN'
ORIGINAL = O'BRIEN     QUOTED = 'O''BRIEN'
ORIGINAL = O''BRIEN    QUOTED = 'O''''BRIEN'
ORIGINAL = OBRIEN'     QUOTED = 'OBRIEN'''

ORIGINAL = ABC         QUOTED = 'ABC'
ORIGINAL = ABC         QUOTED = 'ABC'
ORIGINAL = O'BRIEN     QUOTED = 'O''BRIEN'
ORIGINAL = O'BRIEN     QUOTED = 'O''BRIEN'
ORIGINAL = O'BRIEN     QUOTED = 'O''BRIEN'
ORIGINAL = O''BRIEN    QUOTED = 'O''''BRIEN'
ORIGINAL = OBRIEN'     QUOTED = 'OBRIEN'''
```

Note: The -SET command will remove single quotes around a string. Notice in the example above that the result of -SET &B = 'ABC' was changed to ORIGINAL = ABC (as shown in the output), prior to the QUOTEDSTRING conversion.

Example: Converting User Input to a Standard Quote-Delimited Character String

The following -TYPE command accepts quoted or unquoted input and displays quoted output.

```
-TYPE THE QUOTED VALUE IS: &E.QUOTEDSTRING
```

The output is:

```
PLEASE SUPPLY VALUES REQUESTED
E=
O'BRIEN
THE QUOTED VALUE IS: 'O''BRIEN'
```

Example: **Using Quote-Delimited Strings With Relational Data Adapters**

The following procedure creates an Oracle table named SQLVID from the VIDEOTRK data source.

```
TABLE FILE VIDEOTRK
SUM CUSTID EXPDATE PHONE STREET CITY STATE ZIP
   TRANSDATE PRODCODE TRANSCODE QUANTITY TRANSTOT
BY LASTNAME BY FIRSTNAME
WHERE LASTNAME NE 'NON-MEMBER'
ON TABLE HOLD
END
-RUN
CREATE FILE SQLVID
-RUN
MODIFY FILE SQLVID
FIXFORM FROM HOLD
DATA ON HOLD
END
```

Consider the following SQL Translator request:

```
SET TRACEUSER = ON
SET TRACEON = STMTRACE//CLIENT
SQL
SELECT *
FROM SQLVID WHERE LASTNAME  = &1.QUOTEDSTRING;
END
```

When this request is executed, you must enter a last name, in this case O'BRIEN:

```
PLEASE SUPPLY VALUES REQUESTED
1=
O'BRIEN
```

In the generated SQL request, the character string used for the comparison is correctly enclosed in single quotation marks, and the embedded single quote is doubled:

```
SELECT SQLCOR01.CIN , SQLCOR01.LN , SQLCOR01.FN ,
SQLCOR01.EXDAT , SQLCOR01.TEL , SQLCOR01.STR , SQLCOR01.CITY ,
SQLCOR01.PROV , SQLCOR01.POSTAL_CODE , SQLCOR01.OUTDATE ,
SQLCOR01.PCOD , SQLCOR01.TCOD , SQLCOR01.NO , SQLCOR01.TTOT
FROM SQLVID SQLCOR01  WHERE SQLCOR01.LN  = 'O''BRIEN';
```

The output is:

```
CIN   LN              FN      ...
---   --              --      ...
5564  O'BRIEN         DONALD  ...
```

The following input variations are translated to the correct form of quoted string demonstrated in the trace.

```
'O'BRIEN'
'O''BRIEN'
```

Any other variation results in:

- A valid string that does not match the database value and does not return any rows. For example, O''''BRIEN becomes 'O'''''''BRIEN' in the WHERE predicate.
- An invalid string that produces one of the following messages:

  Error - Semi-colon or END expected

  Error - Missing or Misplaced quotes

  Error - (value entered) is not a valid column

  Error - Syntax error on line ... Unbalanced quotes

Strings without embedded single quotation marks can be entered without quotes or embedded in single quotation marks, either SMITH or 'SMITH'.

If you use &1 without the QUOTEDSTRING suffix in the request, acceptable input strings that retrieve O'Brien's record are:

```
'''O'''BRIEN'''
'''O''''BRIEN'''
```

Using &1 without the QUOTEDSTRING suffix, the acceptable form of a string without embedded single quotation marks is '''SMITH'''.

To make a string enclosed in single quotation marks acceptable without the QUOTEDSTRING suffix, use '&1' in the request. In this case, in order to retrieve O'Brien's record, you must enter the string that would have resulted from the QUOTEDSTRING suffix:

```
'O''''BRIEN'
```

To enter a string without embedded single quotation marks using '&1', you can either omit the surrounding single quotation marks or include them: SMITH or 'SMITH'.

Note: The form '&1.QUOTEDSTRING' is not supported.

Reference: Usage Notes for Quote-Delimited Character Strings

An unmatched single quotation mark at the beginning of a character string is treated as invalid input and generates the following message:

```
(FOC257) MISSING QUOTE MARKS:  value;
```

Performing a Calculation on a Variable

You can use -SET to define a value for a substituted variable based on the results of a logical or arithmetic expression or a combination.

Syntax: How to Perform a Calculation on a Variable

```
-SET &name = expression;
```

where:

&name

Is a user-supplied variable that has its value assigned with the expression.

expression

Is an expression following the rules outlined in the *Creating Reports* manual, but with limitations as defined in this topic. The semicolon after the expression is required to terminate the -SET command. For information about setting a precision for Dialogue Manager calculations, see *How to Specify Precision for Dialogue Manager Calculations* on page 253.

Example: Altering a Variable Value

The following example demonstrates the use of -SET to alter variable values based on tests.

```
-START
-TYPE RETAIL PRICE ABOVE OR BELOW $1.00 IN THIS REPORT?
-PROMPT &CHOICE.ENTER A OR B.
-SET &REL = IF &CHOICE EQ A THEN 'GT' ELSE 'LT';
   TABLE FILE SALES
   PRINT PROD_CODE UNIT_SOLD RETAIL_PRICE
   BY STORE_CODE BY DATE
   IF RETAIL_PRICE &REL 1.00
   END
```

In the example, the &CHOICE variable receives either A or B as the value supplied through -PROMPT. Assuming the user enters the letter A, -SET assigns the string value GT to &REL. Then, the value GT is passed to the &REL variable in the procedure, so that the expanded FOCUS command at execution time is:

```
IF RETAIL_PRICE GT 1.00
```

Changing a Variable Value With the DECODE Function

You can use the DECODE function to change a variable to an associated value.

Example: Changing the Value of a Variable

In this example the variable refers to a label:

```
1.    -PROMPT &SELECT. ENTER CHOICE (A,B,C,D,E).
2.    -SET &GO=DECODE &SELECT (A ONE B TWO C THREE
      - D FOUR E FIVE ELSE EXIT);
3.    -GOTO &GO
      -ONE
      .
      .
      .
      -TWO
      .
      .
      .
```

The example processes as follows:

1. -PROMPT prompts the user at the terminal for a value for the variable &SELECT. Assume the user enters A.
2. -SET defines the variable &GO in terms of the DECODE function. Depending on the value input for &SELECT, DECODE associates a substitution. In this case, ONE is substituted for A.
3. -GOTO &GO transfers control to the label -ONE.

In the example, &GO can be another procedure (see *Dialogue Manager Quick Reference* on page 307) that is executed, depending on the value that is decoded:

```
-TOP
-TYPE
-PROMPT &SELECT.ENTER 1, 2, 3, 4, 5, OR EXIT TO END.
-SET &GO=DECODE &SELECT (1 ONE 2 TWO 3 THREE
-   4 FOUR 5 FIVE ELSE EXIT);
-IF &GO IS EXIT GOTO EXIT;
EX &GO
-RUN
-GOTO TOP
-EXIT
```

For more information on DECODE, see the *Using Functions* manual.

Extracting Characters From a Variable Value With the EDIT Function

You can use the mask option of the EDIT function with amper variables. You can insert characters into an alphanumeric value, or extract certain characters from the value.

Example: Extracting a Character From a Variable

In this example, EDIT extracts a particular character, in this case the J, for comparison in order to branch to the appropriate label. Assume there are nested menus and the user must supply a number to branch to a particular menu. If the first character is a J, the branch is to the label JUMP that enables the user to jump in nested menus (the numbers refer to the explanation below):

```
1.  -TYPE CHOOSE 1 for Edit, 2 for Print, 3 for Math
1.  -TYPE TO JUMP LEVELS OF MENUS TYPE J1.3 ETC.
2.  -PROMPT &OPTION.A4.Please enter selection:.
3.  -SET &XYZ = EDIT(&OPTION, '9$$$');
4.  -IF &XYZ EQ J THEN GOTO JUMP;
        .
        .
        .
5.  -JUMP
        .
        .
        .
```

The example processes as follows:

1. -TYPE send messages to the screen explaining the options to the user.
2. -PROMPT asks the user to enter a value for the variable &OPTION. It can have as many as four characters.
3. -SET calculates the variable &XYZ, which is the &OPTION variable, using the mask option of EDIT. The first character is screened.
4. -IF determines the branch. If the variable &XYZ is equal to J, processing continues to the label JUMP. Otherwise, processing continues to the next command in the procedure.
5. -JUMP is a label. The coding that follows contains the necessary FOCUS commands to enable the user to jump to the various menus.

Removing Trailing Blanks From Variables With the TRUNCATE Function

The Dialogue Manager TRUNCATE function removes trailing blanks from Dialogue Manager amper variables and adjusts the length accordingly.

The Dialogue Manager TRUNCATE function has only one argument, the string or variable to be truncated. If you attempt to use the Dialogue Manager TRUNCATE function with more than one argument, the following error message is generated:

```
(FOC03665) Error loading external function 'TRUNCATE'
```

This function can only be used in Dialogue Manager commands that support function calls, such as -SET and -IF commands. It cannot be used in -TYPE or -CRTFORM commands or in arguments passed to stored procedures.

Note: A user-written function of the same name can exist without conflict.

Syntax: How to Remove Trailing Blanks From Variables

```
-SET &var2 = TRUNCATE(&var1);
```

where:

&var2

Is the Dialogue Manager variable to which the truncated string is returned. The length of this variable is the length of the original string or variable minus the trailing blanks. If the original string consisted of only blanks, a single blank, with a length of one is returned.

&var1

Is a Dialogue Manager variable or a literal string enclosed in single quotation marks. System variables and statistical variables are allowed as well as user-created local and global variables.

Example: Removing Trailing Blanks

The following example shows the result of truncating trailing blanks:

```
-SET &LONG = 'ABC   ' ;
-SET &RESULT =  TRUNCATE(&LONG);
-SET &LL = &LONG.LENGTH;
-SET &RL = &RESULT.LENGTH;
-TYPE LONG   =  &LONG  LENGTH =  &LL
-TYPE RESULT =  &RESULT LENGTH =  &RL
```

The output is:

```
LONG   =  ABC     LENGTH =  06
RESULT =  ABC LENGTH =  03
```

The following example shows the result of truncating a string that consists of all blanks:

```
-SET &LONG = '      ' ;
-SET &RESULT =  TRUNCATE(&LONG);
-SET &LL = &LONG.LENGTH;
-SET &RL = &RESULT.LENGTH;
-TYPE LONG   =  &LONG  LENGTH =  &LL
-TYPE RESULT =  &RESULT LENGTH =  &RL
```

The output is:

```
LONG   =          LENGTH =  06
RESULT =    LENGTH =  01
```

The following example uses the TRUNCATE function as an argument for EDIT:

```
-SET &LONG = 'ABC   ' ;
-SET &RESULT =  EDIT(TRUNCATE(&LONG)|'Z','9999');
-SET &LL = &LONG.LENGTH;
-SET &RL = &RESULT.LENGTH;
-TYPE LONG   =  &LONG  LENGTH =  &LL
-TYPE RESULT =  &RESULT LENGTH =  &RL
```

The output is:

```
LONG   =  ABC     LENGTH =  06
RESULT =  ABCZ LENGTH =  04
```

Calling a Function

Any function name encountered in a Dialogue Manager expression that is not recognized as a system standard name or FOCUS function is assumed to be a function. These functions are externally programmed by users and stored in a library that is available at the time referenced. One or more arguments are passed to the user program, which performs an operation or calculation and returns a single value or character string.

Dialogue Manager variables can receive the values from functions through the -SET command.

Syntax: How to Set a Variable Value Based on the Result From a Function

```
-SET &name = routine(argument,...,'format');
```

where:

name

Is the name of the variable in which the result is stored.

routine

Is the name of the function.

argument

Represents the argument(s) that must be passed to the function. Numeric arguments are converted to double-precision (D) format.

format

Is the predefined format of the result. This is used to convert numeric results back to character representation. It must be enclosed in single quotation marks.

Example: Setting a Variable Value Based on the Result From a Function

In the following example, FOCUS invokes the function RATE, adds 0.5 to the calculated value, and then formats the result as a double precision number. This result is then stored in the variable &COST:

```
-PROMPT &COMPANY.WHAT COMPANY ARE YOU USING?.
-PROMPT &DEST.WHERE ARE YOU SENDING THE PACKAGE TO?.
-PROMPT &WEIGHT.HOW HEAVY IS THE PACKAGE IN POUNDS?.
-SET &COST = RATE(&COMPANY,&DEST,&WEIGHT,'D6.2') + 0.5;
-TYPE THE COST TO SEND A &WEIGHT pound PACKAGE
-TYPE TO &DEST BY &COMPANY IS &COST
```

Syntax: How to Load and Execute a Function With -TSO/-MVS RUN

These Dialogue Manager commands cause a function to be loaded and executed.

The commands provide an alternative to -SET, which is generally the preferred method for calling user-supplied functions (see *How to Set a Variable Value Based on the Result From a Function* on page 293).

However,-TSO/-MVS RUN must be used for this purpose when the function being called:

- Does not have arguments.
- Has no return argument.
- Does not accept numeric arguments in double precision format. In this case it is the user's responsibility to do the appropriate conversion.

The syntax is

```
{-TSO|-MVS} RUN routine[, argument,...]
```

where:

routine

Is the name of the function.

argument

Represents the argument(s) being passed to the function. Arguments that are variables must have sizes predefined in prior -SET commands.

If you use this syntax, please note the following:

- If the function returns a value that is not alphanumeric, Dialogue Manager is not able to display or interpret the value correctly.
- You must convert all numeric arguments to double precision before they are passed to the function. (You can use the ATODBL function to convert them.) However, if any portion of the double precision number can be interpreted as an EBCDIC comma, Dialogue Manager incorrectly interprets this argument as two arguments.
- A user-written function may employ an argument for both input and output purposes. It is the responsibility of the user program to move the correct number of characters into the output variables.

Example: Loading and Executing a Function

In this example, the function is CODENAME. The arguments that are variables are either prompted for or set at the beginning of the procedure and values are then supplied for the arguments.

```
-PROMPT &MYCODE.A3.
-SET &MYNAME = '';
-SET &MYFACTOR = '' ;
-TSO RUN CODENAME,&MYCODE,&MYNAME,&MYFACTOR
```

Using Variables to Alter Commands

A variable can refer to a FOCUS command or to a particular field. Therefore, the command structure of a procedure can be determined by the value of the variable.

Example: Using a Variable to Control What the TABLE Command Prints

In this example, the variable &FIELD determines the field to print in the TABLE request.

In the file named SALES, the variable &FIELD can display the values RETURNS, DAMAGED, or UNIT_SOLD.

```
TABLE FILE SALES
 .
 .
 .
PRINT &FIELD
BY PROD_CODE
 .
 .
 .
```

Using Numeric Amper Variables in Functions

FOCUS stores all amper variables as strings in alphanumeric format whether they contain alphanumeric or numeric data or a mixture of the two. There are only two data types available to amper variables: alphanumeric and numeric.

Determining Amper Variable Data Type

Data typing for amper variables is determined by the data content only. As a result, using quotation marks around a numeric value in a -SET command has no effect on the data type of the amper variable.

For example, the following request stores numeric data in variables &A, &B, and &C:

```
-SET &A=12345;
-SET &B='12345';
-SET &C=123.45
-TYPE &A &B &C
-TYPE &A.TYPE &B.TYPE &C.TYPE
```

The output shows that &A, &B, and &C all have the numeric data type:

```
12345 12345 123.45
N  N  N
```

Manipulating Amper Variables

When an amper variable is displayed, substituted, concatenated, or appended, there is no transformation of the value contained in the amper variable.

Substitution

```
-SET &C=123.45
IF RETAIL_COST EQ &C
```

becomes

```
IF RETAIL_COST EQ 123.45
```

Also, consider the following:

```
-SET &D= &C;
-TYPE &D &D.TYPE
```

The output shows that &D has the same value as &C and is also numeric:

```
123.45  N
```

Concatenation

The amper variable &F is created by concatenating &A and &C:

```
-SET &F =  &A | &C;
-TYPE &F &F.TYPE
```

The output shows that the value of &F is the value of &A followed by the value of &C, and that the type is numeric:

```
12345123.45  N
```

The following example creates the amper variable &E by embedding an ampersand in the string. The ampersand is not recognized as the start of a variable name and is treated as an alphanumeric symbol in a string:

```
-SET &E = 1234&C;
-TYPE &E &E.TYPE
```

The output shows that the variable is of type alphanumeric, not numeric. It is not the concatenation of the string '1234' with the variable &C:

```
1234&C  A
```

This same behavior can be produced with concatenation:

```
-SET &G = AT&|T;
-TYPE &G &G.TYPE
```

The output is:

```
AT&T  A
```

Using an Amper Variable in an Expression

When an amper variable is used in an expression, conversion may be required in order to process the expression. The amper variable used in the expression is generally seen as a literal, and its value is substituted in before the expression is processed. Under these circumstances, data conversion necessary to process the expression is performed. Numerics contained in amper variables are seen as integers. If the expression can be evaluated as integer, it will be.

In the following example, &C is set to 123.55. Then an expression creates &D by adding 100 to &C :

```
-SET &C=123.55;
-SET &D=&C + 100;
-TYPE &D &D.TYPE
```

The output shows that &D is numeric and its value is 123.55+100 truncated to the integer 223 because integer arithmetic is used:

```
223  N
```

The following expression requires conversion to double precision, as the numeric literal (100.49) in the expression is not an integer:

```
-SET &C=123.55;
-SET &D=&C + 100.49;
-TYPE &D &D.TYPE
```

The output shows that while the arithmetic was done by converting the value of &C to double precision, the result is truncated before being returned to &D:

```
224  N
```

If you want the result to retain the decimal places, you can set the DMPRECISION parameter to the number of decimal places you want returned to the resulting amper variable.

For example:

```
SET DMPRECISION=2
-RUN
-SET &C=123.55;
-SET &D=&C + 100.49;
```

Now the result retains the decimal places:

```
224.04 N
```

Using Amper Variables as Subroutine Parameters

How you treat numeric amper variables when passing them to a subroutine depends on the data type of the subroutine parameter.

Using a Numeric Amper Variable as a Numeric Subroutine Parameter

When using a numeric amper variable as a numeric parameter in a subroutine call, the amper variable is treated as a field. Since as a field, it has no specified type in either the Master File or the FOCEXEC, it takes the default data type of double precision.

Note that when the result is returned to an output variable, its type is determined by its content. If it has only numbers and a decimal point, it is numeric. If it contains other symbols, it is alphanumeric.

For example, the FTOA subroutine converts a double or single precision number (D or F) to an alphanumeric string with the format specified within the parentheses of the second parameter:

```
FTOA (number_to_convert, '(format)', 'alpha_output_format' )
```

The following example sets &C to 123.55 and passes it to the FTOA subroutine to be converted to an alphanumeric string with a dollar sign:

```
-SET &C=123.55;
-SET &G=FTOA(&C,'(D7.2M)','A11');
-TYPE &G &G.TYPE
```

The output shows that the string $123.55 has been returned to &G. Since it has a symbol other than numeric digits and a decimal point, its type is alphanumeric:

```
$123.55    A
```

In the following example, the format returned does not specify a dollar sign:

```
-SET &A=12345;
-SET  &G=FTOA(&A/100,'(D7.2)','A11');
-TYPE &G &G.TYPE
```

Since the returned string contains only numeric digits and a decimal point, its type is numeric:

```
123.45    N
```

Note that if the number had another digit, it would be returned with a comma, and its type would be alphanumeric:

```
-SET &A=123456;
-SET  &G=FTOA(&A/100,'(D7.2)','A11');
-TYPE &G &G.TYPE
```

The output is:

```
1,234.56    A
```

Using a Numeric Amper Variable as an Alphanumeric Subroutine Parameter

When using a numeric amper variable as an alphanumeric parameter in a subroutine call, you must convert the numeric value to an alphanumeric string before using it in order to avoid failure due to a format error. You can do this using one of the subroutines designed to convert numerics to alphanumeric, or you can concatenate an alphanumeric character to the numeric value in order to assign it an alphanumeric data type.

For example, the following converts &C to a string and returns the string to the variable &G. It then passes &G to the RJUST subroutine, which right justifies the value and returns it to the variable &H:

```
-SET &C=123.55;
-SET  &G=FTOA(&C,'(D7.2M)','A11');
-SET &H = RJUST(11,&G,'A11');
-TYPE &G &G.TYPE
-TYPE &H &H.TYPE
```

The output is:

```
$123.55   A
  $123.55 A
```

Debugging a Procedure

You can test and debug your procedure with the following.

- The &ECHO variable controls the display of command lines as they execute so you can test and debug procedure.
- The &STACK variable enables you to test the logic of Dialogue Manager commands. Setting this variable to OFF lets you run the procedure while preventing the execution of stacked (non-Dialogue Manager) commands. This gives you the ability to view the sequence of commands and see how the variable values are resolved.
- The &RETCODE variable returns a code after a procedure is executed. If the procedure results in normal output or no records are retrieved, the value of &RETCODE is 1. If an error occurs while parsing the procedure, the value of &RETCODE is 8.

  &RETCODE can be used to test the result of an operating system command. This retrieves the return code from the operating system.
- The &IORETURN variable tests the result of Dialogue Manager -READ and -WRITE commands. After a -READ or -WRITE operation, a non-zero return code indicates an error such as end-of-file being reached.

  &IORETURN can be used to test the result of the following:

  - A -READ command. If &IORETURN equals zero, a value was successfully read from the external file.
  - A -WRITE command. If &IORETURN equals zero, a value was successfully written to the external file.

Syntax: How to Display Command Lines as They Execute

```
{-DEFAULT|-SET|EX} &ECHO = {ON|ALL|OFF|NONE}
```

where:

`ON`

Displays FOCUS commands that are expanded and stacked for execution.

`ALL`

Displays Dialogue Manager commands and FOCUS commands that are expanded and stacked for execution.

`OFF`

Suppresses the display of both stacked commands and Dialogue Manager commands. This value is the default.

`NONE`

Prevents procedure code from being displayed (echoed). Once the value of &ECHO has been set to NONE, it cannot be changed during the session or connection.

By default, any procedure that does not explicitly set the &ECHO variable executes with the value OFF. You can change this default value for &ECHO with the SET DEFECHO command, as described in *How to Establish a Default Value for the &ECHO Variable* on page 301.

Syntax: How to Establish a Default Value for the &ECHO Variable

```
SET DEFECHO = {OFF|ON|ALL|NONE}
```

where:

`OFF`

Establishes OFF as the default value for &ECHO. OFF is the default value.

`ON`

Establishes ON as the default value for &ECHO. Displays FOCUS commands that are expanded and stacked for execution.

`ALL`

Establishes ALL as the default value for &ECHO. ALL displays Dialogue Manager commands and FOCUS commands that are expanded and stacked for execution.

NONE

Prevents procedure code from being displayed (echoed). Once the value of DEFECHO or &ECHO has been set to NONE, it cannot be changed during the session or connection.

Reference: Usage Notes for SET DEFECHO = NONE

- If you issue the SET DEFECHO=NONE command in a FOCEXEC, the setting does not affect &ECHO in that routine. It takes effect as the value of &ECHO in the next executed (EX) procedure after which it may not be changed.
- If you attempt to reset &ECHO within the duration of its NONE value, the value you attempted to set will display if you issue a -TYPE command, but the value will not actually change.

Example: Preventing Procedure Code From Being Displayed

The following procedure queries the value of the DEFECHO parameter and issues a TABLE request against the EMPLOYEE data source:

```
? SET DEFECHO
-RUN
-TYPE ECHO = &ECHO
TABLE FILE EMPLOYEE
PRINT CURR_SAL CURR_JOBCODE
BY LAST_NAME BY FIRST_NAME
END
-RUN
```

The query command output shows that DEFECHO is OFF (the default value):

```
 DEFECHO                  OFF
```

The -TYPE command shows that the value of &ECHO is OFF (the default):

```
ECHO = OFF
```

Because &ECHO is OFF the TABLE commands do not display as the procedure executes:

```
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
PAGE     1

LAST_NAME        FIRST_NAME         CURR_SAL  CURR_JOBCODE
---------        ----------         --------  ------------
BANNING          JOHN             $29,700.00  A17
BLACKWOOD        ROSEMARIE        $21,780.00  B04
CROSS            BARBARA          $27,062.00  A17
GREENSPAN        MARY              $9,000.00  A07
IRVING           JOAN             $26,862.00  A15
JONES            DIANE            $18,480.00  B03
MCCOY            JOHN             $18,480.00  B02
MCKNIGHT         ROGER            $16,100.00  B02
ROMANS           ANTHONY          $21,120.00  B04
SMITH            MARY             $13,200.00  B14
                 RICHARD           $9,500.00  A01
STEVENS          ALFRED           $11,000.00  A07




                                 END OF REPORT
```

Now, set DEFECHO=ON and re-run the procedure.

The query command output shows that DEFECHO is ON:

```
 DEFECHO                  ON
```

The -TYPE command shows that the value of &ECHO has been changed to ON:

```
ECHO = ON
```

Because &ECHO is ON, the TABLE commands display as the procedure executes:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL CURR_JOBCODE
BY LAST_NAME BY FIRST_NAME
END
```

The output displays next:

```
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
PAGE     1


LAST_NAME        FIRST_NAME         CURR_SAL  CURR_JOBCODE
---------        ----------         --------  ------------
BANNING          JOHN             $29,700.00  A17
BLACKWOOD        ROSEMARIE        $21,780.00  B04
CROSS            BARBARA          $27,062.00  A17
GREENSPAN        MARY              $9,000.00  A07
IRVING           JOAN             $26,862.00  A15
JONES            DIANE            $18,480.00  B03
MCCOY            JOHN             $18,480.00  B02
MCKNIGHT         ROGER            $16,100.00  B02
ROMANS           ANTHONY          $21,120.00  B04
SMITH            MARY             $13,200.00  B14
                 RICHARD           $9,500.00  A01
STEVENS          ALFRED           $11,000.00  A07




                                END OF REPORT
```

Now, issue the SET DEFECHO = NONE command and rerun the procedure:

```
SET DEFECHO = NONE
```

The query command output shows that the value of DEFECHO has been changed to NONE:

```
 DEFECHO                 NONE
```

The -TYPE command shows that the value of &ECHO is NONE:

```
ECHO = NONE
```

Because DEFECHO has the value NONE, the TABLE commands do not display as the procedure executes. The output is:

```
 NUMBER OF RECORDS IN TABLE=       12  LINES=     12

 PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY
PAGE     1


LAST_NAME        FIRST_NAME         CURR_SAL  CURR_JOBCODE
---------        ----------         --------  ------------
BANNING          JOHN             $29,700.00  A17
BLACKWOOD        ROSEMARIE        $21,780.00  B04
CROSS            BARBARA          $27,062.00  A17
GREENSPAN        MARY              $9,000.00  A07
IRVING           JOAN             $26,862.00  A15
JONES            DIANE            $18,480.00  B03
MCCOY            JOHN             $18,480.00  B02
MCKNIGHT         ROGER            $16,100.00  B02
ROMANS           ANTHONY          $21,120.00  B04
SMITH            MARY             $13,200.00  B14
                 RICHARD           $9,500.00  A01
STEVENS          ALFRED           $11,000.00  A07




                                END OF REPORT
```

Once the value of DEFECHO has been set to NONE, it cannot be changed. The following SET command attempts to change the value to ON, but the query command output shows that it is still NONE:

```
SET DEFECHO=ON
? SET DEFECHO
 DEFECHO                 NONE
```

Syntax: How to Test Dialogue Manager Command Logic

```
{-DEFAULT|-SET|EX procname} &STACK = {ON|OFF}
```

where:

procname

Is the procedure to execute.

ON

Executes stacked commands normally. This value is the default.

`OFF`

Prevents the execution of stacked commands. In addition, system variables (for example, &RECORDS or &LINES) are not set. Dialogue Manager commands are executed so you can test the logic of the procedure.

Note: &STACK is usually used with &ECHO = ALL for debugging purposes. The terminal displays both the Dialogue Manager commands, as well as the FOCUS commands with the supplied values. You can view the logic of the procedure.

Example: Using the &RETCODE Variable to Test the Result of a Command

If you are using Simultaneous Usage (SU), you must know if the FOCUS Database Server is available before beginning a particular procedure. The following procedure tests whether SINK1 is available before launching PROC1.

```
? SU SINK1
-RUN
-IF &RETCODE EQ 16 GOTO BAD;
-INCLUDE PROC1
-BAD
-EXIT
```

Reference: Testing the Status of a Query

The system variable &RETCODE returns a code after a query is executed. If the query results in a normal display, the value of &RETCODE is 0. If a display error occurs, or no display results (as can happen when the query finds no data), the value of &RETCODE is 8. (If the error occurs on a ? SU, the value of &RETCODE is 16.)

The value of &RETCODE is set following the execution of any of these queries:

	NORMAL	NODISPLAY	ERROR
`? HOLD`	0	8	
`? SU*`	0	8	16
`? JOIN`	0	8	
`? COMBINE`	0	8	
`? DEFINE`	0	8	

	NORMAL	NODISPLAY	ERROR
? USE	0	8	
? LOAD	0	8	

*The &RETCODE value of ? SU means: 0 indicates that the FOCUS Database Server (formerly called the sink machine) is up with one or more users; 8 indicates that the FOCUS Database Server is up with no users; 16 indicates that there is an error in communicating to the FOCUS Database Server.

You can test the status of any of these queries by checking the &RETCODE variable and providing branching instructions in your procedure.

Issuing an Operating System Command

You can issue an operating system command to set up an environment in which a request must run. For example, a program may allocate files, rename files, copy files, or perform other operations before executing a request.

Syntax: How to Execute an Operating System Command

```
op_system command
```

where:

op_system

Specifies the operating system.

-MVS specifies the z/OS operating system.

-TSO specifies the z/OS operating system.

command

Is an operating system command.

Dialogue Manager Quick Reference

This topic provides an alphabetical list of all Dialogue Manager commands, including a description of functions and syntax.

It also provides a grouped list of Dialogue Manager defaults and limits.

Note that this information is also presented throughout the chapter in the context of the task to which it applies.

-* Command

The command -* signals the beginning of a comment line.

Any number of comment lines can follow one another, but each must begin with -*. A comment line may be placed at the beginning or end of a procedure, or in between commands. However, it cannot be on the same line as a command.

Use comment lines liberally to document a procedure so that its purpose and history are clear to others.

The syntax is

```
-* text
```

where:

text

Is a comment. A space is not required between -* and text.

? Command

The command -? displays the current value of a local variable.

The syntax is

```
-? &[variablename]
```

where:

variablename

Is a variable name of up to 12 characters. If this parameter is not specified, the current values of all local, global, and defined system and statistical variables are displayed.

-CLOSE Command

-CLOSE closes an external file opened with the -READ or -WRITE NOCLOSE option. The NOCLOSE option keeps a file open until the -READ or -WRITE operation is complete.

The syntax is

```
-CLOSE {ddname|*}
```

where:

ddname

Is the ddname of the open file described to FOCUS via an allocation.

*

Closes all -READ and -WRITE files that are currently open.

-CRTCLEAR Command

-CRTCLEAR clears the current screen display.

The syntax is

```
-CRTCLEAR
```

-CRTFORM Command

-CRTFORM creates forms that prompt the user for values for variables.

All lines following a -CRTFORM command that begin with a hyphen and enclose text in double quotation marks (") are part of a single-screen form. Pressing ENTER passes all input data to associated variables.

With -CRTFORM, the first line that does not begin with a -" signals the end of the form. With -CRTFORM BEGIN, the command -CRTFORM END signals the end of the form.

All FIDEL facilities are available to -CRTFORM except HEIGHT, WIDTH, and LINE.

CRTFORM in MODIFY functions identically to -CRTFORM in Dialogue Manager.

For additional information, see *-PROMPT Command* on page 314.

The syntax is

```
-CRTFORM [TYPE n] [BEGIN|END [LOWER|UPPER]]
```

where:

-CRTFORM

Invokes FIDEL and signals the beginning of the screen form.

TYPE *n*

Enables you to define the number of lines (*n*) to reserve for messages. You can specify a number from 1 to 4. The default is 4.

`BEGIN`

Supports the use of other Dialogue Manager commands to help build the form.

`END`

Signals the end of the -CRTFORM. Used with -CRTFORM BEGIN.

`LOWER`

Reads lowercase data from the screen. Once you specify LOWER, every screen thereafter is a lowercase screen until you specify otherwise.

`UPPER`

Translates lowercase letters to uppercase. This is the default.

-DEFAULT[S|H] Command

DEFAULT commands set default values for local or global variables. -DEFAULT guarantees that the variables are always given a value and helps ensure that it executes correctly.

You can issue multiple -DEFAULT commands for a variable. If the variable is global, these -DEFAULT commands can be issued in separate FOCEXECs. At any point before another method is used to establish a value for the variable, the most recently issued -DEFAULT command will be in effect.

However, as soon as a value for the variable is established using any other method, subsequent -DEFAULT commands issued for that variable are ignored.

You can override -DEFAULT values by supplying values for the variables on the command line, by specifically prompting for values with -PROMPT or -CRTFORM, or by supplying a value with -SET subsequent to -DEFAULT.

Default values are provided in other FOCUS modules to anticipate user needs and reduce the need for keystrokes in situations where most users desire a predefined outcome. For additional information, see also *-SET Command* on page 319.

The syntax is

```
-DEFAULT[S|H] &[&]name=value [...]
```

where:

&name

Is the name of the variable.

value

Is the default value assigned to the variable.

-EXIT Command

-EXIT forces a procedure to end. All stacked commands are executed and the procedure exits. If the procedure was called by another one, the calling procedure continues processing.

Use -EXIT for terminating a procedure after processing a final branch that completes the desired task. The last line of a procedure is an implicit -EXIT.

The syntax is

```
-EXIT
```

-GOTO Command

-GOTO transfers control to a specified label.

If Dialogue Manager finds the label, processing continues with the line following it. If Dialogue Manager does not find the label, processing ends and an error message is displayed.

The syntax is

```
-GOTO label

    .
    .
    .
-label [TYPE text]
```

where:

label

The label in the *-label* command Is a user-defined name of up to 12 characters that specifies the target of the -GOTO action.

Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use words that can be confused with functions, arithmetic and logical operations, and so on.

TYPE *text*

Optionally sends a message to the client application.

-IF Command

-IF routes execution of a procedure based on the evaluation of the specified expression.

An -IF without an explicitly specified ELSE whose expression is false continues processing with the line immediately following it.

The syntax is

```
-IF expression [THEN] GOTO label1
[- ELSE GOTO label2]
[- ELSE IF...];
```

where:

label

Is a user-defined name of up to 12 characters that specifies the target of the GOTO action.

Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use words that can be confused with functions, arithmetic or logical operations, and so on.

expression

Is a valid expression. Literals need not be enclosed in single quotation marks unless they contain embedded blanks or commas.

THEN

Is an optional keyword that increases readability of the command.

ELSE GOTO

Passes control to label2 when the -IF test fails.

ELSE IF

Specifies a compound -IF test.

The semicolon is required at the end of the command, and continuation lines must begin with a hyphen.

-INCLUDE Command

-INCLUDE specifies another procedure to be incorporated and executed at run time, as if it were part of the calling procedure. The specified procedure may comprise either a fully developed or partial procedure. Note that a partial procedure does not execute if called outside of the procedure containing -INCLUDE.

When using -INCLUDE, you may not branch to a label outside of the specified procedure.

A procedure may contain more than one -INCLUDE. Any number of -INCLUDEs may be nested, but recursive -INCLUDEs are limited to four levels.

You may use any valid command in a -INCLUDE.

EXEC may also be used to execute a procedure inside another procedure.

The syntax is

```
-INCLUDE filename [filetype]
```

where:

filename

Is the procedure to be incorporated in the calling procedure.

filetype

Is the procedure's DDNAME. If none is included, FOCEXEC is assumed.

-label Command

The label specified in the *-label* command Is the target of a -GOTO command or -IF criteria.

The syntax is

```
-label [TYPE message]
```

where:

label

Is a user-supplied name of up to 12 characters that identifies the target for a branch.

Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use words that can be confused with functions, arithmetic or logical operations, and so on.

TYPE *message*

Sends a message to the client application.

-MVS Command

-MVS executes a z/OS command. -MVS is a synonym for -TSO. It is only supported with the RUN command.

-MVS RUN Command

This command is the same as -TSO RUN.

The syntax is

```
-MVS RUN
```

-PASS Command

-PASS directly issues and controls passwords. This feature is especially useful for specifying a particular file or set of files that a given user can read or write. Passwords have detailed sets of functions associated with them through DBA module.

The procedure that sets passwords should be encrypted so that it and the passwords that it sets cannot be typed and made known.

A variable can be associated with -PASS so that you can prompt for and assign a password value.

The PASS command provides the same function at the command level, as does the PASS parameter of the SET command.

The syntax is

```
-PASS password
```

where:

password

Is a literal FOCUS password or a variable containing a password.

-PROMPT Command

Types a message to the terminal and reads the reply from the user. This reply assigns a value to the variable named.

If a format is specified and the supplied value does not conform, FOCUS displays an error message and prompts the user again for the value.

If a (list) is specified and the user does not reply with a value on the list, FOCUS reprompts and prints the list of acceptable values.

Note: You cannot use format and list together.

In MODIFY, PROMPT specifies additional data input needs.

In GRAPH, when it is set on, GPROMPT automatically prompts for all parameters needed to execute the graph request. This is quite a different function from -PROMPT in Dialogue Manager.

For additional information, see *-CRTFORM Command* on page 309.

The syntax is

```
-PROMPT &name [[.format|.(list)] [.text].]
```

where:

&name

Is a user-defined variable.

format

Optionally specifies alphanumeric or integer data type and length.

text

Optionally specifies prompting text that appears on the screen. Must be delimited by periods.

list

Optionally specifies a range of acceptable responses. Must be enclosed in parentheses.

-QUIT Command

-QUIT forces an immediate exit from the procedure. Stacked lines are not executed. This differs from an -EXIT, which executes all lines that are currently on the stack.

Like -EXIT, -QUIT returns the user to the FOCUS prompt.

-QUIT FOCUS takes the user out of FOCUS altogether and returns the user to the operating system level.

-QUIT can be made the target of a branch, with the same results as those already described.

QUIT can be entered in response to -PROMPT or -CRTFORM to force an exit from the procedure. The QUIT command can, however, be turned off from within Dialogue Manager to prevent the user from exiting FOCUS prompt.

The QUIT command can also be used to exit from MODIFY and TABLE requests as well as Dialogue Manager procedures.

The principle of QUIT remains consistent throughout FOCUS, namely that the exited request or procedure is not executed and the user is returned to the FOCUS prompt.

For additional information, see also *-RUN Command* on page 318 and *-EXIT Command* on page 311.

The syntax is

```
-QUIT or -QUIT FOCUS [n]
```

where:

n

Is the operating system return code. It can be a constant or an integer variable up to 4095. If you do not supply a value or if you supply a non-integer value for *n*, the return code is 8 (the default value).

-READ Command

Reads data from an external (non-FOCUS) file. -READ can access data in either fixed or free form.

For additional information, see *-WRITE Command* on page 322.

The syntax is

```
-READ ddname[,] [NOCLOSE] &name[.format.][,] ...
```

where:

ddname

Is the logical name of the file as defined to FOCUS using ALLOCATE or DYNAM ALLOCATE. A space after the ddname denotes a fixed format file while a comma denotes a comma-delimited file.

NOCLOSE

Indicates that the ddname should be kept open even after a -RUN is executed. The ddname is closed upon completion of the procedure or when a -CLOSE or subsequent -WRITE command is encountered.

name

Is the variable name. You may specify more than one variable. Using a comma to separate variables is optional.

If the list of variables is longer than one line, end the first line with a comma and begin the next line with a dash followed by a blank (-) for comma-delimited files or a dash followed by a comma followed by a blank (-,) for fixed format files. For example:

Comma-delimited files

```
-READ EXTFILE, &CITY,&CODE1,- &CODE2
```

Fixed format files

```
-READ EXTFILE &CITY.A8. &CODE1.A3.,-, &CODE2.A3
```

format

Is the format of the variable. It may be Alphanumeric (A) or Integer (I). Note that format must be delimited by periods. The format is ignored for comma-delimited files.

-READFILE Command

-READFILE reads a Master File, then reads values from a file into variables based on the fields listed in the Master File.

```
-READFILE mastername
```

where:

mastername

Is the name of the Master File to be read.

-REMOTE Command

-REMOTE passes execution of the commands within a -REMOTE BEGIN and -REMOTE END command to a server.

For information, see the *Overview and Operating Environments* Manual.

The syntax is

```
-REMOTE BEGIN
commands
-REMOTE END
```

-REPEAT Command

-REPEAT allows looping in a procedure.

A loop ends when any of the following occurs:

- It is executed in its entirety.
- A -QUIT or -EXIT is issued.
- A -GOTO is issued to a label outside of the loop. If a -GOTO is later issued to return to the loop, the loop proceeds from the point it left off.

The syntax is

```
-REPEAT label n TIMES
-REPEAT label WHILE condition
-REPEAT label FOR &variable
     [FROM fromval] [TO toval] [STEP s]
```

where:

label

Identifies the code to be repeated (the loop). A label can include another loop if the label for the second loop has a different name from the first.

n TIMES

Specifies the number of times to execute the loop. The value of *n* can be a local variable, a global variable, or a constant. If it is a variable, it is evaluated only once, so you cannot change the number of times to execute the loop. The loop can only be ended early using -QUIT or -EXIT.

WHILE *condition*

Specifies the condition under which to execute the loop. The condition is any logical expression that can be true or false. The loop is run if the condition is true.

&variable

Is a variable that is tested at the start of each execution of the loop and incremented by *s* with each execution. It is compared with the value of *fromval* and *toval*, if supplied. The loop is executed only if *&variable* is greater than or equal to *fromval* or less than or equal to *toval*.

fromval

Is a constant that is compared with *&variable* at the start of the execution of the loop. The default value is 1.

toval

Is a value that is compared with *&variable* at the start of the execution of the loop. The default value is 1,000,000.

STEP *s*

Is a constant used to increment *&variable* at the end of the execution of the loop. It may be positive or negative. The default increment is 1.

Note: The parameters FROM, TO, and STEP can appear in any order.

-RUN Command

-RUN causes immediate execution of all stacked FOCUS commands.

Following execution, processing of the procedure continues with the line that follows -RUN.

-RUN is commonly used to do the following:

- Generate results from a request that can then be used in testing and branching.
- Close an external file opened with -READ or -WRITE. When a file is closed, the line pointer is placed at the beginning of the file for a -READ. The line pointer for -WRITE is positioned depending on the allocation and definition of the file.

The syntax is

```
-RUN
```

-SET Command

-SET assigns a literal value, or a value that is computed in an arithmetic or logical expression, to a variable.

Single quotation marks around a literal value are optional unless it contains an embedded blank, comma, or equal sign, in which case you must include them.

The syntax is

```
-SET &[&]name= {expression|value};
```

where:

&name

Is the name of the variable.

expression

Is a valid expression. Expressions can occupy several lines, so you should end the command with a semicolon.

value

Is a literal value, or arithmetic or logical expression assigned to the variable. If the literal value contains commas or embedded blanks, you must enclose the value in single quotation marks.

-TSO Command

-TSO executes a TSO operating system command from within Dialogue Manager. It is only supported with the RUN command.

The syntax is

```
-TSO command
```

where:

command

Is a TSO RUN command.

-TSO RUN Command

In TSO, loads and executes the specified user-written function.

Note that the preferred way to execute user-written programs is with the -SET command.

The syntax is

```
-TSO RUN function
```

where:

function

Is the name of a user-written function.

-TYPE Command

Transmits informative messages to the user at the terminal. Any number of -TYPE lines may follow one another but each must begin with -TYPE.

Substitutable variables may be embedded in text. The values currently assigned to each variable is displayed in the assigned position in the text.

-TYPE1 and TYPE+ are not supported by IBM 3270-type terminals.

TYPE is used in a variety of ways in FOCUS to send informative messages to the screen. A TYPE command may appear on the same line as a label in Dialogue Manager. In MODIFY, TYPE is used to print messages at the start and end of processes, at selected positions in MATCH or NOMATCH, NEXT or NONEXT, and to send a message after an INVALID data condition.

The syntax is

```
-TYPE[+] text
-TYPE[0] text
-TYPE[1] text
```

where:

-TYPE1

Sends the text after issuing a page eject.

-TYPE0

Sends the text after skipping a line.

`-TYPE+`

Sends the text but does not add a line feed.

`text`

Is a character string that fits on a line.

-WINDOW Command

-WINDOW executes a window file. When the command is encountered, control is transferred from the procedure to the specified window file. The window specified in the command becomes the first active window. Control remains within the window file until a menu option is chosen, or a window is activated, for which there is no goto value.

The window file, and the windows in it, are created using Window Painter.

The syntax is

```
-WINDOW windowfile windowname [PFKEY|NOPFKEY]
[GETHOLD][BLANK|NOBLANK][CLEAR|NOCLEAR]
```

where:

`windowfile`

Identifies the file in which the windows are stored. This is a member name. The member must belong to a PDS allocated to ddname FMU.

`windowname`

Identifies which window in the file is displayed first.

`PFKEY`

Enables you to test for function key values during window execution.

`NOPFKEY`

You are unable to test for function key values during window execution.

`GETHOLD`

Retrieves stored amper variables collected from a Multi-Select window.

`BLANK`

Clears all previously set amper variable values when -WINDOW is encountered. This is the default setting.

`NOBLANK`

When -WINDOW is encountered, the values of previously set amper variables are retained.

CLEAR

Clears the screen before displaying the first window. This is the default behavior. When specified in conjunction with the Terminal Operator Environment (TOE), the TOE screen is redisplayed when control is transferred back to the procedure.

NOCLEAR

Displays the specified window directly over the current screen.

-WRITE Command

-WRITE writes data to a sequential file.

If the command continues over several lines, put a comma at the end of the line and a hyphen at the beginning of each subsequent line.

Unless you specify the NOCLOSE option, an opened file is closed upon termination of the procedure with -RUN, -EXIT, or -QUIT.

In TABLE, WRITE is a synonym for SUM; functionally it is quite different from -WRITE.

For additional information, see *-READ Command* on page 316.

The syntax is

```
-WRITE ddname [NOCLOSE] text
```

where:

ddname

Is the logical name of the file as defined to FOCUS using ALLOCATE or DYNAM ALLOCATE.

NOCLOSE

Indicates that the file should be kept open even if a -RUN is encountered. The file is closed upon completion of the procedure or when a -CLOSE or subsequent -READ command is encountered.

text

Is any combination of variables and text. To write more than one line, end the first line with a comma (,) and begin the next line with a hyphen followed by a space (-).

-" " Command

The -" " syntax is associated with the FIDEL -CRTFORM command. All textual data enclosed by the double quotation marks is printed to the screen. You can use position markers and specify variable fields within double quotation marks.

When -CRTFORM is processed, the screen displays a form and the cursor stops at each amper variable date entry field. If a variable has not been declared prior to the -CRTFORM, FOCUS prompts the user for a value to assign to the variable.

In MODIFY, enclosing data in double quotation marks (" ") without the leading hyphen is used with CRTFORM, or for headings, footings, subheads, and subfoots within a TABLE request.

For additional information, see *-CRTFORM Command* on page 309.

The syntax is

```
-" "
```

where:

```
" "
```

Enclose textual information, fields and spot markers.

Dialogue Manager Defaults and Limits

This topic provides you with an easier way of locating default values, operating system and FOCUS limits, summary tables, general rules, and tips for ease-of-use.

General rules to follow when you are creating procedures are:

- If a Dialogue Manager command exceeds one line, the following line must begin with a hyphen (-).
- The hyphen (-) must be placed at the first position of the command line.
- The command is usually attached to the hyphen (-), but you may leave space between the hyphen and the Dialogue Manager command.
- At least one space must be inserted between the Dialogue Manager command and other text.

General rules for supplying values for variables:

- The lengths of values stored in Dialogue Manager (amper) variables vary by context:
  - When used with the commands -READ, -TYPE, and WRITE, the maximum length of a variable is approximately 32,000 characters (32K).
  - When used with other Dialogue Manager commands or the EX command, a variable value cannot exceed 4,096 character (4K).
- If a value contains an embedded comma (,) or embedded equal sign (=) the value must be enclosed between single quotation marks. For example:

```
EX SLRPT AREA=S, LOCATION='NY, NY'
```

- Once a value is supplied for a local variable, it is used for that variable throughout the procedure, unless it is changed through a -PROMPT, -SET, or -READ.
- Once a value is supplied for a global variable, it is used for that global variable throughout the FOCUS session in all procedures, unless it is changed through a -PROMPT, -SET, or -READ.
- Dialogue Manager automatically sends a prompt to the terminal if a value has not been supplied for a variable. Automatic prompts (implied prompting) are identical in syntax and function to the direct prompts created with -PROMPT.

Operating system default values, limits, and format specifications.

- The default value for the operating system return code value is 8.
- Literals must be surrounded by single quotation marks if they contain embedded blanks or commas. To produce a literal that starts or ends with a single quotation mark, place two single quotation marks where you want one to appear.
- Alphanumeric formats are described by the letter A followed by the number of characters. The number of characters can be from 1 to 3968.
- Integer formats are described by the letter I followed by the number of digits to be entered. The number can be from one to 10 digits in length, value must be less than 2^{31}-1.
- A label is a user-defined name of up to 12 characters. You cannot use blanks and should not use the name of any other Dialogue Manager command except QUIT and EXIT. The label may precede or follow GOTO in the procedure.
- A date supplied to Dialogue Manager cannot exceed 20 characters, including spaces.
- The level of nested -INCLUDE files is limited only by available memory. However, recursive -INCLUDE commands are limited to four levels.
- The default setting for &QUIT is ON.
- When using Window Painter:
  - Screens should not begin in row 0, column 0, or column 1.
  - The maximum screen size is 22 rows by 77 columns.
  - A File Contents window has a limit of 12K worth of data. This is approximately 150 lines.

- The maximum number of menu items is 41.
- File Name windows must have a WIDTH of 24 or greater, or meaningless characters will appear.

Chapter 4

Testing and Debugging With Query Commands

You can debug a FOCUS application by querying your environment to display information, such as release, FOCUS information, and joins, as well as by identifying files you are using.

In this chapter:

- Using Query Commands
- Displaying Combined Structures
- Displaying Virtual Fields
- Displaying the Currency Data Source in Effect
- Displaying Available Fields
- Displaying the File Directory Table
- Displaying Field Information for a Master File
- Displaying Data Source Statistics
- Displaying Defined Functions
- Displaying HOLD Fields
- Displaying JOIN Structures
- Displaying National Language Support
- Displaying LET Substitutions
- Displaying Information About Loaded Files
- Displaying Explanations of Error Messages
- Displaying PF Key Assignments
- Displaying the Release Number
- Displaying Parameter Settings
- Displaying Graph Parameters
- Displaying the Site Code
- Displaying Command Statistics
- Displaying StyleSheet Parameter Settings
- Displaying Information About the SU Machine
- Displaying Data Sources Specified With USE
- Displaying Global Variable Values
- Reporting Dynamically From System Tables

Using Query Commands

Query commands display information about your metadata, physical data sources, language environment, and development and run-time environment.

Syntax: How to Issue a Query Command

```
? query [filename]
```

where:

query

Is the subject of the query.

filename

Is the name of the file that is the subject of the query. This parameter applies to only some queries.

To list the query commands, type a question mark in a procedure or at the command prompt.

Reference: Query Command Summary

The following is a list of query commands. This topic contains a detailed description of each.

Query Command	Description
? COMBINE	Displays a list of combined file structures.
? DEFINE	Displays currently active virtual fields created by the DEFINE command or attribute.
?F	Lists fields currently available.
? FDT	Displays physical attributes of a FOCUS data source.
?FF	Lists field names, aliases, and format information for an active Master File.
? FILE	Displays the number of segment instances in a FOCUS data source and the last time the data sources was changed.
? FUNCTION	Displays functions created with the DEFINE command.

Query Command	Description
? HOLD	Displays fields described in a HOLD Master File.
? JOIN	Displays JOIN structures that exist between data sources.
? LANG	Displays information about National Language Support.
? LET	Displays word substitutions created with the LET command.
? LOAD	Provides information about all loaded files: the file type, file name, and resident size.
? MDI	Generates statistics and descriptions for multi-dimensional indexes.
? *n*	Displays an explanation of an error message (*n* represents the number of the error message).
? PFKEY	Displays the PF key assignments.
? RELEASE	Displays the release number of your product.
? SET	Displays parameter settings that control FOCUS.
? SET GRAPH	Displays parameter settings that control graphs produced with the GRAPH command.
? SET NOT	Produces a list of SET commands that cannot be set in a specific area.
? SITECODE	Retrieves the site code.
? STAT	Displays statistics about the last command executed.
? STYLE	Displays the current settings for StyleSheet parameters.
? SU	Is communication available to the SU machine.
? USE	Displays data sources specified with the USE command.

Query Command	Description
? &&	Displays values of global variables.

Displaying Combined Structures

The ? COMBINE command displays files that are in the current combined structures.

Syntax: How to Display Combined Structures

```
? COMBINE
```

Example: Displaying Combined Structures

Issuing the command

```
? COMBINE
```

produces information similar to the following:

```
COMBINE EDUCFILE AND JOBFILE AS EDJOB
>
? COMBINE
 FILE=EDJOB            TAG            PREFIX

   EDUCFILE
   JOBFILE
>
```

Displaying Virtual Fields

The? DEFINE command lists the active virtual fields used in a request. The fields can be created by either the DEFINE command or DEFINE attribute in the Master File. The command displays field names of up to 32 characters. If a name exceeds 32 characters, an ampersand (&) in the 32nd position indicates a longer field name.

Syntax: How to Display Virtual Fields

```
? DEFINE [appname/][filename]
```

where:

appname
Is the application directory.

filename

Is the data source containing the virtual fields. If *filename* is omitted, the command displays all virtual fields.

Example: **Displaying Virtual Fields**

Assume that you created a virtual field named FULLNAME in a request against the EMPLOYEE database.

Issuing

```
? DEFINE
```

produces the following information:

```
FILE        FIELD NAME              FORMAT    SEGMENT    VIEW    TYPE
EMPLOYEE    PROJECTEDSAL            D12.2
EMPLOYEE    FULLNAME                A26
>
```

Reference: **? DEFINE Query Information**

The following information is listed for each virtual field created with DEFINE:

Option	Description
FILE	Is the name of the data source containing the virtual field.
FIELD NAME	Is the name of the virtual field.
FORMAT	Is the format of the virtual field. The notation is the same as that used for the FORMAT attribute in a Master File.
SEGMENT	Is the number of the segment in the Master File containing the virtual field. During reporting, your application treats the virtual field as a field in this segment. To relate segment numbers to segment names, use ? FDT.
VIEW	Is the root segment of DEFINE that specifies an alternate view. For example: `DEFINE FILE EMPLOYEE.JOBCODE`

Option	Description
TYPE	Indicates whether the virtual field is created by the DEFINE attribute in the Master File, or by a DEFINE command, identified by MASTER or a blank, respectively.

Displaying the Currency Data Source in Effect

The ? SET EUROFILE command displays the currency data source in effect.

Syntax: How to Display the Currency Data Source in Effect

To display the currency data source in effect, issue the command:

```
? SET EUROFILE
```

Displaying Available Fields

The ?F command displays the fields that are currently available.

?F displays entire 66 character field names.

Syntax: How to Display Available Fields

```
?F filename
```

where:

filename

Is the name of a data source.

Example: Displaying Available Fields

Issuing the command

```
?F EMPLOYEE
```

produces the following information:

```
FILENAME =  EMPLOYEE
EMP_INFO    EMP_ID       LAST_NAME      FIRST_NAME     HIRE_DATE
DEPARTMENT  CURR_SAL     CURR_JOBCODE   ED_HRS
BANK_NAME   BANK_CODE    BANK_ACCT      EFFECT_DATE
DAT_INC     PCT_INC      SALARY         PAYINFO        JOBCODE
TYPE        ADDRESS_LN1ADDRESS_LN2      ADDRESS_LN3    ACCTNUMBER
PAY_DATE    GROSS
DED_CODE    DED_AMT
JOBSEG      JOBCODE      JOB_DESC
SEC_CLEAR
SKILLS      SKILLS_DESC
DATE_ATTEND ATTENDSEG.EMP_ID
COURSE_CODE COURSE_NAME
```

Displaying the File Directory Table

The ? FDT command displays the file directory table, which lists the physical characteristics of a FOCUS data source.

Each segment and index (those fields designated by the keyword FIELDTYPE=I in the Master File) occupies an integral number of pages. The file directory table shows the amount of space occupied by each segment instance in a page, the starting and ending page numbers, and the number of pages in between for each segment and index.

Syntax: How to Display a File Directory Table

```
? FDT filename
```

where:

filename

Is the name of the data source.

Example: Displaying a File Directory Table

Issuing the command

```
? FDT EMPLOYEE
```

produces the following information:

```
    DIRECTORY:EMPLOYEEFOCUS   F ON 09/25/1997 AT 09.50.28
    DATE/TIME OF LAST CHANGE:   03/30/1999   16.19.22

    SEGNAME   LENGTH   PARENT   START     END    PAGES  LINKS  TYPE

 1  EMPINFO       22                1       1       1      6
 2  FUNDTRAN      10        1       2       2       1      2
 3  PAYINFO        8        1       3       3       1      3
 4  JOBSEG        11        3                               4
 5  SECSEG         4        4                               2
 6  SKILLSEG      11        4                               2
 7  ADDRESS       19        1       4       4       1      2
 8  SALINFO        6        1       5       5       1      3
 9  DEDUCT         5        8       6       8       3      2
10  ATTNDSEG       7        1                               3
11  COURSEG       11       10                               2
>
```

Reference: **? FDT Query Information**

The following information is listed in the file directory table:

SEGNAME	Is the name of each segment in the file. The segments are also numbered consecutively down the left of the table. Unnumbered entries at the foot of the table are indexes, which belong to fields having the attribute FIELDTYPE=I in the Master File.
LENGTH	Is the length in words (units of four bytes) of each segment instance. Divide this number into 992 to get the number of instances that fit on a page.
PARENT	Is the parent segment. Each number refers to a segment name in the SEGNAME column.
START	Is the page number on which the segment or index begins.
END	Is the page number on which the segment or index ends.
PAGES	Is the number of pages occupied by the segment or index.
LINKS	Is the length, in words, of the pointer portion in each segment instance. Every segment instance consists of two parts, data and pointers. Pointers are internal numbers used to find other instances.

TYPE	Is the type of index. NEW indicates a binary index. OLD indicates a hash index. Segments of type KU, LM, DKU, DKM, KL, and KLU are not physically in this file. Therefore, this information is omitted from the table.

Displaying Field Information for a Master File

The ?FF command displays field names, aliases, and format information for an active Master File.

Syntax: How to Display Field Information for a Master File

```
?FF filename [string]
```

where:

filename

Is the name of the Master File.

string

Is a character string up to 66 characters long. The command displays information only for fields beginning with the specified character string. If you omit this parameter, the command displays information for all fields in the Master File.

Example: Displaying Field Information for a Master File

Issuing the command

```
?FF EMPLOYEE
```

produces the following information:

```
FILENAME= EMPLOYEE
EMP_INFO
EMP_ID          EID             A9
LAST_NAME       LN              A15
FIRST_NAME      FN              A10
HIRE_DATE       HDT             16YMD
DEPARTMENT      DPT             A10
CURR_SAL        CSAL            D12.2M
CURR_JOBCODE    CJC             A3
ED_HRS          OJT             F6.2
```

```
BANK_NAME      BN             A20
BANK_CODE      BC             I6S
BANK_ACCT      BA             I9S
EFFECT_DATE    EDATE          16YMD

DAT_INC        DI             I6YMD
PCT_INC        PI             F6.2
SALARY         SAL            D12.2M
PAY_INFOJOBCODEJBC             A3
```

Displaying Data Source Statistics

The ? FILE command displays information, such as the number of segment instances in a FOCUS data source and when the data source was last changed.

Syntax: How to Display Data Source Statistics

```
? FILE filename
```

where:

filename

Is the name of the data source.

Example: Displaying Data Source Statistics

Issuing the command

```
? FILE EMPLOYEE
```

produces statistics similar to the following:

```
    STATUS OF FOCUS FILE: EMPLOYEEFOCUS   A1 ON 03/12/99 AT 12.29.51
              ACTIVE  DELETED    DATE OF   TIME OF     LAST TRANS
SEGNAME       COUNT   COUNT      LAST CHG  LAST CHG    NUMBER

EMPINFO          12             12/21/93  11.01.32         1
FUNDTRAN          6             11/16/89  16.19.19        12
PAYINFO          19             11/16/89  16.19.20        19
ADDRESS          21             11/16/89  16.19.21        21
SALINFO          70             11/16/89  16.19.22       448
DEDUCT          448             11/16/89  16.19.22       448
TOTAL SEGS      576
TOTAL CHARS    8984
TOTAL PAGES       8
LAST CHANGE                     01/29/96  11.01.32         1
```

Reference: **? FILE Query Information**

The following data source statistics are listed:

SEGNAME	Is the name of each segment in the data source. After the segments, the indexes are listed, if applicable. Indexes are those fields specified by the attribute FIELDTYPE=I in the Master File.
ACTIVE COUNT	Is the number of instances of each segment.
DELETED COUNT	Is the number of segment instances deleted, for which the space is not reused.
DATE OF LAST CHG	Is the date on which data in a segment instance or index was last changed.
TIME OF LAST CHG	Is the time of day, on a 24-hour clock, when the last update of a file was made for that segment or index.
LAST TRANS NUMBER	Is the number of transactions performed by the last update request to access the segment. If the data source was changed under Simultaneous Usage mode, this column refers to the REF NUMB column of the CR HLIPRINT file.
TOTAL SEGS	Is the total number of segment instances in the file (shown under ACTIVE COUNT), and the number of segments deleted when the file was last changed (shown under DELETED COUNT).
TOTAL CHARS	Is the number of characters of data in the file.
TOTAL PAGES	Is the number of pages in the data source. Pages are physical records in FOCUS data sources.
LAST CHANGE	Is the date and time the data source was last changed.

If a data source is disorganized by more than 29%, that is, the physical placement of data in the data source is considerably different from its logical or apparent placement, the following message appears

```
FILE APPEARS TO NEED THE -REBUILD- UTILITY
REORG PERCENT IS A MEASURE OF FILE DISORGANIZATION
0 PCT IS PERFECT -- 100 PCT IS BAD
REORG PERCENT IS x%
```

where:

x

Is a percentage between 30 and 100.

The variable &FOCDISORG also indicates the level of disorganization. Following is an example of how to use &FOCDISORG in a Dialogue Manager -TYPE command:

```
-TYPE THE AMOUNT OF DISORGANIZATION OF THIS FILE IS: &FOCDISORG
```

This command, depending on the amount of disorganization, produces a message similar to the following:

```
THE AMOUNT OF DISORGANIZATION OF THIS FILE IS: 10
```

When using a -TYPE command with &FOCDISORG, a message is displayed even if the percentage of disorganization is less than 30%.

Displaying Defined Functions

The ? FUNCTION command displays all defined functions and the parameters.

Syntax: How to Display DEFINE Functions

To display defined functions, issue the command:

```
? FUNCTION
```

Example: Displaying DEFINE Functions

Issuing the command

```
? FUNCTION
```

produces information similar to the following:

```
Name    Format   Parameter   Format
DIFF    D8       VAL1        D8
                 VAL2        D8
```

Displaying HOLD Fields

The ? HOLD command lists fields described in a Master File created by the ON TABLE HOLD command. The list displays the field names, the aliases, and the formats as defined by the FORMAT (USAGE) attribute. The ? HOLD command displays field names up to 32 characters. If a field name exceeds 32 characters, an ampersand (&) in the 32nd position indicates a longer field name.

The ? HOLD command displays fields of a HOLD Master File created by the current request.

Syntax: How to Display HOLD Fields

```
? HOLD [filename]
```

where:

filename

Is the name assigned in the AS phrase in the ON TABLE HOLD command. If you omit the file name, it defaults to HOLD.

Example: Displaying HOLD Fields

Issuing the command

```
? HOLD
```

produces information similar to the following:

```
DEFINITION OF CURRENT HOLD FILE FIELDNAME ALIAS FORMAT COUNTRY E01 A10
CAR E02 A16
```

Displaying JOIN Structures

The ? JOIN command lists the JOIN structures currently in effect. The command displays field names up to 12 characters. If a field name exceeds 12 characters, an ampersand in the twelfth position indicates a longer field name.

Syntax: How to Display JOIN Structures

To display JOIN structures, issue the command:

```
? JOIN
```

Example: Displaying JOIN Structures

Issuing the command

```
? JOIN
```

produces information similar to the following:

```
JOINS CURRENTLY ACTIVE
HOST                        CROSSREFERENCE
FIELD     FILE      TAG     FIELD            FILE     TAG   AS  ALL   WH
-----     ----      ---     -----            ----     ---   --  ---   --
JOBCODE   EMPLOYEE          JOBCODE          JOBFILE            N     N
```

Reference: ? JOIN Query Information

The following JOIN information is listed:

`HOST FIELD`	Is the name of the host field that is joining the data sources.
`FILE`	Is the name of the host data source.
`TAG`	Is a tag name used as a unique qualifier for field names in the host data source.
`CROSSREFERENCE FIELD`	Is the name of the cross-referenced field used to join the data sources.
`FILE`	Is the name of the cross-referenced data source.
`TAG`	Is a tag name used as a unique qualifier for field names in the cross-referenced data source.
`AS`	Is the name of the joined structure.
`ALL`	Displays Y for a non-unique join and N for a unique join.
`WH`	Specifies whether the join is a conditional join or an equi-join.

Displaying National Language Support

The ? LANG command displays information about National Language Support.

Syntax: How to Display Information About National Language Support

To display information about National Language Support:

```
? LANG
```

Example: Displaying Information About National Language Support

Issuing the command

```
? LANG
```

produces information similar to the following:

```
        LANGUAGE AND DBCS STATUS

Language                01/AMENGLISH   (    )
Code Page               00037
Dollar value            5B($)
DBCS Flag               OFF(SBCS)
```

Displaying LET Substitutions

The ? LET command lists the active word substitutions created by the LET command. A word in the left column is used in a report request to represent the word or phrase in the right column. For more information on the LET command, see Defining a Word Substitution.

Syntax: How to Display LET Substitutions

To display word substitutions, issue the command:

```
? LET
```

Example: Displaying LET Substitutions

Issuing the command

```
? LET
```

produces information similar to the following:

```
PR                    PRINT
TF                    TABLE FILE EMPLOYEE
```

Displaying Information About Loaded Files

The ? LOAD command displays the file type, file name, and resident size of currently loaded files.

Syntax: How to Display Information About Loaded Files

```
? LOAD [filetype]
```

where:

filetype

Specifies the type of file (MASTER, FOCEXEC, Access File, FOCCOMP, or MODIFY) on which information displays. To display information on all memory-resident files, omit file type.

Example: Displaying Information About Loaded Files

Issuing the command

```
? LOAD
```

produces information similar to the following:

```
FILES CURRENTLY LOADED
CARCRYPTMODIFY 00213548 BYTES
CAR MASTER
VIDEOACXFOCSQL
CARCRYPTFOCEXEC
RTEST    FOCEXEC  8400   BYTES
```

Displaying Explanations of Error Messages

The ? *n* command displays a detailed explanation of an error message, providing assistance in correcting the error.

Error messages generated by certain data adapters, such as the DB2 and MODEL 204 data adapters, are also accessible through this feature.

Syntax: How to Display Explanations of Error Messages

```
? n
```

where:

n

Is the error message number.

Example: Displaying Explanations of Error Messages

If you receive the message

```
(FOC125) RECAP CALCULATIONS MISSING
```

issuing the command

```
? 125
```

produces the following message:

```
(FOC125) RECAP CALCULATIONS MISSING
The word RECAP is not followed by a calculation. Either the RECAP
should be removed, or a calculation provided.
```

Displaying PF Key Assignments

The ? PFKEY command displays the PF key assignments.

Syntax: How to Display PF Key Assignments

To display the PF key assignments, issue the command:

```
? PFKEY
```

Example: Displaying PF Key Assignments

Issuing the command

```
? PFKEY
```

produces results similar to the following:

```
PF01 = HX         PF02 = CANCEL     PF03 = END        PF04 = RETURN
PF05 = RETURN     PF06 = SORT       PF07 = BACKWARD   PF08 = FORWARD
PF09 = RETURN     PF10 = LEFT       PF11 = RIGHT      PF12 = UNDO
PF13 = RETURN     PF14 = RETURN     PF15 = END        PF16 = RETURN
PF17 = RETURN     PF18 = RETURN     PF19 = BACKWARD   PF20 = FORWARD
PF21 = RETURN     PF22 = RETURN     PF23 = RETURN     PF24 = UNDO
```

Displaying the Release Number

The ? RELEASE command displays the number of the currently installed release of your product.

Syntax: How to Display the Release Number

To display the release number, issue the command:

```
? RELEASE
```

Example: Displaying the Release Number

Issuing the command

```
? RELEASE
```

produces information similar to the following:

```
FOCUS 7.2.0            created 11/07/2001 11.38.32
```

Displaying Parameter Settings

The ? SET command lists the parameter settings that control your FOCUS environment. Your application sets default values for these parameters, but you can change them with the SET command.

SET parameters are described in *Customizing Your Environment*.

Syntax: How to Display Parameter Settings

```
? SET [ALL|[FOR] parameter]
```

where:

`ALL`

Displays all possible parameter settings.

`parameter`

Is a SET parameter. This displays the setting for the specific parameter.

`FOR`

Includes where the parameter can be set from in addition to the parameter setting.

Example: **Displaying Parameter Settings**

Issuing the command

```
? SET
```

produces information similar to the following:

```
                          PARAMETER SETTINGS
ALL.                  OFF   FOCSTACK SIZE         8   QUALCHAR               .
ASNAMES             FOCUS   FOC2GIGDB           OFF   QUALTITLES           OFF
AUTOINDEX              ON   HDAY                      REBUILDMSG          1000
AUTOPATH               ON   HIPERFOCUS          OFF   RECAP-COUNT          OFF
BINS                   64   HOLDATTRS         FOCUS   SAVEMATRIX           OFF
BLKCALC               NEW   HOLDLIST            ALL   SCREEN                ON
BUSDAYS          _MTWTF_    HOLDSTAT            OFF   SHADOW PAGE          OFF
BYPANELING            OFF   HOTMENU             OFF   SPACES               AUTO
CACHE                   0   IBMLE               OFF   SQLENGINE
CARTESIAN             OFF   INDEX TYPE          NEW   SUMPREFIX             LST
CDN                   OFF   LANGUAGE      AMENGLISH   TCPIPINT              OFF
COLUMNSCROLL          OFF   LINES/PAGE           66   TEMP DISK              A
DATEDISPLAY           OFF   LINES/PRINT          57   TERMINAL         IBM3270
DATEFNS                ON   MESSAGE             ON   TESTDATE           TODAY
DATETIME   STARTUP/RESET    MODE                    TITLES                ON
DEFCENT                19   MULTIPATH        SIMPLE   VIEWNAMESIZE          18
EMPTYREPORT           OFF   NODATA                .   WIDTH                 80
EXL2KLANG               1   PAGE-NUM             ON   WINPFKEY              OLD
EXTAGGR                ON   PANEL                0   XFBINS       16 (passive)
EXTHOLD                ON   PAUSE               ON   XFOCUS               OFF
EXTSORT                ON   XRETRIEVAL          ON
FIELDNAME             NEW   PRINT             ONLINE   YRTHRESH              0
FOCALLOC              OFF   PRINTPLUS           OFF
```

Some parameters are listed differently from the way you specify them in the SET command. These include:

SET Parameters	Description
FOCSTACK SIZE	Is the same as the FOCSTACK parameter.
INDEX TYPE	Is the same as the INDEX parameter.
LINES/PAGE	Is the same as the PAPER parameter.
LINES/PRINT	Is the same as the LINES parameter.
SHADOW PAGES	Is the same as the SHADOW parameter.

Example: Displaying a Single Parameter Setting

Issuing the command

```
? SET PAGESIZE
```

produces the following if the parameter is set to its default value:

```
PAGESIZE Letter
```

Example: Displaying Where a Parameter Can Be Set

Issuing the command

```
? SET FOR EXTSORT
```

produces the following information:

```
EXTSORT ON

  ------------------------------------------------
  SETTABLE FROM COMMAND LINE                  : YES
  SETTABLE ON TABLE                           : YES
  SETTABLE FROM SYSTEM-WIDE PROFILE           : YES
  SETTABLE FROM HLI PROFILE                   : YES
```

Displaying Graph Parameters

The ? SET GRAPH command lists the parameter settings that control graphs produced with the GRAPH command. These parameters are described further in *Customizing Your Environment*.

Syntax: How to Display Graph Parameters

To display graph parameters, issue the command:

```
? SET GRAPH
```

Example: Displaying Graph Parameters

Issuing the command

```
? SET GRAPH
```

produces information similar to the following:

```
              GRAPH PARAMETER SETTINGS
AUTOTICK                ON        HISTOGRAM    ON
BARNUMB                OFF        HMAX        .00
BARSPACE                 0        HMIN        .00
BARWIDTH                 1        HSTACK      OFF
BSTACK                 OFF        HTICK       .00
DEVICE              IBM3270       PIE         OFF
GMISSING               OFF        VAUTO        ON
GMISSVAL               .00        VAXIS        66
GPROMPT                OFF        VCLASS      .00
GRIBBON(GCOLOR)        OFF        VGRID       OFF
GRID                   OFF        VMAX        .00
GTREND                 OFF        VMIN        .00
HAUTO                   ON        VTICK       .00
HAXIS                  130        VZERO       OFF
HCLASS                 .00

>
```

If you change the PLOT parameter settings, a small table appears at the end of the list:

```
PLOT TABLE (EBCDIC):

ENTER PLOT MODE   0050 (FOR 3284 WIDTH)
EXIT PLOT MODE    0018 (FOR 3284 HEIGHT)
LEFT              0000
RIGHT             0000
UP                0000
DOWN              0000
```

The entries in the table at the bottom are:

`ENTER PLOT MODE`	Width of graph on IBM 3284 or 3287 printer.
`EXIT PLOT MODE`	Height of graph on IBM 3284 or 3287 printer.

Ignore the parameters LEFT, RIGHT, UP, and DOWN.

Displaying the Site Code

The FOCUS site code is installed as part of the License Management facility.

Once the site code has been installed, you can retrieve its value by issuing the ? SITECODE query command. If the site code has not been installed, you will get a message indicating that the site code is not available.

Syntax: How to Retrieve the Site Code

```
? SITECODE
```

Example: Querying the Site Code

Assume you installed the License Management facility with site code A52709b.

Issue the following query command in a FOCEXEC:

```
? SITECODE
```

The output is:

```
SITE CODE A5270 9b
```

If the site code is not installed, the ? SITECODE query returns the following message:

```
SITE CODE NOT AVAILABLE
```

Displaying Command Statistics

The ? STAT command lists statistics for the most recently executed command.

Each statistic applies only to a certain command. If another command is executed, the statistic is either 0 or does not appear in the list at all. When you execute commands in stored procedures, these statistics are automatically stored in Dialogue Manager statistical variables. See your Dialogue Manager documentation for details.

Syntax: How to Display Command Statistics

To display command statistics, issue the command:

```
? STAT
```

Example: Displaying Command Statistics

Issuing the command

```
? STAT
```

produces information similar to the following:

```
                    STATISTICS OF LAST COMMAND
RECORDS         =      0           SEGS DELTD      =      0
LINES           =      0           NOMATCH         =      0
BASEIO          =      0           DUPLICATES      =      0
SORTIO          =     47           FORMAT ERRORS   =      0
SORT PAGES      =      0           INVALID CONDTS  =      0
READS           =      0           OTHER REJECTS   =      0
TRANSACTIONS    =      0           CACHE READS     =      0
ACCEPTED        =      0           MERGES          =      0
SEGS INPUT      =      0           SORT STRINGS    =      0
SEGS CHNGD      =      0           INDEXIO         =      0

INTERNAL MATRIX CREATED:   YES     AUTOINDEX USED: NO
SORT USED:               FOCUS     AUTOPATH USED:  NO
AGGREGATION BY EXT.SORT:    NO     HOLD FROM EXTERNAL SORT: NO
```

Reference: ? STAT Query Information

The following information displays:

RECORDS
: Is for TABLE, TABLEF, and MATCH commands. It indicates the number of data source records used in the report. The meaning of a record depends on the type of data source used.

LINES
: Is for TABLE and TABLEF commands. It indicates the number of lines displayed in a report.

BASEIO
: Is for TABLE, TABLEF, GRAPH, MODIFY, and FSCAN command. It indicates the number of I/O operations performed on the data source.

SORTIO
: Is for TABLE, TABLEF, GRAPH, and MATCH commands. It indicates the number of I/O operations performed on the FOCSORT file, which is a work file invisible to the end user.

SORTPAGES
: Is for TABLE and TABLEF commands. It indicates the number of physical records in the FOCSORT file.

READS
: Is for the MODIFY and FSCAN commands. It indicates the number of fixed format records read in external files by the FIXFORM command.

TRANSACTIONS
: Is for the MODIFY and FSCAN commands. It indicates the number of transactions processed.

ACCEPTED
Is for the MODIFY and FSCAN commands. It indicates the number of transactions accepted.

SEGS INPUT
Is for the MODIFY and FSCAN commands. It indicates the number of segment instances accepted in the data source.

SEGS CHNGD
Is for the MODIFY and FSCAN commands. It indicates the number of segment instances updated in the data source.

SEGS DELTD
Is for the MODIFY and FSCAN commands. It indicates the number of segment instances deleted from the data source.

NOMATCH
Is for the MODIFY and FSCAN commands. It indicates the number of transactions rejected for lack of matching values in the data source. This occurs on an ON NOMATCH REJECT condition.

DUPLICATES
Is for the MODIFY and FSCAN commands. It indicates the number of transactions rejected because the matching field values already exist in the data source. This occurs on an ON MATCH REJECT condition.

FORMAT ERRORS
Is for the MODIFY and FSCAN commands. It indicates the number of transactions rejected because field values for data fields do not conform to the field formats defined in the Master File.

INVALID CONDTS
Is for the MODIFY and FSCAN commands. It indicates the number of transactions rejected because field values for data fields do not conform to the field formats defined in the Master File.

OTHER REJECTS
Is for the MODIFY and FSCAN commands. It indicates the number of transactions rejected for reasons other than those listed above.

CACHE READS
Is the number of cache reads performed. For details, see *CACHE* on page 66.

MERGES
Is the number of times that merge routines were invoked.

`SORT STRINGS`
: Is the number of times that the sort capacity was exceeded.

`INTERNAL MATRIX CREATED`
: Indicates how report sorting was handled. If an external sort handled it entirely, the value is NO. If both the application and an external sort handled it, the value is Y.

`SORT USED`
: Is the type of sort facility used. It can have a value of FOCUS, EXTERNAL, SQL, or NONE. NONE means that the report did not require sorting.

`AGGREGATION BY EXT. SORT`
: Uses external sorts to perform aggregation.

`AUTOINDEX USED`
: Automatically takes advantage of indexed fields to speed data retrieval.

`AUTOPATH USED`
: Selects an optimal retrieval path for accessing a data source.

`HOLD FROM EXTERNAL SORT`
: Creates hold files with an external sort.

Displaying StyleSheet Parameter Settings

The ? STYLE command displays the current settings for StyleSheet parameters.

Syntax: How to Display StyleSheet Parameter Settings

```
? [SET] STYLE
```

Example: Displaying StyleSheet Parameter Settings

Issuing the command

```
? STYLE
```

produces information similar to the following:

```
ONLINE-FMT
OFFLINE-FMT        STANDARD
STYLESHEET         ON
SQUEEZE            OFF
PAGESIZE           LETTER
ORIENTATION        PORTRAIT
UNITS              INCHES
LABELPROMPT        OFF
LEFTMARGIN         .250
RIGHTMARGIN        .250
TOPMARGIN          .250
BOTTOMMARGIN       .250
STYLEMODE          FULL
TARGETFRAME
FOCEXURL
BASEURL
```

Note: OFFLINE-FMT is not supported. ONLINE-FMT and FOCEXURL apply to WebFOCUS.

Reference: ? STYLE Query Information

The following StyleSheet information is listed:

STYLESHEET	Rejects or accepts StyleSheet parameters that specify formatting options, such as page size, orientation, and margins.
LABELPROMPT	Specifies on which label of the first page to begin printing a multi-pane report, such as a mailing label report.
STYLEMODE	Speeds the retrieval of large report output by displaying output in multiple HTML tables where each table is a separate report page.
ORIENTATION	Is the page orientation for styled reports. Can be either portrait or landscape.
UNITS	Is the unit of measure for PostScript and PDF report output, as inches, centimeters, or points.
TOPMARGIN	Is the top boundary for a page of report output.
BOTTOMMARGIN	Is the bottom boundary for a page of report output.
LEFTMARGIN	Is the left boundary for a page of report output.

RIGHTMARGIN	Is the right boundary for a page of report output.
TARGETFRAME	Is a frame to which all drill-down hyperlinks are directed.
BASEURL	Is the default location where the browser searches for relative URLs specified in the HTML documents created by your application.

Displaying Information About the SU Machine

The ? SU command displays the communication available to the FOCUS Database Server.

Syntax: How to Display Information About the FOCUS Database Server

```
? SU [userid|ddname]
```

where:

userid

Is a sync machine user ID.

ddname

Is a valid ddname.

Example: Displaying Information About the FOCUS Database Server

Issuing the command

```
? SU SYNCA
```

produces the following information:

```
USERID      FILEID      QUEUE
WIBMLH      QUERY
WIBJBP      CAR
```

Displaying Data Sources Specified With USE

The ? USE command displays data sources specified with the USE command.

Syntax: How to Display Data Sources Specified With USE

To display data sources specified with the USE, issue the command:

```
? USE
```

Example: Displaying Data Sources Specified With USE

Issuing the command

```
? USE
```

produces information similar to the following:

```
DIRECTORIES IN USE ARE:
CAR         FOCUS       F
EMPLOYEE    FOCUS       F
LEDGER      FOCUS       F
```

Displaying Global Variable Values

The ? && command lists Dialogue Manager global variables and the current values. Global variables maintain the values for all procedures executed during a FOCUS session.

Note: You can query all Dialogue Manager variables (local, global, system, and statistical) from a stored procedure by issuing:

```
-? &
```

See your Dialogue Manager documentation for details.

Syntax: How to Display Global Variable Values

```
? &&
```

Your site may replace the ampersand (& or &&) indicating Dialogue Manager variables, with another symbol. In that case, use the replacement symbol in your query command. For example, if your installation uses the percent sign (%) to indicate Dialogue Manager variables, list global variables by issuing:

```
? %%
```

Example: Displaying Global Variable Values

Issuing the command

```
? &&
```

produces information similar to the following:

```
&&STORECODE   001
&&STORENAME   MACYS
```

Reporting Dynamically From System Tables

You can issue report requests against a set of synonyms that dynamically gather information about your environment, including its applications, files, columns, directories, tables, indexes, and keys. You can also retrieve information about functions, SET parameters, and error files. These synonyms reside in the MASTER.DATA data set and have the suffix FMI (FOCUS Metadata Interface).

Overview of System Table Synonyms

Each FMI synonym retrieves information about a specific set of files in your environment. If you examine an FMI Master File, the REMARKS and DESCRIPTION attributes document what data will be returned by each system table and each column within the system table.

Note: The system table synonyms may change in future releases. Therefore, you should not design applications that depend on the structure of the system table synonyms.

For example, following is a version of the SYSFILES Master File, which, by default, retrieves information about Master Files in your application path.

```
$-------------------------------------------------------------------------------$
$ Copyright (c) 2013 Information Builders, Inc. All rights reserved. @MFSM_NOPROLOG@ $
$-------------------------------------------------------------------------------$
$--CAN BE USED TO RETRIEVE DIRECTORY INFO - USE THE FOLLOWING TO DEFINE DIRECTORY
$--SQL FMI SET SYSFILES EDASYNM
$--SQL FMI SET SYSFILES FOCEXEC
FILE=SYSFILES, SUFFIX=FMI, REMARKS='Metadata: Directory information', $
SEGMENT=FILE,SEGTYPE=S0,$
  FIELD=FILENAME  ,      ,A64  ,A64B,    DESC='MEMBER  NAME
',$
  FIELD=LGNAME    ,      ,A8   ,A8B ,    DESC='LOGICAL NAME
',$
  FIELD=PHNAME    ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  1ST PART
',$
  FIELD=PHNAME2   ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  2ND PART
',$
  FIELD=PHNAME3   ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  ETC..
',$
  FIELD=PHNAME4   ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  The whole length up to
512 bytes                      ',$
  FIELD=PHNAME5   ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  The last part is 32 but
can be up                      ',$
  FIELD=PHNAME6   ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  to 44 because of nlscut
of ',$
  FIELD=PHNAME7   ,      ,A80  ,A80B,    DESC='PHYSICAL NAME  previous 6 parts (2
bytes per part)',$
  FIELD=VERSION   ,      ,I4   ,I1,     DESC='MF:VERSION
',$
  FIELD=MOD       ,      ,I4   ,I1,     DESC='MF:MODIFICATION NUMBER
',$
  FIELD=LINECNT   ,      ,I4   ,I2,     DESC='MF:CURRENT LINE COUNTER.
',$
  FIELD=DATE      ,      ,A8   ,A8B,     DESC='IBI DATE (DD/MM/YY)
',$
  FIELD=TIME      ,      ,A8   ,A8B,     DESC='IBI TIME (HH.MM.SS)
',$
  FIELD=USERID    ,      ,A100 ,A100B,   DESC='MF: LAST USER WHO CHANGED. UNIX/
NT:owner',$
  FIELD=SIZE      ,      ,I11  ,I4,     DESC='UNIX/NT:SIZE IN BYTES.
',$
  FIELD=EXTENSION ,      ,A3   ,A3B,     DESC='ACCEPTED SHORT EXTENSION FOR FILE
',$
```

A list of some of the most useful FMI synonyms follows. You can generate a list of system table synonyms by issuing a request against the systable synonym.

- **SYSAPPS.** Retrieves information about applications and the files within them.
- **SYSCOLUM.** Retrieves information about tables and their columns.
- **SYSDEFFN.** Retrieves DEFINE FUNCTION information.

- **SYSERR.** Retrieves information about error message files.
- **SYSFILES.** Retrieves directory information.
- **SYSIMP.** Retrieves impact analysis information.
- **SYSINDEX.** Retrieves index information.
- **SYSKEYS.** Retrieves information about keys.
- **SYSRPDIR.** Retrieves information about all available FOCEXECs in your application path.
- **SYSSET.** Retrieves information about SET commands and global Dialogue Manager variables.
- **SYSSQLOP.** Retrieves function information.
- **SYSTABLE.** Retrieves table information.
- **SYSVDTP.** Retrieves data type information for Relational Adapters.

SYSAPPS: Reporting on Applications and Application Files

The sysapps synonym retrieves information about applications and the files within them.

Example: Retrieving Application and File Information

The following request retrieves the application name and path and the file name, extension, suffix, keys, and number of segments for Master Files in the ibisamp application, where the file names start with the letters a through g.

```
TABLE FILE SYSAPPS
PRINT APPNAME AS App APPLOC AS Path
FNAME AS 'File,Name' SUFFIX AS 'File,Type' KEYS
NUMSEG AS '# of,Segments'
WHERE APPNAME EQ 'ibisamp'
WHERE FEXT EQ 'mas'
WHERE FQNAME LT 'C:\ibi\apps\ibisamp\hday'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

App	Path	File Name	File Type	KEYS	# of Segments
ibisamp	C:\ibi\apps\ibisamp	brokers	FOC	BROKER_ID	1
ibisamp	C:\ibi\apps\ibisamp	car	FOC	COUNTRY CAR MODEL BODYTYPE WARRANTY STANDARD	7
ibisamp	C:\ibi\apps\ibisamp	carolap	FOC	COUNTRY CAR MODEL BODYTYPE WARRANTY STANDARD	7
ibisamp	C:\ibi\apps\ibisamp	cashflow	FOC	CASH_DATE	1
ibisamp	C:\ibi\apps\ibisamp	course	FOC	COURSECODE	1
ibisamp	C:\ibi\apps\ibisamp	courses	FOC	COURSE_CODE	1
ibisamp	C:\ibi\apps\ibisamp	educfile	FOC	COURSE_CODE DATE_ATTEND EMP_ID	2
ibisamp	C:\ibi\apps\ibisamp	empdata	FOC	PIN	1
ibisamp	C:\ibi\apps\ibisamp	employee	FOC	EMP_ID DAT_INC TYPE PAY_DATE DED_CODE SKILLS DATE_ATTEND	11
ibisamp	C:\ibi\apps\ibisamp	experson	FOC	SOC_SEC_NO	1
ibisamp	C:\ibi\apps\ibisamp	filemntr	FIX		1
ibisamp	C:\ibi\apps\ibisamp	finance	FOC	YEAR ACCOUNT	1
ibisamp	C:\ibi\apps\ibisamp	ggdemog	FOC	ST	1
ibisamp	C:\ibi\apps\ibisamp	ggorder	FOC	ORDER_NUMBER	2
ibisamp	C:\ibi\apps\ibisamp	ggprods	FOC	PRODUCT_ID	1
ibisamp	C:\ibi\apps\ibisamp	ggsales	FOC	SEQ_NO	1
ibisamp	C:\ibi\apps\ibisamp	ggstores	FOC	STORE_CODE	1

SYSCOLUM: Reporting on Tables and Their Columns

The syscolum synonym retrieves table information, including table names creator names, segment names and numbers, segment roles in a dimension view or business view, column names, and column data types. Use it to report on data sources referenced in a Master File.

Example: Retrieving Table and Column Information

The following request retrieves table and column information from tables whose table names start with the characters *wf_*.

```
TABLE FILE SYSCOLUM
PRINT TBNAME AS Table TBTYPE AS Suffix NAME AS Field,Name COLTYPE AS
Data,Type ACTUAL AS Format
WHERE TBNAME LIKE 'wf_%'
WHERE RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Table	Suffix	Field Name	Data Type	Format
wf_retail	SQLMSS	ID_SALES	INTEGER	I4
wf_retail	SQLMSS	ID_STORE	INTEGER	I4
wf_retail	SQLMSS	ID_CURRENCY	INTEGER	I4
wf_retail	SQLMSS	ID_CUSTOMER	INTEGER	I4
wf_retail	SQLMSS	ID_DISCOUNT	INTEGER	I4
wf_retail	SQLMSS	ID_PRODUCT	INTEGER	I4
wf_retail	SQLMSS	ID_TIME	INTEGER	I4
wf_retail	SQLMSS	COGS_LOCAL	DOUBLE	D8
wf_retail	SQLMSS	COGS_US	DOUBLE	D8
wf_retail	SQLMSS	DISCOUNT_LOCAL	DOUBLE	D8
wf_retail	SQLMSS	DISCOUNT_US	DOUBLE	D8
wf_retail	SQLMSS	GROSS_PROFIT_LOCAL	DOUBLE	D8
wf_retail	SQLMSS	GROSS_PROFIT_US	DOUBLE	D8
wf_retail	SQLMSS	MSRP_LOCAL	DOUBLE	D8
wf_retail	SQLMSS	MSRP_US	DOUBLE	D8
wf_retail	SQLMSS	QUANTITY_SOLD	INTEGER	I4
wf_retail	SQLMSS	REVENUE_LOCAL	DOUBLE	D8
wf_retail	SQLMSS	REVENUE_US	DOUBLE	D8
wf_retail	SQLMSS	SALE_UNITY	INTEGER	
wf_retail	SQLMSS	ID_SHIPFACT	INTEGER	I4

SYSDEFFN: Reporting on DEFINE FUNCTIONS

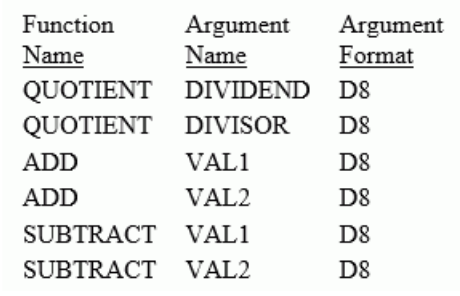
The sysdeffn synonym retrieves information about DEFINE FUNCTIONs, including function names, arguments, argument formats, function fields, and descriptions.

Example: Retrieving DEFINE FUNCTION Information

The following request retrieves DEFINE FUNCTION names, arguments, and argument formats.

```
TABLE FILE SYSDEFFN
PRINT DFNAME AS Function,Name ARGNAME AS Argument,Name ARGFORMAT AS
Argument,Format
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.



Function Name	Argument Name	Argument Format
QUOTIENT	DIVIDEND	D8
QUOTIENT	DIVISOR	D8
ADD	VAL1	D8
ADD	VAL2	D8
SUBTRACT	VAL1	D8
SUBTRACT	VAL2	D8

SYSERR: Reporting on Error Message Files

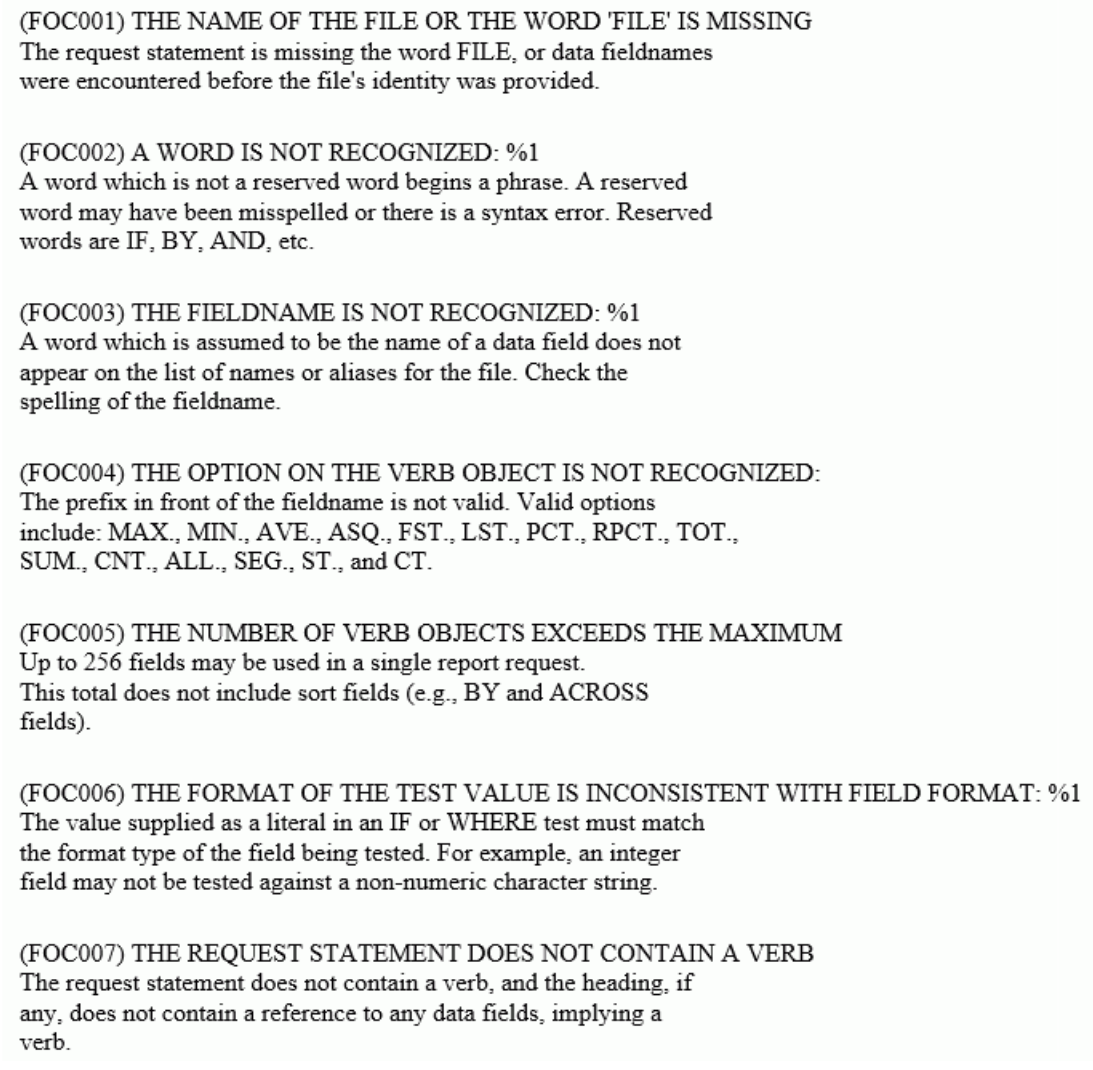
The syserr synonym retrieves error file names, the lowest and highest message numbers in each file, message and explanation text, message number, whether the message is a warning, whether the message is informational, and whether the line number is displayed in a procedure.

Example: Retrieving Error Message File Information

The following request retrieves message text and explanations.

```
TABLE FILE SYSERR
BY ERRNUM NOPRINT SUBHEAD
" <ERRTEXT  "
 "<ERRLINE1 "
 "<ERRLINE2 "
 "<ERRLINE3 "
 "<ERRLINE4 "
WHERE RECORDLIMIT EQ 7
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.



SYSFILES: Reporting on Metadata or Procedure Directory Information

The sysfiles synonym retrieves Master Files or FOCEXEC files in your application path. By default, sysfiles retrieves a list of Master Files and their properties. The SET SYSFILES command determines which type of files are retrieved.

The syntax is

```
SQL FMI SET SYSFILES {EDASYNM|FOCEXEC}
```

where:

EDASYNM
Retrieves information about Master Files. This is the default value.

FOCEXEC
Retrieves information about procedure files.

Example: Retrieving Master File Information

The following request retrieves the file name, extension, and path for the Master File names that start with the letters a through h in the ibisamp application directory.

```
TABLE FILE SYSFILES
PRINT FILENAME AS File
EXTENSION AS Extension
PHNAME AS Path
WHERE PHNAME LIKE 'ibisamp/%'
WHERE FILENAME LT 'i'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

File	Extension	Path
brokers	mas	ibisamp/brokers.mas
car	mas	ibisamp/car.mas
carolap	mas	ibisamp/carolap.mas
cashflow	mas	ibisamp/cashflow.mas
course	mas	ibisamp/course.mas
courses	mas	ibisamp/courses.mas
educfile	mas	ibisamp/educfile.mas
empdata	mas	ibisamp/empdata.mas
employee	mas	ibisamp/employee.mas
experson	mas	ibisamp/experson.mas
filemntr	mas	ibisamp/filemntr.mas
finance	mas	ibisamp/finance.mas
ggdemog	mas	ibisamp/ggdemog.mas
ggorder	mas	ibisamp/ggorder.mas
ggprods	mas	ibisamp/ggprods.mas
ggsales	mas	ibisamp/ggsales.mas
ggstores	mas	ibisamp/ggstores.mas
hday	mas	ibisamp/hday.mas

SYSIMP: Reporting on Impact Analysis Information

The sysimp synonym retrieves information about where files reside and where they are referenced. Sysimp contains a segment for caller information and a child segment for the files called by each caller.

Example: Retrieving Impact Analysis Information

The following request retrieves the names, types, and descriptions of caller files whose names start with the letters s through z in the ibisamp application and the names, types, and descriptions of the files they called.

```
TABLE FILE SYSIMP
PRINT CTYPE AS Caller,Type CDESCRIPTION AS Description
RFILE AS Called,Name
RPT_TYPE AS Called,Type
RLINENUM AS Line RUSAGE AS 'Used In'
REXTENSION AS Extension RDESCRIPTION AS Description
BY CFILE/A15  AS Caller,Name
WHERE CAPPLICATION EQ 'ibisamp'
WHERE CFILE GE 's'
ON TABLE SET PAGE NOLEAD
ON TABLE SET SHOWBLANKS ON
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The partial output is shown in the following image.

Caller Name	Caller Type	Description	Called Name	Called Type	Line	Used In	Extension	Description
subfoot1	Procedure		car	MFD	2	TABLE	mas	Legacy Metadata Sample: car
wf_retail	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_sales	MFD	4	CRFILE	mas	Sales Fact
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_customer	MFD	42	CRFILE	mas	Customer Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_age	MFD	211	CRFILE	mas	Age Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_time	MFD	967	CRFILE	mas	Time Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_time	MFD	935	CRFILE	mas	Time Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_industry	MFD	872	CRFILE	mas	Industry Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_occupation	MFD	344	CRFILE	mas	Occupation Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_marital_status	MFD	339	CRFILE	mas	Marital Status Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_income	MFD	326	CRFILE	mas	Income Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_geography	MFD	231	CRFILE	mas	Geography Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_education	MFD	220	CRFILE	mas	Education Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_currency	MFD	202	CRFILE	mas	Currency Dimension
	Synonym	Cluster Join of Fact Tables Sales, Shipments and Labor for Demo Database	wf_retail_time	MFD	170	CRFILE	mas	Time Dimension

SYSINDEX: Reporting on Index Information

The sysindex synonym retrieves information about indexes defined in a synonym.

Example: Retrieving Index Information

The following request retrieves index field names for files that start with the characters *gg*.

```
TABLE FILE SYSINDEX
PRINT NAME AS Index
BY TBNAME AS File
WHERE TBNAME LIKE 'gg%'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

File	Index
ggdemog	ST
ggorder	PRODUCT_ID
	VENDOR_CODE
ggprods	PRODUCT_ID
	VENDOR_CODE
ggsales	CATEGORY
	PCD
	REGION
	ST
	STCD
ggstores	STORE_CODE
	STATE

SYSKEYS: Reporting on Key Information

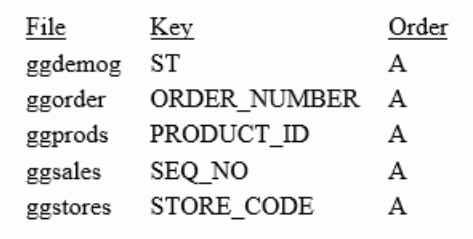
The syskeys synonym retrieves information about keys defined in a synonym.

Example: Retrieving Key Information

The following request retrieves key field names and sort order for files that start with the characters *gg*.

```
TABLE FILE SYSKEYS
PRINT IXNAME AS Key ORDERING AS Order
BY TBNAME AS File
WHERE TBNAME LIKE 'gg%'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.



File	Key	Order
ggdemog	ST	A
ggorder	ORDER_NUMBER	A
ggprods	PRODUCT_ID	A
ggsales	SEQ_NO	A
ggstores	STORE_CODE	A

SYSRPDIR: Reporting on Stored Procedures

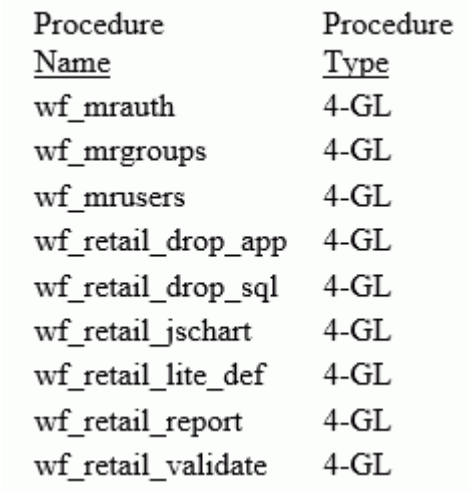
The sysrpdir synonym retrieves all available FOCEXECs in your application path.

Example: Retrieving Stored Procedure Information

The following request retrieves procedures that start with the characters *wf_*.

```
TABLE FILE SYSRPDIR
PRINT RPC_TYPE AS Procedure,Type
BY RPC_NAME AS Procedure,Name
WHERE RPC_NAME LIKE 'wf_%'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.



Procedure Name	Procedure Type
wf_mrauth	4-GL
wf_mrgroups	4-GL
wf_mrusers	4-GL
wf_retail_drop_app	4-GL
wf_retail_drop_sql	4-GL
wf_retail_jschart	4-GL
wf_retail_lite_def	4-GL
wf_retail_report	4-GL
wf_retail_validate	4-GL

SYSSET: Reporting on SET Parameters

The sysset synonym retrieves information about SET parameters, their allowed values, and their default values.

Example: Retrieving Information About SET Parameters

The following request displays SET parameters that start with the letter D, along with their descriptions and values.

```
TABLE FILE SYSSET
PRINT SETDESC CURR_VALUE VALUE IS_DEFAULT
BY SETNAME AS Set,Name
WHERE SETNAME GE 'D' AND SETNAME LT 'E'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The partial output is shown in the following image.

Set Name	Description	Current Value	Available Value	Default
DATEDISPLAY	Date display value when date functions evaluate as zero.	OFF	OFF	1
	Date display value when date functions evaluate as zero.	OFF	ON	0
	Date display value when date functions evaluate as zero.	OFF	COMP	0
DATEFNS	Activates year 2000-compliant versions of date subroutines.	ON	OFF	0
	Activates year 2000-compliant versions of date subroutines.	ON	ON	1
DATEFORMAT	Format of input string in DT() date/time function.	MDY	MDY	1
	Format of input string in DT() date/time function.	MDY	YMD	0
	Format of input string in DT() date/time function.	MDY	DMY	0
	Format of input string in DT() date/time function.	MDY	MYD	0
DATEOUTPUT	Enable locale-sensitive Date Format	DEFAULT	.	.
DATETIME	Sets time and date in reports and controls the format in which CREATE SYNONYM creates date-time columns in a Master File.	STARTUP	STARTUP	0
	Sets time and date in reports and controls the format in which CREATE SYNONYM creates date-time columns in a Master File.	STARTUP	STARTUP/RESET	0
	Sets time and date in reports and controls the format in which CREATE SYNONYM creates date-time columns in a Master File.	STARTUP	RESET	0
	Sets time and date in reports and controls the format in which CREATE SYNONYM creates date-time columns in a Master File.	STARTUP	CURRENT	0
	Sets time and date in reports and controls the format in which CREATE SYNONYM creates date-time columns in a Master File.	STARTUP	CURRENT/NOW	0
	Sets time and date in reports and controls the format in which CREATE SYNONYM creates date-time columns in a Master File.	STARTUP	NOW	0
DBACSENSITIV	Controls whether password validation is case-sensitive.	OFF	OFF	0
	Controls whether password validation is case-sensitive.	OFF	ON	0
DBAJOIN	Controls how to add VALUE restriction as JOIN condition or to TABLE request.	OFF	OFF	1
	Controls how to add VALUE restriction as JOIN condition or to TABLE request.	OFF	ON	0
DBASOURCE	Controls the source of access restrictions in a multi-file structure.	HOST	HOST	0
	Controls the source of access restrictions in a multi-file structure.	HOST	ALL	0

SYSSQLOP: Reporting on Function Information

The syssqlop synonym retrieves information about functions, their descriptions, parameters, syntax, and adapter category.

Example: Retrieving Function Descriptions and Syntax

The following request retrieves the names, descriptions, and syntax for the legacy functions whose names begin with the letters A and B.

```
TABLE FILE SYSSQLOP
SUM FUNCTION_DESC FUNCTION_SYNTAX
BY FUNCTION
WHERE CATEGORY LIKE 'L%'
WHERE FUNCTION LE 'C'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Function name	Function description	Function syntax
ARGLEN	Returns the length of a character string, excluding trailing blanks. Use CHAR_LENGTH instead.	ARGLEN(length, string, output_format)
ATODBL	Converts a character string to double-precision format	ATODBL(string, length, output_format)
AYMDI	Add or subtract days to or from a date	AYMDI(arg1, arg2, arg3, arg4)
AYMI	Add or subtract months to or from dates	AYMI(arg1, arg2, arg3, arg4)
BAR	Produce a bar chart	BAR(barlength, infield, maxvalue, 'char', output_format)
BITSON	Determine if a bit is ON or OFF	BITSON(bitnumber, string, output_format)
BITVAL	Evaluate a bit string as a binary integer	BITVAL(string, startbit, number, output_format)
BNYSRC	Get the number of FOR value	BNYSRC(arg1)
BYTVAL	Translates a character to a decimal value	BYTVAL(character, output_format)

SYSTABLE: Reporting on Table Information

The systable synonym retrieves information about synonyms in your path, including type of synonym, creator, number of columns, keys, record length, and description.

Example: Retrieving A List of FMI Synonyms

The following request retrieves the names, descriptions, and attributes of the system table synonyms.

```
TABLE FILE SYSTABLE
PRINT TBTYPE REMARKS COLCOUNT RECLENGTH KEYCOLUMNS
BY NAME
WHERE NAME LIKE 'sys%'
WHERE TBTYPE EQ 'FMI'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

NAME	TBTYPE	REMARKS	COLCOUNT	RECLENGTH	KEYCOLUMNS
sysacfg	FMI	Metadata: Server admin information	14	2282	0
sysapps	FMI	Metadata: Applications and Files Information	123	51307	0
syscnv	FMI	SQL Adapter: Type Conversion Information	8	846	0
syscolum	FMI	Metadata: Column Information	121	8376	0
syscube	FMI	Metadata: Cube Information - Hierarchical	66	8763	0
syscube2	FMI	Metadata: Cube Information - Parent/Child	29	2478	0
sysdeffn	FMI	Metadata: DEFINE FUNCTION Information	17	3738	0
sysdirs	FMI	Metadata: File System Directory and File Information	17	1361	0
sysentty	FMI	Metadata: Entity Information - OBSOLETE	10	1100	0
syserr	FMI	Metadata: Error Files Information	12	826	0
sysfiles	FMI	Metadata: Directory information	17	774	0
sysfkeys	FMI	Metadata: Foreign Key Information	11	324	0
sysflow	FMI	Metadata: FOCEXEC Flow Information	91	9997	0
sysimp	FMI	Metadata For FMI Impact	31	3515	0
sysindex	FMI	Metadata: Index Information	42	986	0
syskeys	FMI	Metadata: Key Information	11	630	0
sysrpdir	FMI	Metadata: Available Stored Procedures	2	22	0
sysscha	FMI	Scheduler: Agent Information	21	2846	0
syssche	FMI	Scheduler: Event Information	7	704	0
sysset	FMI	Metadata: SET Information	34	2779	0
syssqlop	FMI	Functions Optimization Information for SQL Adapters	23	4090	0
systable	FMI	Metadata: Table Information	40	1149	0

Reporting on Data Types

The sysvdtp synonym retrieves data types and their corresponding USAGE and ACTUAL formats for the SQL Adapters and for fixed sequential data sources. It is not an FMI synonym, but retrieves the data type information from a delimited sequential file.

Note: In order to access the data file containing the data types for each adapter, you must allocate DDNAME SYSVDTP to the SYSVDTP member of your FUSELIB.DATA data set.

```
DYNAM ALLOC DD SYSVDTP DA hlq.FUSELIB.DATA MEMBER SYSVDTP SHR REU
```

Example: Retrieving Data Types for the Adapter for MySQL

The following request retrieves data type information for the Adapter for MySQL

```
TABLE FILE SYSVDTP
PRINT DATA_TYPE_CATEGORY
VENDOR_DATA_TYPES
TYPE_RANGE
SERVER_USAGE_CATEGORY
SERVER_USAGE
SERVER_ACTUAL
REMARKS
BY ADAPTER
WHERE SUFFIX EQ 'SQLMYSQL'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Adapter	Data Type Category	Vendor Data Types	Type Range	Server Data Type	Server USAGE	Server ACTUAL	Remarks
MySQL	Date-Time	DATE		Date-Time	YYMD	DATE	
	Date-Time	TIME		Date-Time	HHIS	HHIS	
	Date-Time	TIMESTAMP/DATETIME		Date-Time	HYYMDS	HYYMDS	
	Date-Time	TIMESTAMP WITH TIME ZONE			N/A	N/A	
	Date-Time	YEAR		Numeric	I6	I4	
	Numeric	TINYINT/SMALLINT		Numeric	I6	I2	
	Numeric	TINYINT(1)/BIT/BOOL		Numeric	I11	I2	
	Numeric	INTEGER/MEDIUMINT		Numeric	I11	I4	
	Numeric	INTEGER UNSIGNED		Numeric	P11	P8	
	Numeric	BIGINT		Numeric	P20	P11	
	Numeric	DECIMAL(p,s)/NUMERIC(p,s)	p=1..31,s=0	Numeric	Pn	Pk	n=p+1,k=(p/2)+1
	Numeric	DECIMAL(p,s)/NUMERIC(p,s)	p=1..31,s>0	Numeric	Pn.m	Pk	n=p+2,m=min(s,31),k=(p/2)+1
	Numeric	DECIMAL(p,s)/NUMERIC(p,s)	p>32,s=0	Numeric	P32	P16	
	Numeric	DECIMAL(p,s)/NUMERIC(p,s)	p>32,s>0	Numeric	P33.m	P16	m=min(s,31)
	Numeric	REAL/FLOAT/DOUBLE PRECISION		Numeric	D20.2	D8	
	LOB and Other	TEXT/MEDIUMTEXT/LONGTEXT		Text	TX50	TX	
	LOB and Other	TINYTEXT		Alphanumeric	A255V	A255V	
	LOB and Other	BLOB/TINYBLOB/LONGBLOB		BLOB	BLOB	BLOB	
	LOB and Other	VARBINARY(n)	n>16383	BLOB	BLOB	BLOB	
	LOB and Other	ENUM		Alphanumeric	An	An	
	LOB and Other	SET		Alphanumeric	An	An	MySQL JDBC Connector version has to be 5.1.7 or higher
	LOB and Other	GEOMETRY		Text	TX50	TX	Server supports this type as read-only via spatial properties:
	LOB and Other			Text			- Master File: GEOGRAPHIC_ROLE=GEOMETRY_AREA;
	LOB and Other			Text			- Access File: SQL_FLD_OBJ_TYPE=OPAQUE,
	LOB and Other			Text			SQL_FLD_OBJ_PROP=GEOMETRY_SHAPE,
	LOB and Other			Text			SQL_FLD_OBJ_EXPR='DB_EXPR()'
	LOB and Other			Text			where DB_EXPR() are vary depends on DBMS
	Character	CHAR(n)		Alphanumeric	An	An	
	Character	VARCHAR(n)		Alphanumeric	AnV	AnV	n>32765 will be truncated
	Character	BINARY(n)		Alphanumeric	Am	Am	m=2*n
	Character	VARBINARY(n)	n<=16383	Alphanumeric	Am	Am	m=2*n
	Character, Unicode	CHAR(n)		Alphanumeric	An	An	
	Character, Unicode	VARCHAR(n)		Alphanumeric	AnV	AnV	n>32765 will be truncated

Chapter 5

Defining a Word Substitution

A LET substitution enables you to define a word to represent other words and phrases. By substituting words for phrases, you can reduce the typing necessary to enter requests (especially when entering phrases repeatedly) and make requests easier to understand.

In this chapter:

- The LET Command
- Variable Substitution
- Null Substitution
- Multiple-Line Substitution
- Recursive Substitution
- Using a LET Substitution in a COMPUTE or DEFINE Command
- Checking Current LET Substitutions
- Interactive LET Query: LET ECHO
- Clearing LET Substitutions
- Saving LET Substitutions in a File
- Assigning Phrases to Function Keys

The LET Command

The LET command enables you to represent a word or phrase with another word. This reduces the amount of typing necessary for issuing requests, and makes the requests easier to understand. A substitution is especially useful when you use the same phrase repeatedly. Note that you cannot use LET substitutions in Dialogue Manager commands, and substitutions cannot be used in a MODIFY or Maintain request.

The LET command has a short form and a long form. Use the short form for one or two LET definitions that fit on one line. Otherwise, use the long form.

When you define a word with LET then use that word in a request, the word is translated into the word or phrase it represents. The result is the same as if you entered the original word or phrase directly. You can substitute any phrase that you enter online unless you are entering a MODIFY request.

A LET substitution lasts until it is cleared or until the request terminates. To clear active LET substitutions, issue the LET CLEAR command. To use the same substitutions in many requests, place the LET commands in a stored procedure. If you want to save currently active LET substitutions, use the LET SAVE facility. These substitutions can then be executed later with one short command.

Syntax: How to Make a Substitution (Short Form)

```
LET word = phrase [;word = phrase...]
```

where:

word

Is a string of up to 80 characters with no embedded blanks.

phrase

Is a string of up to 256 characters, which can include embedded blanks. The phrase can also include other special characters, but semicolons and pound signs need special consideration. If the word you are defining appears in the phrase you are replacing, you must enclose it in single quotation marks.

More than one substitution can be defined on the same line by placing a semicolon between definitions.

Example: Making a Substitution (Short Form)

The LET command defines the word WORKREPORT as a substitute for the phrase TABLE FILE EMPLOYEE:

```
LET WORKREPORT = TABLE FILE EMPLOYEE
```

Issuing the following

```
WORKREPORT
PRINT LAST_NAME
END
```

results in this request:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME
END
```

The next command includes TABLE as both the word you are defining and as part of the phrase it is replacing. It is enclosed in single quotation marks in the phrase:

```
LET TABLE = 'TABLE' FILE EMPLOYEE
```

More than one word is defined in the following command. The definitions are separated by a semicolon:

```
LET WORKREPORT=TABLE FILE EMPLOYEE; PR=PRINT
```

Syntax: How to Make a Substitution (Long Form)

```
LET
word = phrase
.
.
.
END
```

where:

word

Is a string of up to 80 characters with no embedded blanks.

phrase

Is a string of up to 256 characters, and can include embedded blanks.

END

Is required to terminate the command.

As shown, LET and END must each be on a separate line.

As with the short form, you can define several words on one line by separating the definitions with a semicolon.

Example: Making a Single Substitution (Long Form)

The following example illustrates a single substitution.

```
LET
RIGHTNAME = 'STEVENS' OR 'SMITH' OR 'JONES' OR 'BANNING' OR 'MCCOY' OR
'MCKNIGHT'
END
```

Example: Making Multiple Substitutions (Long Form)

The following example illustrates substitutions that span more than one line. Notice that there is no semicolon after the definition PR = PRINT:

```
LET
WORKREPORT = TABLE FILE EMPLOYEE; PR = PRINT
RIGHTNAME  = 'STEVENS' OR 'SMITH' OR 'JONES'
END
```

Example: Defining Substitutions for Translation

Non-English speakers can use LET commands to translate a request into another language. For example, this request

```
TABLE FILE CAR
SUM AVE.RCOST OVER AVE.DCOST
BY CAR ACROSS COUNTRY
END
```

can be translated into French as:

```
CHARGER FICHIER CAR
SOMMER AVE.RCOST SUR AVE.DCOST
PAR CAR TRAVERS COUNTRY
FIN
```

Variable Substitution

Using the LET command, you can define a word that represents a variable phrase. A variable phrase contains placeholder symbols (carets) to indicate missing elements in the phrase. This allows you to give a phrase different meanings in different requests. Placeholders can be parts of words within phrases. They can also be used to represent system commands.

Placeholders can be numbered or unnumbered. If the placeholders are not numbered, then they are filled from left to right: the first word in the request after the LET-defined word fills the first placeholder, the second word fills the second placeholder, and so on to the last placeholder. If they are numbered, the placeholders are filled in numerical order. If you do not supply enough words to fill all the placeholders, the extra placeholders are null.

Example: Making a Variable Substitution

The command

```
LET UNDERSCORE = ON < > UNDER-LINE
```

contains one placeholder. After issuing this command, you can use the word UNDERSCORE in a request:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY EMP_ID BY HIRE_DATE
UNDERSCORE EMP_ID
END
```

The field name following the LET-defined word supplies the missing value to the placeholder. In the example, EMP_ID follows the defined word UNDERSCORE. This field name is inserted in the placeholder and translates UNDERSCORE EMP_ID as:

```
ON EMP_ID UNDER-LINE
```

Example: Making Multiple Variable Substitutions (Unnumbered)

Issuing the LET command

```
LET TESTNAME = WHERE LAST_NAME IS < > OR < > OR < >
```

and then including the following line in a request

```
TESTNAME 'MCKNIGHT' 'STEVENS' 'BLACKWOOD'
```

translates the line as:

```
WHERE LAST_NAME IS 'MCKNIGHT' OR 'STEVENS' OR 'BLACKWOOD'
```

Notice that the variable phrase needs no placeholder at the end, and could also be code as WHERE LAST_NAME IS <> OR <>. Once all the placeholders are filled, the rest of the definition follows. In this example, the words MCKNIGHT and STEVENS would fill the two placeholders. BLACKWOOD would be left over, so it would follow the variable phrase.

If you do not supply enough words to fill in all the placeholders, the extra placeholders are null. For example, issuing this LET command

```
LET TESTNAME = WHERE LAST_NAME IS < > OR < > OR
```

and then entering this command

```
TESTNAME 'MCCOY'
```

translates the statement into:

```
WHERE LAST_NAME IS 'MCCOY' OR OR
```

This statement is illegal and produces an error message.

Example: Making Multiple Variable Substitutions (Numbered)

The following LET command contains numbered placeholders:

```
LET TESTNAME = WHERE LAST_NAME IS <1> OR <2> OR <3>
```

Therefore, the following line

```
TESTNAME 'STEVENS' 'MCKNIGHT' 'BLACKWOOD'
```

is translated as follows:

```
WHERE LAST_NAME IS 'STEVENS' OR 'MCKNIGHT' OR 'BLACKWOOD'
```

If two placeholders have the same number, both placeholders are filled with the same word. For example, if you issue this LET command

```
LET RANGE = SUM MAX.<1> AND MIN.<1>
```

and this line

```
RANGE SALARY
```

the translated statement is:

```
SUM MAX.SALARY AND MIN.SALARY
```

Example: Making a Variable Substitution in a Phrase

Issuing the following LET command

```
LET BIGGEST = MAX.< >
```

and entering the line

```
WRITE BIGGEST SALARY
```

translates the statement as:

```
WRITE MAX.SALARY
```

Example: Defining a System Command

Each of the following LET commands define a system command:

```
LET ALFOC = TSO ALLOC F(< >) DA(< >.FOCUS) SHR
LET LISTMEM = TSO LISTDS < > MEMBERS
```

Null Substitution

With a null substitution, you can use more than one word to represent a phrase. By using more than one word in a request instead of a single word, you can make the request more readable.

You can define a null word using LET. A null word is ignored by the application.

Syntax: How to Define a Null Word

To define a null word, issue the command

```
LET word=;
```

Example: Defining a Null Word

This LET command defines DISPLAY as a null word:

```
LET
DISPLAY=;
AVESAL = SUM AVE.SALARY BY DEPARTMENT
END
```

In the following request, the word DISPLAY is used in the code DISPLAY AVESAL, for readability, to make clear that the request prints the value represented by AVESAL:

```
TABLE FILE EMPLOYEE
DISPLAY AVESAL
WHERE DEPARTMENT IS 'PRODUCTION'
END
```

The word DISPLAY is ignored and the request is translated as:

```
TABLE FILE EMPLOYEE
SUM AVE.SALARY BY DEPARTMENT
WHERE DEPARTMENT IS 'PRODUCTION'
END
```

Multiple-Line Substitution

Many commands, such as END, must appear on a separate line in a report request. To include such a command in a LET definition, place a number sign (#) and a space before the command to indicate a new line. This allows you to substitute one word for several lines of code.

Example: Making Multiple-Line Substitutions

This LET command uses the number sign and a space to indicate that a new line is required for the END command:

```
LET HOLDREP = ON TABLE HOLD # END
```

The following request

```
TABLE FILE EMPLOYEE
SUM AVE.GROSS BY EMP_ID BY PAY_DATE
HOLDREP
```

is translated as:

```
TABLE FILE EMPLOYEE
SUM AVE.GROSS BY EMP_ID BY PAY_DATE
ON TABLE HOLD
END
```

Recursive Substitution

Recursive substitution allows a phrase in one LET definition to contain a word defined in another LET definition. Recursive substitution can also be used to abbreviate long phrases within LET commands.

Example: Making a Recursive Substitution

In the following LET command

```
LET
TESTNAME=IF LAST_NAME IS RIGHTNAME
RIGHTNAME = STEVENS OR MCKNIGHT OR MCCOY
END
```

the word RIGHTNAME in the phrase in the first definition is defined in the second definition. (Note that the two phrases in the LET command could be reversed.) This LET command is equivalent to:

```
LET
TESTNAME = IF LAST_NAME IS STEVENS OR MCKNIGHT OR MCCOY
END
```

Example: Abbreviating a Long Phrase

Consider the following LET command, which illustrates recursive substitution:

```
LET
TESTNAME = STEVENS OR SMITH OR MCCOY OR CONT1
CONT1    = BANNING OR IRVING OR ROMANS OR CONT2
CONT2    = JONES OR BLACKWOOD
END
```

You can use TESTNAME in this request:

```
TABLE FILE EMPLOYEE
PRINT SALARY BY LAST_NAME
IF LAST_NAME IS TESTNAME
END
```

This is the equivalent of:

```
TABLE FILE EMPLOYEE
PRINT SALARY BY LAST_NAME
IF LAST_NAME IS STEVENS OR SMITH OR MCCOY OR
BANNING OR IRVING OR ROMANS
OR JONES OR BLACKWOOD
END
```

Using a LET Substitution in a COMPUTE or DEFINE Command

A semicolon must follow an expression in a COMPUTE or DEFINE command. To use a LET substitution in a DEFINE or COMPUTE, you must include two semicolons in the LET syntax. You cannot create a LET substitution for a phrase that contains a semicolon.

Example: Using a LET Substitution in a COMPUTE or DEFINE Command

The following LET syntax includes two semicolons, since the substitution will be made in a COMPUTE command:

```
LET
SALTEST = LEVEL/A4 = IF SALARY GT 35000 THEN HIGH
ELSE LOW;;
END
```

Issuing the command

```
AND COMPUTE SALTEST
```

translates the line into

```
AND COMPUTE LEVEL/A4 = IF SALARY GT 35000 THEN HIGH
ELSE LOW;
```

with one semicolon after the word LOW, as required by the expression in the COMPUTE.

Checking Current LET Substitutions

The ? LET command displays the currently active LET substitutions.

Syntax: How to Check Current LET Substitutions

```
? LET [word1 word2 ... wordn]
```

where:

word1 word 2...wordn

Are the LET-defined words you want to check. If you omit these parameters, ? LET displays a two-column list of all active LET substitutions. The left column contains the LET-defined words; the right column contains the phrases the words represent.

Example: Checking Selected LET Substitutions

Issuing

```
? LET CHART TESTNAME RIGHTNAME
```

displays a two-column list of the LET substitutions for CHART, TESTNAME, and RIGHTNAME.

Example: Checking All Current LET Substitutions

Issuing

```
? LET
```

displays a list of all current LET substitutions.

Interactive LET Query: LET ECHO

The LET ECHO facility shows how FOCUS interprets FOCUS statements. This facility is a diagnostic tool you can use when statements containing LET-defined words are not being interpreted the way you expect them to.

When the LET ECHO facility is activated, when you enter a FOCUS statement, LET ECHO displays the statement as interpreted by FOCUS.

Syntax: How to Activate the LET ECHO Facility

To activate the LET ECHO facility, issue the command:

```
LET ECHO
```

Syntax: How to Deactivate the LET ECHO Facility

```
ENDECHO
```

Reference: Results of LET ECHO Commands

The following explains the results of a LET ECHO command:

- If you enter a statement containing no LET-defined words, LET ECHO displays the statement as you entered it.
- If you enter a statement containing LET-defined words, LET ECHO displays the statement with the substitutions made.
- If the statement contains variable substitutions, LET ECHO displays the substitutions with the placeholders filled in.
- If the statement contains multiple-line substitutions, LET ECHO displays the statement with the substitutions on multiple lines.
- If the statement contains null substitutions, LET ECHO displays the statement with the LET-defined words deleted.
- If the statement contains recursive substitutions, the substitutions appear as they are finally resolved.
- LET ECHO may be coded as the first line of a FOCEXEC and ENDECHO as the last line.

Note: If you enter a statement containing a variable substitution, you must enter as many words after the LET-defined word as there are placeholders in the phrase; otherwise, LET ECHO will wait for additional input.

Clearing LET Substitutions

Use the LET CLEAR command to clear LET substitutions.

Syntax: How to Clear LET Substitutions

```
LET CLEAR {*|word1 [word2...wordn]}
```

where:

`*`

Clears all substitutions.

`word1...wordn`

Are the LET-defined words that you want to clear.

Example: Clearing LET Substitutions

Issuing the following command

```
LET CLEAR CHART TESTNAME RIGHTNAME
```

clears substitutions for CHART, TESTNAME, and RIGHTNAME. If there are no additional LET substitutions in effect, the following command would have the same effect:

```
LET CLEAR *
```

Saving LET Substitutions in a File

Since LET substitutions only last the duration of a session, saving them is helpful if you need the same substitutions for another request.

To save LET substitutions currently in effect, use the LET SAVE command.

Syntax: How to Save LET Substitutions

```
LET SAVE [filename]
```

where:

filename

Is the eight-character name of the file in which you want to save the substitutions. If you do not supply a file name, the default file name is LETSAVE.

Assigning Phrases to Function Keys

You can assign a phrase to a function key. Then when you have a blank line and press a function key, that phrase appears as if you actually typed it. This process works only in situations where the LET facility is operative.

Syntax: How to Assign a Phrase to a Function Key

```
LET !n= [.]phrase
```

where:

n

Is a function key number from 1 to 24.

.

Suppresses the echo of the phrase when you press the function key.

phrase

Is the phrase that the specified function key represents.

Example: Assigning Phrases to Function Keys

The following assigns values to function keys:

```
LET !4 = EX DAILYRPT
LET !6 = END
LET !20 = IF RECORDLIMIT EQ 10
LET !21 = .EX MYREPORT
```

Chapter 6

Enhancing Application Performance

This topic covers FOCUS facilities that are available across command environment boundaries. These facilities are easy to use and, in many cases, step-by-step instructions are provided.

In this chapter:

- FOCUS Facilities
- Loading a File
- Saving Master Files in Memory for Reuse
- Accessing a FOCUS Data Source

FOCUS Facilities

The FOCUS facilities discussed in this topic are classified as file utilities for FOCUS and external files. They are summarized in the following table:

Command	Description
`LOAD`	Loads FOCUS procedures and Master Files into memory (see *Loading a File* on page 385).
`MINIO`	Improves performance by reducing I/O operations when accessing FOCUS data sources (see *Accessing a FOCUS Data Source* on page 392).
`SET SAVEDMASTERS`	Improves performance by saving Master Files in memory.

Loading a File

Use the LOAD command to load the following types of files into memory for use within a FOCUS session:

- Master Files (MASTER).

- Access Files.
- FOCUS procedures (FOCEXEC).

Using memory-resident files decreases execution time because the files do not have to be read from the disk. Use the UNLOAD command to remove the files from memory.

The LOAD command loads unparsed Master Files into memory. To store parsed Master Files in memory, use the SET SAVEDMASTERS command described in *Saving Master Files in Memory for Reuse* on page 389.

Syntax: How to Load a File

```
LOAD filetype filename1... [filename2...]
```

where:

filetype

Specifies the type of file to be loaded (MASTER, FOCEXEC, or Access File). For a list of Access File Types, see *Considerations for Loading a Master File, FOCUS Procedure, or Access File* on page 387.

filename1...

Specifies one or more files to be loaded. Separate the file type and file name(s) with a space.

Example: Loading Multiple Files

The following command loads four FOCEXECs—CARTEST, FOCMAP1, FOCMAP2, and FOCMAP3—into memory:

```
>LOAD FOCEXEC CARTEST FOCMAP1 FOCMAP2 FOCMAP3
```

A subsequent reference to one of these files during the current FOCUS session will use the loaded, rather than the disk version.

Syntax: How to Unload a File

```
UNLOAD [*|filetype] [*| filename1... [filename2...] ]
```

where:

filetype

Specifies the type of file to be unloaded (MASTER, FOCEXEC, or Access File). For a list of Access File Types, see *Considerations for Loading a Master File, FOCUS Procedure, or Access File* on page 387.

To unload all files of all types, use an asterisk.

filename1...

Specifies one or more files to be unloaded. Separate the file type and file name(s) with a space. To unload all files of that file type, use an asterisk.

Example: Unloading Multiple Files

The following command unloads two memory-resident FOCEXECs— CARTEST and FOCMAP3:

```
>UNLOAD FOCEXEC CARTEST FOCMAP3
```

Any subsequent reference to one of these files will use the disk version.

Loading Master Files, FOCUS Procedures, and Access Files

Loading Master Files, Access Files, and FOCEXECs into memory eliminates the I/Os required to read each time they are referenced. Whenever FOCUS requires a Master File, Access File, or executes a FOCEXEC, it first looks for a memory-resident MASTER, Access File, or FOCEXEC file. If FOCUS cannot find the file in memory, it then searches for a disk version in the normal way.

Reference: Considerations for Loading a Master File, FOCUS Procedure, or Access File

The following are considerations for loading a Master File, FOCUS procedure, and Access File:

- If you load a Master File, Access File, or a FOCEXEC that has already been loaded into memory, the new copy replaces the old copy.
- Do not load a Master File, Access File, or a FOCEXEC that you are developing because FOCUS will always use the memory-resident copy of the file (until you reload it), rather than the one you are developing. The copy that you are developing on TED or your system editor is the disk copy, not the memory-resident copy.
- A loaded Master File, Access File, or FOCEXEC requires a maximum of 80 bytes of memory for each of its records plus a small amount of control information, rounded up to a multiple of 4200 bytes.

- The following are the file types for the various Access Files:

Access File	File Type
ADABAS	FOCADBS
CA-DATACOM	FOCDTCM
DB2	FOCSQL
FOCUS	ACCESS
CA-IDMS	FOCIDMS
IDMS/SQL	FOCSQL
IMS/DB	ACCESS
Model 204	ACCESS
ORACLE	FOCSQL
TERADATA	FOCDBC

Displaying Information About Loaded Files

The ? LOAD command displays the file type, file name, and resident size of currently loaded files.

Syntax: How to Display Information About Loaded Files

```
? LOAD [filetype]
```

where:

filetype

Specifies the type of file (MASTER, Access File, FOCEXEC, or MODIFY) on which information will be displayed. For a list of Access File Types, see *Considerations for Loading a Master File, FOCUS Procedure, or Access File* on page 387.

To display information on all memory-resident files, omit the file type.

Example: Displaying Information About Loaded Files

Issuing the command

```
? LOAD
```

produces information similar to the following:

```
FILES CURRENTLY LOADED

   CAR         MASTER   4200   BYTES
   EXPERSON    MASTER   4200   BYTES
   CARTEST     FOCEXEC  8400   BYTES
```

Saving Master Files in Memory for Reuse

You can save up to 99 Master Files in memory after they have been used in a request. The saved Master Files are not re-parsed when referenced in subsequent requests, resulting in a significant performance improvement. The greatest improvement occurs in Master Files with a great many fields, where parsing is slowest.

Saving Master Files in memory is particularly helpful when running multiple requests against several Master Files. The most recently used Master File is stored in memory regardless of this setting. With each request that specifies a new Master File, the prior Master File is moved down on the saved list and the new Master File is placed at the top of the list. Once all of the slots on the list are full, parsing a new Master File causes the one at the bottom to drop off the list. If an already saved Master File is used in a request, it moves to the top of the list.

Only one occurrence of a Master File name is maintained on the list. Therefore, if you use an already saved Master File as the host file in a JOIN, in a HOLD command (with the same AS name), in a USE...AS command, or in a COMBINE command without specifying a unique name, the new version of the Master File replaces the previous version on the list. A JOIN CLEAR or USE CLEAR command purges the parsed Master File from memory.

If a Master File will be re-parsed multiple times, you can save the I/O needed to retrieve it from disk by loading it into memory using the LOAD command described in *Loading a File* on page 385.

Note: SAVEDMASTERS is not an effective technique to use with massive amounts of data because the amount of time saved by not re-parsing is small in comparison to the time for processing the data.

Syntax: How to Save Parsed Master Files in Memory

```
SET SAVEDMASTERS = n
```

where:

n

Is an integer between 0 and 99 that specifies the maximum number of Master Files on the SAVEDMASTERS list. The default value is 0. Note that the most recently used Master File is always stored in memory, even with SAVEDMASTERS set to zero. However, the zero setting does not generate the list of saved Master Files.

Syntax: How to Query the SAVEDMASTERS Setting

The following query command indicates the number of Master Files allowed on the list of saved Master Files and lists the names of the Master Files on the list.

```
? SET SAVEDMASTERS
```

Example: Saving and Querying Parsed Master Files

The following command specifies that up to three parsed Master Files can be saved:

```
SET SAVEDMASTERS = 3
```

Issue the Query command:

```
? SET SAVEDMASTERS
```

The output of the query command indicates that the list can contain up to three Master Files, but none are currently saved:

```
SAVEDMASTERS               3
```

The following procedure parses two Master Files, EMPLOYEE and MOVIES:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME BY EMP_ID
END
-RUN
TABLE FILE MOVIES
PRINT TITLE BY DIRECTOR
END
-RUN

? SET SAVEDMASTERS
```

In this example, the output of the query command indicates that the list can contain up to three Master Files and that the list currently consists of MOVIES and EMPLOYEE:

```
SAVEDMASTERS                3

  MOVIES
  EMPLOYEE
```

Reference: **Usage Notes for SET SAVEDMASTERS**

- Memory resources are used to store the parsed Master Files, reducing the amount of memory available for other processes.
- You cannot selectively purge Master Files from the list.
- The SAVEDMASTERS parameter is not supported in a request (ON TABLE SET) or in FOCPARM.
- The SAVEDMASTERS setting is not supported on a FOCUS Database Server or with a Maintain procedure.
- The SAVEDMASTERS setting is not supported with SCAN or FSCAN.
- Issuing the CHECK FILE or REBUILD command causes the specified Master File to be re-parsed.
- The ?F and ?FF commands only re-parse the Master File when issued outside of a request for a Master File other than the most recently used Master File.
- Using an alternate file view (TABLE FILE filename.fieldname) or the AUTOPATH=ON setting re-parses the Master File.
- If the SAVEDMASTERS value is changed between requests:
  - Raising the number allows more Master Files to be saved as they are parsed.
  - Lowering the number drops the oldest saved Master Files.
- If changes are made to a Master File that is saved, the changes will not be implemented until the Master File is re-parsed.
- When only one Master File has been used, it is not placed on the SAVEDMASTERS list.
- DEFINE expressions are not stored and, therefore, are re-parsed every time they are used.
- Creating a HOLD file erases the Master File name from the list if it is there, and the HOLD command does not place the new Master File on the list.
- SAVEDMASTERS is most effective when a Master File has a lot of fields.

- The FML Hierarchy LOAD CHART command does not add the Master File to the SAVEDMASTERS list.

Accessing a FOCUS Data Source

MINIO is a new I/O buffering technique that improves performance by reducing I/O operations when accessing FOCUS data sources. With MINIO set on, no block is ever read more than once, and therefore the number of reads performed is the same as the number of tracks present. This results in an overall reduction in elapsed times when reading and writing.

With FOCUS data sources that are not disorganized, MINIO can greatly reduce the number of I/O operations for TABLE and MODIFY commands. I/O reductions of up to 50% are achievable with MINIO. The actual reduction varies depending on data source structure and average numbers of children segments per parent segment. By reducing I/O operations, elapsed times for TABLE and MODIFY commands also drop.

Syntax: How to Set MINIO

```
SET MINIO = {ON|OFF}
```

where:

ON

Does not read a block more than once; the number of reads performed will be the same as the number of tracks present. This results in an overall reduction in elapsed times when reading and writing. This value is the default.

OFF

Disables MINIO.

Using MINIO

MINIO reduces CPU time slightly while slightly raising memory utilization. MINIO requires one track I/O buffer per referenced segment type. Between 40K and 48K of above-the-line virtual memory is needed per referenced segment.

When MINIO is enabled, FOCUS decides for each command whether or not to employ it, and which data sources to use it with. It is possible in executing a single command referencing several data sources that MINIO might be used for some but not for others. Data sources accessed via indexes, or physically disordered through online updates, are not candidates for MINIO buffering. Physical disorganization, in this case, means that the sequence of selected records jumps all over the data source, as opposed to progressing steadily forward. When disorganization occurs, MINIO abandons its buffering techniques and resorts to the standard I/O methodology.

When reading data sources, MINIO is used with TABLE, TABLEF, GRAPH, MATCH and during the DUMP phase of the REBUILD command, provided the target data source is not accessed via an index or is physically disorganized.

When writing to data sources, MINIO is used with MODIFY but never with MAINTAIN, provided there is no CRTFORM or COMMIT subcommand. CRTFORMs indicate online transaction processing, which requires that completed transactions be written out to the data source. COMMITs are explicit orders to do so. These events are incompatible with MINIO minimization logic and therefore rule out its use.

As with reads, using MINIO with MODIFY also requires that a data source be accessed sequentially. Attempts to access an index, or to update physically disorganized data sources can cause MINIO to be disabled. In addition, frequent repositioning to previously accessed records, even within well-organized data sources, will cause MINIO to be disabled.

Determining If a Previous Command Used MINIO

The ? STAT command is used to determine whether the previous data source access command employed MINIO.

Syntax: How to Determine If a Previous Command Used MINIO

To determine if a previous command used MINIO, issue the command:

```
? STAT
```

Example: Determining If a Previous Command Used MINIO

Typing ? STAT generates a screen similar to the following:

```
                    STATISTICS OF LAST COMMAND

RECORDS        =           0       SEGS CHNGD     =           0
LINES          =           0       SEGS DELTD     =           0
BASEIO         =          87       NOMATCH        =           0
TRACKIO        =          16       DUPLICATES     =           0
SORTIO         =           0       FORMAT ERRORS  =           0
SORT PAGES     =           0       INVALID CONDTS =           0
READS          =           1       OTHER REJECTS  =           0
TRANSACTIONS   =        1500       CACHE READS    =           0
ACCEPTED       =        1500       MERGES         =           0
SEGS INPUT     =        1500       SORT STRINGS   =           0

INTERNAL MATRIX CREATED: YES       AUTOINDEX USED:          NO
SORT USED:             FOCUS       AUTOPATH USED:           NO
MINIO USED:              YES
```

In the preceding example MINIO USED is displayed as YES. It may also display NO or DISABLED.

- YES means that MINIO buffering has taken place reducing the number of tracks read/written to the FOCUS data source.
- NO means that MINIO buffering has not taken place.
- DISABLED means that MINIO buffering was started but terminated as no performance gains could be made. This does not mean that the command did not complete successfully. It only indicates that MINIO buffering began and ended during the read/write.

Reference: Restrictions for Using MINIO

Note the following restrictions when you are using the MINIO command:

- When MINIO is used with MODIFY, all CHECK subcommands are ignored. If a MODIFY command terminates abnormally, the condition of the data source is unpredictable, and it should be restored from a backup copy and the update repeated. Since MINIO is designed to minimize I/O during large data source loads and updates, it has no checkpoint or restart facility. If this is unacceptable, set MINIO off.
- MINIO is not used to access data sources through FOCUS Database Servers (formerly called sink machines) or HLI programs.
- MINIO requires the presence of the TRACKIO feature. Meaning, TRACKIO must be set to ON which is the default setting. If TRACKIO is set to OFF, then MINIO is deactivated.
- MINIO buffering starts when the FOCUS data source exceeds 64 pages in size. If this size is never reached, MINIO is never activated.

- If the file being modified UPDATEs, INCLUDEs, or DELETEs a field that is indexed, MINIO is disabled. In other words, FIELDTYPE=I or INDEX=I is coded in the Master File for this field.
- CRTFORM and COMMIT commands disable MINIO.
- MAINTAIN procedures will not use MINIO buffering techniques.
- MINIO is not enabled if the data source is physically disorganized by transaction processing.

Chapter 7

Working With Cross-Century Dates

Many existing business applications use two digits to designate a year, instead of four digits. When they receive a value for a year, such as 00, they typically interpret it as 1900, assuming that the first two digits are 19, for the twentieth century. These applications require a way to handle dates when the century changes (for example, from the twentieth to the twenty-first), or when they need to perform comparisons or arithmetic on dates that span more than one century.

The cross-century date feature described in this topic enables the correct interpretation of the century if it is not explicitly provided, or is assumed to be the twentieth. The feature is application-based, that is, it involves modifications to procedures or metadata so that dates are accurately interpreted and processed. The feature is called the sliding window technique.

In this chapter:

- When Do You Use the Sliding Window Technique?
- The Sliding Window Technique
- Applying the Sliding Window Technique
- Defining a Global Window With SET
- Defining a Dynamic Global Window With SET
- Querying the Current Global Value of DEFCENT and YRTHRESH
- Defining a File-Level or Field-Level Window in a Master File
- Defining a Window for a Virtual Field
- Defining a Window for a Calculated Value
- Additional Support for Cross-Century Dates

When Do You Use the Sliding Window Technique?

If your application accesses dates that contain an explicit century, the century is accepted as is. Your application can run correctly across centuries, and you do not need to use the sliding window technique.

If your application accesses dates without explicit centuries, they assume the default value 19. Your application will require remediation, such as the sliding window technique, to ensure the correct interpretation of the century if the default is not valid, and to run as expected in the next century.

This topic does not cover remediation options such as date expansion, which requires that data be changed in the data source to accommodate explicit century values. For a list of Information Builders documentation on remediation, see your latest *Publications Catalog*.

This topic covers the use of the sliding window technique in reporting applications. Details on when to use the sliding window technique are provided later in this topic. It also includes reference information on the use of the technique with FOCUS MODIFY requests. For additional information on implementing this technique with Maintain, see your database maintenance documentation the *Maintaining Databases* manual. References to MODIFY and Maintain apply only to Developer Studio.

The Sliding Window Technique

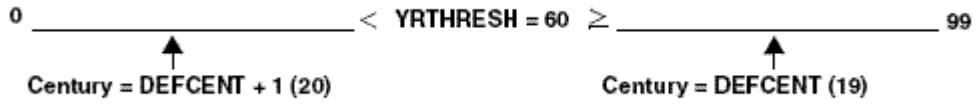
With the sliding window technique, you do not need to change stored data from a 2-digit year format to a 4-digit year format in order to determine the century. Instead, you can continue storing 2-digit years and expand them when accessed.

The sliding window technique recognizes that the earliest and latest values for a single date field in most business applications are within 100 years of one another. For example, a human resources application typically contains a field for the birth date of each active employee. The difference in the birth date (or age) of the oldest active employee and the youngest active employee is not likely to be more than 100.

The technique is implemented as follows:

- You define the start of a 100-year sliding window by supplying two values: one for the default century (DEFCENT) and one for the year threshold (YRTHRESH). For example, a value of 19 for the century, combined with a value of 60 for the threshold, creates a window that starts in 1960 and ends in 2059.
- The threshold provides a way to assign a value to the century of a 2-digit year:
  - A year greater than or equal to the threshold assumes the value of the default century (DEFCENT). Using the sample value 19 for the default century and 60 for the threshold, a 2-digit year of 70 is interpreted as 1970 (70 is greater than 60).
  - A year less than the threshold assumes the value of the default century plus 1 (DEFCENT + 1). Using the same sample values (19 and 60), a 2-digit year of 50 is interpreted as 2050 (50 is less than 60), and a 2-digit year of 00 is interpreted as 2000 (00 is also less than 60).

The conversion rule for this example is illustrated as follows:



Any 2-digit year is assumed to fall within the window. You must handle dates that fall outside the defined window by coding.

Each file or each date field used in an application can have its own conversion rule, which provides the flexibility required by most applications.

Defining a Sliding Window

You can define a sliding window in several ways, depending on the specific requirements of your application:

- **Globally.** The SET DEFCENT and SET YRTHRESH commands define a window on a global level.
- **On a file level.** The FDEFCENT and FYRTHRESH attributes in a Master File define a window on a file level, allowing the correct interpretation of date fields from multiple files that span different time periods.
- **On a field level.** The DEFCENT and YRTHRESH attributes in a Master File define a window on a field level, allowing the correct interpretation of date fields, within a single file, that span different time periods.
- **For a virtual field.** The DEFCENT and YRTHRESH parameters on a DEFINE command, in either a request or a Master File, define a window for a virtual field.
- **For a calculated value.** The DEFCENT and YRTHRESH parameters on a COMPUTE command define a window for a calculated value.

If you define more than one window using any of the preceding methods, the precedence is as follows:

1. DEFCENT and YRTHRESH on a DEFINE or COMPUTE command.
2. DEFCENT and YRTHRESH field-level attributes in a Master File.
3. FDEFCENT and FYRTHRESH file-level attributes in a Master File.

4. SET DEFCENT and SET YRTHRESH on a global level; if you do not specify values, the defaults are used (DEFCENT = 19, YRTHRESH = 0).

Creating a Dynamic Window Based on the Current Year

An optional feature of the sliding window technique enables you to create a dynamic window, defining the start of a 100-year span based on the current year. The start year and threshold for the window automatically change at the beginning of each new year.

If an application requires that a window's start year change when a new year begins, use of this feature avoids the necessity of manually re-coding it.

To implement this feature, YRTHRESH or FYRTHRESH is offset from the current year, or given a negative value.

For example, if the current year is 1999 and YRTHRESH is set to -38, a window from 1961 to 2060 is created. The start year 1961 is derived by subtracting 38 (the value of YRTHRESH) from 1999 (the current year). To interpret dates that fall within this window, the threshold 61 is used.

At the beginning of the year 2000, a new window from 1962 to 2061 is automatically created; for dates that fall within this window, the threshold 62 is used. In the year 2001, the window becomes 1963 to 2062, and the threshold is 63, and so on.

With each new year, the start year for the window is incremented by one.

When using this feature, do not code a value for DEFCENT or FDEFCENT, since the feature is designed to automatically calculate the value for the default century. Be aware of the following:

- If you do code a value for DEFCENT on the field level in a Master File, or for FDEFCENT on the file level in a Master File, the feature will not work as intended. The value for the century, which is automatically calculated by YRTHRESH by design, will be reset to the value you code for DEFCENT or FDEFCENT.
- If you code a value for DEFCENT anywhere other than the field level in a Master File (for example, on the global level), and YRTHRESH is negative, the coded value will be ignored. The default century will be automatically calculated as designed.

Applying the Sliding Window Technique

To apply the sliding window technique correctly, you need to understand the difference between a date format (formerly called a smart date) and a legacy date:

- A date format refers to an internally stored integer that represents the number of days between a real date value and a base date (either December 31, 1900, for dates with YMD or YYMD format; or January 1901, for dates with YM, YYM, YQ, or YYQ format). A Master File does not specify a data type or length for a date format; instead, it specifies display options such as D (day), M (month), Y (2-digit year), or YY (4-digit year). For example, MDYY in the USAGE (also known as FORMAT) attribute of a Master File is a date format. A real date value such as March 5, 1999, displays as 03/05/1999, and is internally stored as the offset from December 31, 1900.
- A legacy date refers to an integer, packed decimal, double precision, floating point, or alphanumeric format with date edit options, such as I6YMD, A6MDY, I8YYMD, or A8MDYY. For example, A6MDY is a 6-byte alphanumeric string; the suffix MDY indicates how Information Builders will return the data in the field. The sample value 030599 displays as 03/05/99.

For details on date fields, see the *Describing Data* manual.

When to Supply Settings for DEFCENT and YRTHRESH

The rest of this topic refers simply to DEFCENT when either DEFCENT or FDEFCENT applies, and to YRTHRESH when either YRTHRESH or FYRTHRESH applies.

Supply settings for DEFCENT and YRTHRESH in the following cases:

- When you issue a DEFINE or COMPUTE command to convert a legacy date without century digits to a date format with century digits (for example, to convert the format I6YMD to YYMD). With DEFINE and COMPUTE, DEFCENT and YRTHRESH do not work directly on legacy dates; for example, you cannot use them to convert the legacy date format I6YMD to the legacy date format I8YYMD.
- When a DEFINE command, COMPUTE command, or Dialogue Manager -SET command calls a function, supplied by Information Builders, that uses legacy dates, the input date does not contain century digits.

On input, the function will use the window defined for an 16 legacy date field (with edit options). The output format may be 18 (again, with edit options), which includes a 4-digit year.

- When data is entered or changed in a date format field in a FOCUS data source, or a SQL date is entered or changed in a Relational Database Management System (RDBMS), and the input date does not contain century digits.

  For example, you can use the sliding window technique in applications that use FIXFORM or CRTFORM with MODFIY.

- When a data source is read, and the ACTUAL attribute in the Master File is non-date specific (for example, A6, I6, or P6), without century digits, and the FORMAT or USAGE attribute specifies a date format. This case does not apply to FOCUS data sources.

Follow these rules when implementing the sliding window technique:

- Specify values for both DEFCENT and YRTHRESH to ensure consistent coding and accurate results, except when YRTHRESH has a negative value. In that case, specify a value for YRTHRESH only; do not code a value for DEFCENT.
- Do not use DEFCENT and YRTHRESH with ON TABLE SET.

Finally, keep in mind that the sliding window technique does not change the way existing data is stored. Rather, it accurately interprets data during application processing.

Date Validation

Date formats are validated on input. For example, 11/99/1999 is rejected as input to a date field formatted as MDYY, because 99 is not a valid day. Information Builders generates an error message.

Legacy dates are not validated. The date 11991999, described with the format A8MDYY, is accepted, even though it, too, contains the invalid day 99.

Defining a Global Window With SET

The SET DEFCENT and SET YRTHRESH commands define a window on a global level. The time span created by the SET commands applies to every 2-digit year used by the application unless you specify file-level or field-level windows elsewhere.

For details on specifying parameters that govern the environment, see *Customizing Your Environment*.

Syntax: How to Define a Global Window With SET

To define a global window, issue two SET commands.

The first command is

```
SET DEFCENT = {cc|19}
```

where:

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

The second command is

```
SET YRTHRESH = {[-]yy|0}
```

where:

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of FDEFCENT for the century. Two-digit years less than the threshold assume the value of FDEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and FDEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

Example: Defining a Global Window With SET

In the following request, the SET command defines a global window from 1983 to 2082.

As SET syntax allows, the command is entered on one line, with the parameters separated by a comma. You do not need to repeat the keyword SET for YRTHRESH.

The DEFINE command converts the legacy date EFFECT_DATE into the date format NEW_DATE. It creates NEW_DATE as a virtual field, derived from the existing field EFFECT_DATE. The format of EFFECT_DATE is I6YMD, which is a 2-digit year. NEW_DATE is formatted as YYMD, which is a 4-digit year. For details on DEFINE, see the *Creating Reports* manual.

The request is:

```
SET DEFCENT = 19, YRTHRESH = 83

DEFINE FILE EMPLOYEE
NEW_DATE/YYMD = EFFECT_DATE;
END

TABLE FILE EMPLOYEE
PRINT EFFECT_DATE NEW_DATE BY EMP_ID
END
```

In the report, the value of the 2-digit year 82 is less than the threshold 83, so it assumes the value 20 for the century (DEFCENT + 1) and is returned as 2082 in the NEW_DATE column. The other year values (83 and 84) are greater than or equal to the threshold 83, so the century defaults to the value 19 (DEFCENT); they are returned as 1983 and 1984 under NEW_DATE.

The output is:

```
PAGE     1

EMP_ID     EFFECT_DATE   NEW_DATE
------     -----------   --------
071382660
112847612
117593129     82/11/01   2082/11/01
119265415
119329144     83/01/01   1983/01/01
123764317     83/03/01   1983/03/01
126724188
219984371
326179357     82/12/01   2082/12/01
451123478     84/09/01   1984/09/01
543729165
818692173     83/05/01   1983/05/01
```

In the example, missing date values appear as blanks by default. To retrieve the base date value for the NEW_DATE field instead of blanks, issue the command

```
SET DATEDISPLAY = ON
```

before running the request.

The base date value for NEW_DATE, which is formatted as YYMD, is returned as 1900/12/31:

```
PAGE     1

EMP_ID     EFFECT_DATE   NEW_DATE
------     -----------   --------
071382660                1900/12/31
112847612                1900/12/31
117593129     82/11/01   2082/11/01
119265415                1900/12/31
119329144     83/01/01   1983/01/01
123764317     83/03/01   1983/03/01
126724188                1900/12/31
219984371                1900/12/31
326179357     82/12/01   2082/12/01
451123478     84/09/01   1984/09/01
543729165                1900/12/31
818692173     83/05/01   1983/05/01
```

If NEW_DATE had a YYM format, the base date would appear as 1901/01. If it had a YYQ format, it would appear as 1901 Q1.

If the value of NEW_DATE is 0 and SET DATEDISPLAY = OFF (the default), blanks are displayed. With SET DATEDISPLAY = ON, the base date is displayed instead of blanks. Zero (0) is treated as an offset from the base date, which results in the base date.

For details on SET DATEDISPLAY, see *Customizing Your Environment*.

Defining a Dynamic Global Window With SET

This topic illustrates the creation of a dynamic window using the global command SET YRTHRESH. You can also implement this feature on the file and field level, and on a DEFINE or COMPUTE.

With this option of the sliding window technique, the start year and threshold for the window automatically changes at the beginning of each new year. The default century (DEFCENT) is automatically calculated.

You can use SET TESTDATE to alter the system date when testing a dynamic window (that is, when YRTHRESH has a negative value). However, when testing a dynamic window defined in a Master File, you must issue a CHECK FILE command each time you issue a SET TESTDATE command. CHECK FILE reloads the Master File into memory and ensures the correct recalculation of the start date of the dynamic window. For details on SET TESTDATE, see your documentation on the SET command. For details on CHECK FILE, see the *Describing Data* manual.

Example: Defining a Dynamic Global Window With SET

In the following request, the COMPUTE command calls the function AYMD, supplied by Information Builders. AYMD adds one day to the input field, HIRE_DATE; the output field, HIRE_DATE_PLUS_ONE, contains the result. HIRE_DATE is formatted as I6YMD, which is a legacy date with a 2-digit year. HIRE_DATE_PLUS_ONE is formatted as I8YYMD, which is a legacy date with a 4-digit year.

The function uses the YRTHRESH value set at the beginning of the request to create a dynamic window for the input field HIRE_DATE. The start date of the window is incremented by one at the beginning of each new year. Notice that DEFCENT is not coded, since the default century is automatically calculated whenever YRTHRESH has a negative value.

The function inputs a 2-digit year, which is windowed. It then outputs a 4-digit year that includes the century digits.

Sample values are shown in the reports for 1999, 2000, and 2018, which follow the request.

For details on AYMD, see the *Using Functions* manual.

The request is:

```
SET YRTHRESH = -18

TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
      HIRE_DATE_PLUS_ONE/I8YYMD = AYMD
      (HIRE_DATE, 1, HIRE_DATE_PLUS_ONE);
END
```

In 1999, the window spans the years 1981 to 2080. The threshold is 81 (1999 - 18). In the report, the 2-digit year 80 is less than the threshold 81, so it assumes the value 20 for the century (DEFCENT + 1), and is returned as 2080 in the HIRE_DATE_PLUS_ONE column. The other year values (81 and 82) are greater than or equal to the threshold 81, so the century defaults to the value of DEFCENT (19); they are returned as 1981 and 1982.

The output is:

```
PAGE     1

HIRE_DATE   HIRE_DATE_PLUS_ONE
---------   ------------------
 80/06/02           2080/06/03
 81/07/01           1981/07/02
 82/05/01           1982/05/02
 82/01/04           1982/01/05
 82/08/01           1982/08/02
 82/01/04           1982/01/05
 82/07/01           1982/07/02
 81/07/01           1981/07/02
 82/04/01           1982/04/02
 82/02/02           1982/02/03
 82/04/01           1982/04/02
 81/11/02           1981/11/03
```

In 2000, the window spans the years 1982 to 2081. The threshold is 82 (2000 - 18). In the report, the 2-digit years 80 and 81 are less than the threshold; for the century, they assume the value 20 (DEFCENT + 1). The 2-digit year 82 is equal to the threshold; for the century, it defaults to the value 19 (DEFCENT).

The output is:

```
PAGE     1

HIRE_DATE   HIRE_DATE_PLUS_ONE
---------   ------------------
 80/06/02           2080/06/03
 81/07/01           2081/07/02
 82/05/01           1982/05/02
 82/01/04           1982/01/05
 82/08/01           1982/08/02
 82/01/04           1982/01/05
 82/07/01           1982/07/02
 81/07/01           2081/07/02
 82/04/01           1982/04/02
 82/02/02           1982/02/03
 82/04/01           1982/04/02
 81/11/02           2081/11/03
```

Running the report in 2018 illustrates the automatic recalculation of DEFCENT from 19 to 20. In 2018, the window spans the years 2000 to 2099. The threshold is 0 (2018 - 18). A 2-digit year greater than or equal to 0 defaults to the recalculated value 20 (DEFCENT).

Since all the values for the HIRE_DATE year are greater than 0, the century defaults to 20.

The output is:

```
PAGE     1

HIRE_DATE   HIRE_DATE_PLUS_ONE
---------   ------------------
 80/06/02           2080/06/03
 81/07/01           2081/07/02
 82/05/01           2082/05/02
 82/01/04           2082/01/05
 82/08/01           2082/08/02
 82/01/04           2082/01/05
 82/07/01           2082/07/02
 81/07/01           2081/07/02
 82/04/01           2082/04/02
 82/02/02           2082/02/03
 82/04/01           2082/04/02
 81/11/02           2081/11/03
```

Querying the Current Global Value of DEFCENT and YRTHRESH

You can query the current global value of DEFCENT and YRTHRESH.

Syntax: How to Query the Current Global Value of DEFCENT and YRTHRESH

```
? SET DEFCENT
? SET YRTHRESH
```

where:

`DEFCENT`

Returns the value for the DEFCENT parameter.

`YRTHRESH`

Returns the value for the YRTHRESH parameter.

Example: Querying the Current Global Value of DEFCENT and YRTHRESH

Enter

```
? SET DEFCENT
? SET YRTHRESH
```

to query the current global value of DEFCENT and YRTHRESH.

The following is a response to the query:

```
DEFCENT        19
YRTHRESH 0
```

Defining a File-Level or Field-Level Window in a Master File

In this implementation of the sliding window technique, you change the metadata used by an application. Two pairs of Master File attributes enable you to define a window on a file or field level:

- The FDEFCENT and FYRTHRESH attributes define a window on a file level. They enable the correct interpretation of legacy date fields from multiple files that span different time periods.

  A file-level window takes precedence over a global window for the dates associated with that file.

- The DEFCENT and YRTHRESH attributes define a window on a field level, enabling the correct interpretation of legacy date fields, within a single file, that span different time periods. Each legacy date field in a file can have its own window. For example, in an insurance application, the range of dates for date of birth may be from 1910 to 2009, and the range of dates for expected death may be from 1990 to 2089.

  A field-level window takes precedence over a file-level or global window for the dates associated with that field.

For details on Master Files, see the *Describing Data* manual.

Syntax: How to Define a File-Level Window in a Master File

To define a window that applies to all legacy date fields in a file, add the FDEFCENT and FYRTHRESH attributes to the Master File on the file declaration.

The syntax for the first attribute is

```
{FDEFCENT|FDFC} = {cc|19}
```

where:

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

The syntax for the second attribute is

```
{FYRTHRESH|FYRT} = {[-]yy|0}
```

where:

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of DEFCENT for the century. Two-digit years less than the threshold assume the value of DEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and DEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

Example: Defining a File-Level Window in a Master File

Tip: Use the abbreviated forms of FDEFCENT/FYRTHRESH or DEFCENT/YRTHRESH to reduce keystrokes. The examples in this topic use the abbreviated forms where available (for instance, FDFC instead of FDEFCENT). Maintain supports only the abbreviated forms in certain command syntax (for example, on a COMPUTE or DECLARE command). For details, see the *Maintaining Databases* manual.

In the following example, the FDEFCENT and FYRTHRESH attributes define a window from 1982 to 2081. The window is applied to all legacy date fields in the file, including HIRE_DATE, DAT_INC, and others, if they are converted to a date format.

The Master File is:

```
FILENAME=EMPLOYEE, SUFFIX=FOC, FDFC=19, FYRT=82
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,        ALIAS=EID,     FORMAT=A9,       $
  FIELDNAME=LAST_NAME,     ALIAS=LN,      FORMAT=A15,      $
  FIELDNAME=FIRST_NAME,    ALIAS=FN,      FORMAT=A10,      $
  FIELDNAME=HIRE_DATE,     ALIAS=HDT,     FORMAT=I6YMD,    $
.
.
.
  FIELDNAME=DAT_INC,       ALIAS=DI,      FORMAT=I6YMD,    $
.
.
.
```

The DEFINE command in the following request creates two virtual fields named NEW_HIRE_DATE, which is derived from the existing field HIRE_DATE; and NEW_DAT_INC, which is derived from DAT_INC. The format of HIRE_DATE and DAT_INC is I6YMD, which is a legacy date with a 2-digit year. NEW_HIRE_DATE and NEW_DAT_INC are date formats with 4-digit years (YYMD). For details on DEFINE, see the *Creating Reports* manual.

```
DEFINE FILE EMPLOYEE
NEW_HIRE_DATE/YYMD = HIRE_DATE;
NEW_DAT_INC/YYMD = DAT_INC;
END

TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE DAT_INC NEW_DAT_INC
END
```

The window created in the Master File applies to both legacy date fields. In the report, the year 82 (which is equal to the threshold), for both HIRE_DATE and DAT_INC, defaults to the century value 19 and is returned as 1982 in the NEW_HIRE_DATE and NEW_DAT_INC columns. The year 81, for both HIRE_DATE and DAT_INC, is less than the threshold 82 and assumes the century value 20 (FDEFCENT + 1).

The partial output is:

```
PAGE     1

HIRE_DATE   NEW_HIRE_DATE   DAT_INC   NEW_DAT_INC
---------   -------------   -------   -----------
 80/06/02   2080/06/02     82/01/01   1982/01/01
 80/06/02   2080/06/02     81/01/01   2081/01/01
 81/07/01   2081/07/01     82/01/01   1982/01/01
 82/05/01   1982/05/01     82/06/01   1982/06/01
 82/05/01   1982/05/01     82/05/01   1982/05/01
.
.
.
```

Syntax: How to Define a Field-Level Window in a Master File

To define a window that applies to a specific legacy date field, add the DEFCENT and YRTHRESH attributes to the Master File on the field declaration.

The syntax for the first attribute is

```
{DEFCENT|DFC} = {cc|19}
```

where:

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

The syntax for the second attribute is

```
{YRTHRESH|YRT} = {[-]yy|0}
```

where:

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of DEFCENT for the century. Two-digit years less than the threshold assume the value of DEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and DEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

Example: Defining a Field-Level Window in a Master File

In this example, the application requires a different window for two legacy date fields in the same file.

The DEFCENT and YRTHRESH attributes in the Master File define a window for HIRE_DATE from 1982 to 2081, and a window for DAT_INC from 1983 to 2082.

The Master File is:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,       ALIAS=EID,    FORMAT=A9,                    $
  FIELDNAME=LAST_NAME,    ALIAS=LN,     FORMAT=A15,                   $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,     FORMAT=A10,                   $
  FIELDNAME=HIRE_DATE,    ALIAS=HDT,    FORMAT=I6YMD, DFC=19, YRT=82, $
.
.
.
  FIELDNAME=DAT_INC,      ALIAS=DI,     FORMAT=I6YMD, DFC=19, YRT=83, $
.
.
.
```

The request is the same one used in the previous example (defining a file-level window in a Master File):

```
DEFINE FILE EMPLOYEE
NEW_HIRE_DATE/YYMD = HIRE_DATE;
NEW_DAT_INC/YYMD = DAT_INC;
END

TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE DAT_INC NEW_DAT_INC
END
```

However, the report illustrates the use of two different windows for the two legacy date fields. For example, the year 82 for HIRE_DATE defaults to the century value 19, since 82 is equal to the threshold for the window for this field. The date returned for NEW_HIRE_DATE is 1982.

The year 82 for DAT_INC assumes the century value 20 (DEFCENT + 1), since 82 is less than the threshold for the window for this field (83). The date returned for NEW_DAT_INC is 2082.

The partial output is:

```
PAGE     1

HIRE_DATE   NEW_HIRE_DATE   DAT_INC   NEW_DAT_INC
---------   -------------   -------   -----------
 80/06/02   2080/06/02     82/01/01   2082/01/01
 80/06/02   2080/06/02     81/01/01   2081/01/01
 81/07/01   2081/07/01     82/01/01   2082/01/01
 82/05/01   1982/05/01     82/06/01   2082/06/01
 82/05/01   1982/05/01     82/05/01   2082/05/01
.
.
```

Example: Defining a Field-Level Window in a Master File Used With MODIFY

This example illustrates the use of field-level DEFCENT and YRTHRESH attributes to define a window used with MODIFY. To run this example yourself, you need to create a Master File named DATE and a procedure named DATELOAD.

The Master File describes a segment with 12 date fields of different formats. The first field is a date format field. The DEFCENT and YRTHRESH attributes included on this field create a window from 1990 to 2089. The window is required because the input data for the first date field does not contain century digits, and the default value 19 cannot be assumed.

The Master File looks like this:

```
FILENAME=DATE, SUFFIX=FOC
 SEGNAME=ONE,  SEGTYPE=S1
  FIELDNAME=D1_YYMD,    ALIAS=D1,   FORMAT=YYMD, DFC=19, YRT=90,  $
  FIELDNAME=D2_I6YMD,   ALIAS=D2,   FORMAT=I6YMD,                 $
  FIELDNAME=D3_I8YYMD, ALIAS=D3,   FORMAT=I8,                     $
  FIELDNAME=D4_A6YMD,   ALIAS=D4,   FORMAT=A6YMD,                 $
  FIELDNAME=D5_A8YYMD, ALIAS=D5,   FORMAT=A8YYMD,                 $
  FIELDNAME=D6_I4YM,    ALIAS=D6,   FORMAT=I4YM,                  $
  FIELDNAME=D7_YQ,      ALIAS=D7,   FORMAT=YQ,                    $
  FIELDNAME=D8_YM,      ALIAS=D8,   FORMAT=YM,                    $
  FIELDNAME=D9_JUL,     ALIAS=D9,   FORMAT=JUL,                   $
  FIELDNAME=D10_Y,      ALIAS=D10,  FORMAT=Y,                     $
  FIELDNAME=D11_YY,     ALIAS=D11,   FORMAT=YY,                   $
  FIELDNAME=D12_MDYY,   ALIAS=D12,  FORMAT=MDYY,                  $
```

The procedure (DATELOAD) creates a FOCUS data source named DATE and loads two records into it. The first field of the first record contains the 2-digit year 92. The first field of the second record contains the 2-digit year 88. For details on commands such as CREATE and MODIFY, and others used in this file, see the *Maintaining Databases* manual.

The procedure looks like this:

```
CREATE FILE DATE
MODIFY FILE DATE
FIXFORM D1/8 D2/6 D3/8 D4/6 D5/8 D6/4 D7/4 D8/4 D9/5 D10/2 D11/4 D12/8
MATCH D1
   ON NOMATCH INCLUDE
   ON MATCH REJECT
DATA
  9202290002292000022900022920000229000200010002000600020000229200 0
  8802290002292000022900022920000229000200010002000600020000229200 0
END
```

The following request accesses all the fields in the new data source:

```
TABLE FILE DATE
PRINT *
END
```

In the report, the year 92 for D1_YYMD defaults to the century value 19, since 92 is greater than the threshold for the window for this field (90). It is returned as 1992 in the D1_YYMD column. The year 88 assumes the century value 20 (DEFCENT + 1), because 88 is less than the threshold. It is returned as 2088 in the D1_YYMD column.

The partial output is:

```
PAGE     1

D1_YYMD     D2_I6YMD  D3_I8YYMD  D4_A6YMD  D5_A8YYMD  D6_I4YM  D7_YQ  D8_YM ...
-------     --------  ---------  --------  ---------  -------  -----  ----- 
1992/02/29  00/02/29   20000229  00/02/29  2000/02/29   00/02  00 Q1  00/02 ...
2088/02/29  00/02/29   20000229  00/02/29  2000/02/29   00/02  00 Q1  00/02 ...
```

Example: Defining Both File-Level and Field-Level Windows

The following Master File defines windows at both the file and field level:

```
FILENAME=EMPLOYEE, SUFFIX=FOC, FDFC=19, FYRT=83
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,        ALIAS=EID,    FORMAT=A9,                      $
  FIELDNAME=LAST_NAME,     ALIAS=LN,     FORMAT=A15,                     $
  FIELDNAME=FIRST_NAME,    ALIAS=FN,     FORMAT=A10,                     $
  FIELDNAME=HIRE_DATE,     ALIAS=HDT,    FORMAT=I6YMD, DFC=19, YRT=82,   $
.
.
.
  FIELDNAME=EFFECT_DATE,   ALIAS=EDATE,  FORMAT=I6YMD,                   $
.
.
.

  FIELDNAME=DAT_INC,       ALIAS=DI,     FORMAT=I6YMD,                   $
.
.
.
```

The request is:

```
DEFINE FILE EMPLOYEE
NEW_HIRE_DATE/YYMD = HIRE_DATE;
NEW_EFFECT_DATE/YYMD = EFFECT_DATE;
NEW_DAT_INC/YYMD = DAT_INC;
END

TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE EFFECT_DATE NEW_EFFECT_DATE DAT_INC
NEW_DAT_INC
END
```

When the field HIRE_DATE is accessed, the time span 1982 to 2081 is applied. For all other legacy date fields in the file, such as EFFECT_DATE and DAT_INC, the time span specified at the file level is applied, that is, 1983 to 2082.

For example, the year 82 for HIRE_DATE is returned as 1982 in the NEW_HIRE_DATE column, since 82 is equal to the threshold of the window for that particular field. The year 82 for EFFECT_DATE and DAT_INC is returned as 2082 in the columns NEW_EFFECT_DATE and NEW_DAT_INC, since 82 is less than the threshold of the file-level window (83).

The partial output is:

```
PAGE     1

HIRE_DATE NEW_HIRE_DATE  EFFECT_DATE  NEW_EFFECT_DATE  DAT_INC  NEW_DAT_INC
--------- -------------  -----------  ---------------   -------  ----------
 80/06/02  2080/06/02                                  82/01/01  2082/01/01
 80/06/02  2080/06/02                                  81/01/01  2081/01/01
 81/07/01  2081/07/01                                  82/01/01  2082/01/01
 82/05/01  1982/05/01      82/11/01  2082/11/01        82/06/01  2082/06/01
 82/05/01  1982/05/01      82/11/01  2082/11/01        82/05/01  2082/05/01
.
```

Missing date values for NEW_EFFECT_DATE appear as blanks by default. To retrieve the base date value for NEW_EFFECT_DATE instead of blanks, issue the command

```
SET DATEDISPLAY = ON
```

before running the request. The base date value is returned as 1900/12/31. See *Defining a Global Window With SET* on page 402 for sample results.

Defining a Window for a Virtual Field

The DEFCENT and YRTHRESH parameters on a DEFINE command create a window for a virtual field. The window is used to interpret date values for the virtual field when the century is not supplied. You can issue a DEFINE command in either a request or a Master File.

The DEFCENT and YRTHRESH parameters must immediately follow the field format specification; the values are always taken from the left side of the DEFINE syntax (that is, from the left side of the equal sign). If the expression in the DEFINE contains a function call, the function uses the DEFCENT and YRTHRESH values for the input field. The standard order of precedence (field level/file level/global level) applies to the DEFCENT and YRTHRESH values for the input field.

Syntax: How to Define a Window for a Virtual Field in a Request

Use standard DEFINE syntax for a request, as described in the *Creating Reports* manual. Partial DEFINE syntax is shown here.

On the line that specifies the name of the virtual field, include the DEFCENT and YRTHRESH parameters and values. The parameters must immediately follow the field format information.

```
DEFINE FILE filename
  fieldname[/format] [{DEFCENT|DFC} {cc|19} {YRTHRESH|YRT} {[-]yy|0}] =
    expression;
.
.
.
END
```

where:

filename

Is the name of the file for which you are creating the virtual field.

fieldname

Is the name of the virtual field.

format

Is a date format such as DMY or YYMD.

DEFCENT

Is the parameter for the default century.

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

YRTHRESH

Is the parameter for the year threshold. You must code values for both DEFCENT and YRTHRESH unless YRTHRESH is negative. In that case, only code a value for YRTHRESH.

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of DEFCENT for the century. Two-digit years less than the threshold assume the value of DEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and DEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

expression

Is a valid arithmetic or logical expression, function, or function that determines the value of the virtual field.

END

Is required to terminate the DEFINE command.

Example: Defining a Window for a Virtual Field in a Request

In the following request, the DEFINE command creates two virtual fields, GLOBAL_HIRE_DATE and WINDOWED_HIRE_DATE. Both virtual fields are derived from the existing field HIRE_DATE. The format of HIRE_DATE is I6YMD, which is a legacy date with a 2-digit year. The virtual fields are date formats with a 4-digit year (YYMD).

The second virtual field, WINDOWED_HIRE_DATE, has the additional parameters DEFCENT and YRTHRESH, which define a window from 1982 to 2081. Notice that both DEFCENT and YRTHRESH are coded, as required.

The request is:

```
DEFINE FILE EMPLOYEE
GLOBAL_HIRE_DATE/YYMD = HIRE_DATE;
WINDOWED_HIRE_DATE/YYMD DFC 19 YRT 82 = HIRE_DATE;
END

TABLE FILE EMPLOYEE
PRINT HIRE_DATE GLOBAL_HIRE_DATE WINDOWED_HIRE_DATE
END
```

Assuming that there are no FDEFCENT and FYRTHRESH file-level settings in the Master File for EMPLOYEE, the global default settings (DEFCENT = 19, YRTHRESH = 0) are used to interpret 2-digit years for HIRE_DATE when deriving the value of GLOBAL_HIRE_DATE. For example, the value of all years for HIRE_DATE (80, 81, and 82) is greater than 0; consequently they default to 19 for the century and are returned as 1980, 1981, and 1982 in the GLOBAL_HIRE_DATE column.

For WINDOWED_HIRE_DATE, the window created specifically for that field (1982 to 2081) is used. The 2-digit years 80 and 81 for HIRE_DATE are less than the threshold for the window (82); consequently, they are returned as 2080 and 2081 in the WINDOWED_HIRE_DATE column.

The output is:

```
PAGE     1

HIRE_DATE   GLOBAL_HIRE_DATE  WINDOWED_HIRE_DATE
---------   ----------------  ------------------
 80/06/02   1980/06/02        2080/06/02
 81/07/01   1981/07/01        2081/07/01
 82/05/01   1982/05/01        1982/05/01
 82/01/04   1982/01/04        1982/01/04
 82/08/01   1982/08/01        1982/08/01
 82/01/04   1982/01/04        1982/01/04
 82/07/01   1982/07/01        1982/07/01
 81/07/01   1981/07/01        2081/07/01
 82/04/01   1982/04/01        1982/04/01
 82/02/02   1982/02/02        1982/02/02
 82/04/01   1982/04/01        1982/04/01
 81/11/02   1981/11/02        2081/11/02
```

Example: Defining a Window for Function Input in a DEFINE Command

The following sample request illustrates a call to the function AYMD in a DEFINE command. AYMD adds 60 days to the input field, HIRE_DATE; the output field, SIXTY_DAYS, contains the result. HIRE_DATE is formatted as I6YMD, which is a legacy date with a 2-digit year. SIXTY_DAYS is formatted as I8YYMD, which is a legacy date with a 4-digit year.

For details on AYMD, see the *Using Functions* manual.

```
DEFINE FILE EMPLOYEE
SIXTY_DAYS/I8YYMD = AYMD(HIRE_DATE, 60, 'I8YYMD');
END


TABLE FILE EMPLOYEE
PRINT HIRE_DATE SIXTY_DAYS
END
```

The function uses the DEFCENT and YRTHRESH values for the input field HIRE_DATE. In this example, they are set on the field level in the Master File:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,       ALIAS=EID,    FORMAT=A9,                    $
  FIELDNAME=LAST_NAME,    ALIAS=LN,     FORMAT=A15,                   $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,     FORMAT=A10,                   $
  FIELDNAME=HIRE_DATE,    ALIAS=HDT,    FORMAT=I6YMD, DFC=19, YRT=82, $
.
.
.
```

The function inputs a 2-digit year, which is windowed. It then outputs a 4-digit year that includes the century digits.

The input values 80 and 81 are less than the threshold 82, so they assume the value 20 for the century. The input value 82 is equal to the threshold, so it defaults to 19 for the century.

The output is:

```
PAGE     1

HIRE_DATE   SIXTY_DAYS
---------   ----------
 80/06/02   2080/08/01
 81/07/01   2081/08/30
 82/05/01   1982/06/30
 82/01/04   1982/03/05
 82/08/01   1982/09/30
 82/01/04   1982/03/05
 82/07/01   1982/08/30
 81/07/01   2081/08/30
 82/04/01   1982/05/31
 82/02/02   1982/04/03
 82/04/01   1982/05/31
 81/11/02   2082/01/01
```

Syntax: How to Define a Window for a Virtual Field in a Master File

Use standard DEFINE syntax for a Master File, as discussed in your documentation on describing data the *Describing Data* manual. Partial DEFINE syntax is shown here.

The parameters DEFCENT and YRTHRESH must immediately follow the field format information.

```
DEFINE fieldname/[format] [{DEFCENT|DFC} {cc|19}
       {YRTHRESH|YRT} {[-]yy|0}] =    expression;$
```

where:

fieldname

Is the name of the virtual field.

format

Is a date format such as DMY or YYMD.

DEFCENT

Is the parameter for the default century.

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

YRTHRESH

Is the parameter for the year threshold. You must code values for both DEFCENT and YRTHRESH unless YRTHRESH is negative. In that case, only code a value for YRTHRESH.

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of DEFCENT for the century. Two-digit years less than the threshold assume the value of DEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and DEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

expression

Is a valid arithmetic or logical expression, function, or function that determines the value of the virtual field.

Example: Defining a Window for a Virtual Field in a Master File

In the following example, the DEFINE command in a Master File creates a virtual field named NEW_HIRE_DATE. It is derived from the existing field HIRE_DATE. The format of HIRE_DATE is I6YMD, which is a legacy date with a 2-digit year. NEW_HIRE_DATE is a date format with a 4-digit year (YYMD).

The parameters DEFCENT and YRTHRESH on the DEFINE command create a window from 1982 to 2081, which is used to interpret all 2-digit years for the virtual field. Notice that both DEFCENT and YRTHRESH are coded, as required.

The field-level window takes precedence over any global settings in effect. There is no file-level setting in the Master File.

The Master File is:

```
FILENAME=EMPLOYEE, SUFFIX=FOC
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,       ALIAS=EID,    FORMAT=A9,          $
  FIELDNAME=LAST_NAME,    ALIAS=LN,     FORMAT=A15,         $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,     FORMAT=A10,         $
  FIELDNAME=HIRE_DATE,    ALIAS=HDT,    FORMAT=I6YMD,       $
.
.
.
DEFINE NEW_HIRE_DATE/YYMD DFC 19 YRT 82 = HIRE_DATE;$
```

The following request generates the values in the sample report:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE
END
```

Since the 2-digit years 80 and 81 are less than the threshold 82, the century assumes the value of DEFCENT + 1 (20), and they are returned as 2080 and 2081 in the NEW_HIRE_DATE column. The 2-digit year 82 is equal to the threshold and therefore defaults to the value of DEFCENT (19). It is returned as 1982.

The output is:

```
PAGE     1

HIRE_DATE     NEW_HIRE_DATE
---------     -------------
 80/06/02     2080/06/02
 81/07/01     2081/07/01
 82/05/01     1982/05/01
 82/01/04     1982/01/04
 82/08/01     1982/08/01
 82/01/04     1982/01/04
 82/07/01     1982/07/01
 81/07/01     2081/07/01
 82/04/01     1982/04/01
 82/02/02     1982/02/02
 82/04/01     1982/04/01
 81/11/02     2081/11/02
```

Defining a Window for a Calculated Value

Use the DEFCENT and YRTHRESH parameters on a COMPUTE command in a report request to create a window for a temporary field that is calculated from the result of a PRINT, LIST, SUM, or COUNT command. The window is used to interpret a date value for that field when the century is not supplied.

The DEFCENT and YRTHRESH parameters must immediately follow the field format specification; the values are always taken from the left side of the COMPUTE syntax (that is, from the left side of the equal sign). If the expression in the COMPUTE contains a function call, the function uses the DEFCENT and YRTHRESH values for the input field. The standard order of precedence (field level/file level/global level) applies to the DEFCENT and YRTHRESH values for the input field.

You can also use the parameters on a COMPUTE command in a MODIFY or Maintain procedure, or on a DECLARE command in Maintain. For details on the use of the parameters in Maintain, see the *Maintaining Databases* manual.

Syntax: How to Define a Window for a Calculated Value in a Report

Use standard COMPUTE syntax, as described in the *Creating Reports* manual. Partial COMPUTE syntax is shown here.

On the line that specifies the name of the calculated value, include the DEFCENT and YRTHRESH parameters and values. The parameters must immediately follow the field format information.

```
TABLE FILE filename
command
[AND] COMPUTE fieldname[/format] [{DEFCENT|DFC} {cc|19} {YRTHRESH|YRT}
{[-]yy|0}] =
    expression;
.
.
.
END
```

where:

filename

Is the name of the file for which you are creating the calculated value.

command

Is a command such as PRINT, LIST, SUM, or COUNT.

fieldname

Is the name of the calculated value.

format

Is a date format such as DMY or YYMD.

DEFCENT

Is the parameter for the default century.

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

YRTHRESH

Is the parameter for the year threshold. You must code values for both DEFCENT and YRTHRESH unless YRTHRESH is negative. In that case, only code a value for YRTHRESH.

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of DEFCENT for the century. Two-digit years less than the threshold assume the value of DEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and DEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

expression

Is a valid arithmetic or logical expression, function, or function that determines the value of the temporary field.

END

Is required to terminate the request.

Syntax: How to Define a Window for a Calculated Value in a MODIFY Request

Use standard MODIFY and COMPUTE syntax, as described in the *Maintaining Databases* manual; partial syntax is shown here.

On the line that specifies the name of the calculated value, include the DEFCENT and YRTHRESH parameters and values. The parameters must immediately follow the field format information.

```
MODIFY FILE filename
.
.
.
COMPUTE
  fieldname[/format] [{DEFCENT|DFC} {cc|19} {YRTHRESH|YRT} {[-]yy|0}] =
    expression;
.
.
.
[END]
```

where:

filename

Is the name of the file you are modifying.

fieldname

Is the name of the field being set to the value of *expression*.

format

Is a date format such as MDY or YYMD.

DEFCENT

Is the parameter for the default century.

cc

Is the century for the start date of the window. If you do not supply a value, *cc* defaults to 19, for the twentieth century.

YRTHRESH

Is the parameter for the year threshold. You must code values for both DEFCENT and YRTHRESH unless YRTHRESH is negative. In that case, only code a value for YRTHRESH.

yy

Is the year threshold for the window. If you do not supply a value, *yy* defaults to zero (0).

If *yy* is a positive number, two-digit years greater than or equal to the threshold default to the value of DEFCENT for the century. Two-digit years less than the threshold assume the value of DEFCENT + 1.

If *yy* is a negative number (*-yy*), the start date of the window is derived by subtracting that number from the current year, and DEFCENT is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

expression

Is a valid arithmetic or logical expression, function, or function that determines the value of *fieldname*.

END

Terminates the request. Do not add this command if the request contains PROMPT statements.

Example: Defining a Window for a Calculated Value

In the following request, the parameters DEFCENT and YRTHRESH on the COMPUTE command define a window from 1999 to 2098. Notice that both DEFCENT and YRTHRESH are coded, as required. The window is applied to the field created by the COMPUTE command, LATEST_DAT_INC.

DAT_INC is formatted as I6YMD, which is a legacy date with a 2-digit year. LATEST_DAT_INC is a date format with a 4-digit year (YYMD). The prefix MAX retrieves the highest value of DAT_INC.

The request is:

```
TABLE FILE EMPLOYEE
SUM SALARY AND COMPUTE
  LATEST_DAT_INC/YYMD DFC 19 YRT 99 = MAX.DAT_INC;
END
```

The highest value of DAT_INC is 82/08/01. Since the year 82 is less than the threshold 99, it assumes the value 20 for the century (DEFCENT + 1).

The output is:

```
PAGE     1

         SALARY  LATEST_DAT_INC
         ------  --------------
    $332,929.00  2082/08/01
```

Example: Defining a Window for Function Input in a COMPUTE Command

The following sample request illustrates a call to the function JULDAT in a COMPUTE command. JULDAT converts dates from Gregorian format (year/month/day) to Julian format (year/day). For century display, dates in Julian format are 7-digit numbers. The first 4 digits are the century. The last three digits represent the number of days, counting from January 1.

For details on JULDAT, see the *Using Functions* manual.

In the request, the input field is HIRE_DATE. The function converts it to Julian format and returns it as JULIAN_DATE. HIRE_DATE is formatted as I6YMD, which is a legacy date with a 2-digit year. JULIAN_DATE is formatted as I7, which is a legacy date with a 4-digit year.

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT HIRE_DATE
AND COMPUTE
   JULIAN_DATE/I7 = JULDAT(HIRE_DATE, JULIAN_DATE);
BY LAST_NAME BY FIRST_NAME
END
```

The function uses the FDEFCENT and FYRTHRESH values for the input field HIRE_DATE. In this example, they are set on the file level in the Master File:

```
FILENAME=EMPLOYEE, SUFFIX=FOC, FDFC=19, FYRT=82
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,       ALIAS=EID,    FORMAT=A9,         $
  FIELDNAME=LAST_NAME,    ALIAS=LN,     FORMAT=A15,        $
  FIELDNAME=FIRST_NAME,   ALIAS=FN,     FORMAT=A10,        $
  FIELDNAME=HIRE_DATE,    ALIAS=HDT,    FORMAT=I6YMD,      $
.
.
.
```

The function inputs a 2-digit year, which is windowed. It then outputs a 4-digit year that includes the century digits.

The input values 80 and 81 are less than the threshold 82, so they assume the value 20 for the century. The input value 82 is equal to the threshold, so it defaults to 19 for the century.

The output follows. By default, the second occurrence of the last name SMITH displays as blanks.

```
PAGE     1

LAST_NAME      FIRST_NAME   DEPARTMENT   HIRE_DATE   JULIAN_DATE
---------      ----------   ----------   ---------   -----------
BANNING        JOHN         PRODUCTION    82/08/01       1982213
BLACKWOOD      ROSEMARIE    MIS           82/04/01       1982091
CROSS          BARBARA      MIS           81/11/02       2081306
GREENSPAN      MARY         MIS           82/04/01       1982091
IRVING         JOAN         PRODUCTION    82/01/04       1982004
JONES          DIANE        MIS           82/05/01       1982121
MCCOY          JOHN         MIS           81/07/01       2081182
MCKNIGHT       ROGER        PRODUCTION    82/02/02       1982033
ROMANS         ANTHONY      PRODUCTION    82/07/01       1982182
SMITH          MARY         MIS           81/07/01       2081182
               RICHARD      PRODUCTION    82/01/04       1982004
STEVENS        ALFRED       PRODUCTION    80/06/02       2080154
```

Additional Support for Cross-Century Dates

The following features apply to the use of dates in your applications.

Default Date Display Format

The default date display format is MM/DD/CCYY, where MM is the month; DD is the day of the month; CC is the first two digits of a 4-digit year, indicating the century; and YY is the last two digits of a 4-digit year.

For example:

```
02/11/1999
```

For a table that fully describes the display of a date based on the specified format and user input, see the *Describing Data* manual.

Date Display Options

The following date display options are available:

- You can display a row of data, even though it contains an invalid date field, using the command SET ALLOWCVTERR. The invalid date field is returned as the base date or as blanks, depending on other settings. For details, see your documentation on the SET command. This feature applies to non-FOCUS data sources when converting from the way data is stored (ACTUAL attribute) to the way it is formatted (FORMAT or USAGE attribute).
- If a date format field contains the value zero (0), you can display its base date using the command SET DATEDISPLAY = ON. By default, the value zero in a date format field such as YYMD is returned as a blank. For details, see *Customizing Your Environment*.
- You can display the current date with a 4-digit year using the Dialogue Manager system variables &YYMD, &MDYY, and &DMYY. The system variable &DATE*fmt* displays the current date as specified by the value of *fmt*, which is a combination of allowable date options, including a 4-digit year (for example, &DATEYYMD). For details, see *Managing Flow of Control in an Application* on page 201.

System Date Masking

You can temporarily alter the system date for application testing and debugging using the command SET TESTDATE. With this feature, you can simulate clock settings beyond the year 1999 to determine the way your program will behave. For details, see *Customizing Your Environment*.

Date Functions

The date functions supplied with your software work across centuries. Many of them facilitate date manipulation. For details, see the *Using Functions* manual.

Date Conversion

You can convert a legacy date to a date format in a FOCUS data source using the option DATE NEW on the REBUILD command. For details, see the *Maintaining Databases* manual.

Century and Threshold Information

The ALL option, in conjunction with the HOLD option, on the CHECK FILE command includes file-level and field-level default century and year thresholds as specified in a Master File. For details, see the *Describing Data* manual.

Date Time Stamp

The year in the time stamp for a FOCUS data source is physically written to page one of the file in the format CCYY.

Chapter 8

Euro Currency Support

The following topics describe how to create and use a currency data source to convert to and from the new euro currency.

In this chapter:

- Integrating the Euro Currency
- Converting Currencies
- Creating the Currency Data Source
- Identifying Fields That Contain Currency Data
- Activating the Currency Data Source
- Processing Currency Data
- Querying the Currency Data Source in Effect
- Punctuating Numbers
- Selecting an Extended Currency Symbol

Integrating the Euro Currency

With the introduction of the euro currency, businesses need to maintain books in two currencies, add new fields to the data source designs, and perform new types of currency conversions. You can perform currency conversions according to the rules specified by the European Union. To do this:

1. Create a currency data source with the currency IDs and exchange rates you will use. See *Creating the Currency Data Source* on page 433.
2. Identify fields in your data sources that represent currency data. See *Identifying Fields That Contain Currency Data* on page 436.
3. Activate your currency data source. See *Activating the Currency Data Source* on page 438.
4. Perform currency conversions. See *Processing Currency Data* on page 439.

Note: As the euro symbol becomes available to operating systems, Information Builders will support it.

Converting Currencies

Euro currency was introduced in Euroland on January 1, 2002, and on July 1, 2002 it became the only legal tender. All monetary transactions now occur in euro currency.

The European Union has set fixed exchange rates between the euro and the traditional national currency in each of the 12 adopting member nations. Although 12 or more currencies in the European Union use the euro, more than 100 currencies have a recognized status worldwide. In addition, you may need to define custom currencies for other applications.

While the exchange rates within Euroland remain fixed, exchange rates between the euro and non-euro countries continue to vary freely and, in fact, several rates may be in use at one time (for example, actual and budgeted rates).

You identify your currency codes and rates by creating a currency data source. For more information, see *Creating the Currency Data Source* on page 433.

Reference: Currency Conversion Rules

The European Union has established the following rules for currency conversions:

- The exchange rate must be specified as a decimal value, *r*, with six significant digits.

  This rate will establish the following relationship between the euro and the particular national currency:

  1 euro = *r* national units

- To convert from the euro to the national unit, multiply by *r* and round the result to two decimal places.
- To convert from the national currency to the euro, divide by *r* and round the result to two decimal places.
- To convert from one national currency to another, first convert from one national unit to the euro, rounding the result to three decimal places (your application rounds to exactly three decimal places). Then convert from the euro to the second national unit, rounding the result to two decimal places. This two-step conversion process is *triangulation*.

Example: Performing Triangulation

The following example illustrates triangulation. In this case, 10 US dollars (USD) are converted to French francs (FRF). The exchange rate for USD to euros (EUR) is 1.17249. The exchange rate for FRF to euros is 6.55957.

- The 10 USD are converted to EUR by dividing the 10 USD by the EUR exchange rate of 0.8840:

```
EUR = 10 / 0.8840
```

This results in 11.3122 euros.

- The euros are converted to FRF by multiplying the above result by the exchange rate of FRF for euros (6.55957):

```
FRF = 11.3122 * 6.55957
```

The result is 74.26. FRF. This means 74.26 FRF are equivalent to 10 USD.

Creating the Currency Data Source

Supply values for each type of currency you need.

You must supply the following values in your currency data source:

- A three-character code to identify the currency, such as USD for US dollars or BEF for Belgian francs. (For a partial list of recognized currency codes, see *Sample Currency Codes* on page 434.)
- One or more exchange rates for the currency.

There is no limit to the number of currencies you can add to your currency data source, and the currencies you can define are not limited to official currencies. Therefore, the currency data source can be fully customized for your applications.

The currency data source can be any type of data source your application can access (for example, FOCUS, FIX, DB2, or VSAM). The currency Master File must have one field that identifies each currency ID you will use and one or more fields to specify the exchange rates.

We strongly recommend that you create a separate data source for the currency data rather than adding the currency fields to another data source. A separate currency data source enhances performance and minimizes resource utilization because the currency data source is loaded into memory before you perform currency conversions.

Syntax: How to Create a Currency Data Source

```
FILE = name, SUFFIX = suffix,$
FIELD  = CURRENCY_ID,, FORMAT =  A3,                    [ACTUAL = A3 ,]$
FIELD  = rate_1,,      FORMAT = {D12.6|numeric_format1},[ACTUAL = A12,]$
   .
   .
   .
FIELD  = rate_n,,      FORMAT = {D12.6|numeric_formatn}, [ACTUAL = A12,]$
```

where:

name

Is the name of the currency data source.

suffix

Is the suffix of the currency data source. The currency data source can be any type of data source your application can access.

CURRENCY_ID

Is the required field name. The values stored in this field are the three-character codes that identify each currency, such as USD for U.S. dollars. Each currency ID can be a universally recognized code or a user-defined code.

Note: The code EUR is automatically recognized. You should *not* store this code in your currency data source. See *Sample Currency Codes* on page 434 for a list of common currency codes.

rate_1...rate_n

Are types of rates (such as BUDGET, FASB, ACTUAL) to be used in currency conversions. Each rate is the number of national units that represent one euro.

numeric_format1...numeric_formatn

Are the display formats for the exchange rates. Each format must be numeric. The recommended format, D12.6, ensures that the rate is expressed with six significant digits as required by the European Union conversion rules. Do not use Integer format (I).

ACTUAL A*n*

Is required only for non-FOCUS data sources.

Note: The maximum number of fields in the currency data source must not exceed 255 (that is, the CURRENCY_ID field plus 254 currency conversion fields).

Reference: Sample Currency Codes

On January 1, 1999, Euroland set exchange rates between the euro and other currencies. Countries included in Euroland as of that date are marked with an asterisk (*). The rates are fixed and will not change, although the rates for other countries change over time.

Currency Name	Currency Code	Rate
American dollar	USD	.974298

Currency Name	Currency Code	Rate
Austrian schilling	ATS	13.7603
Belgian franc*	BEF	40.3399
British pound	GBP	.625152
Canadian dollar	CAD	1.54504
Danish krone	DKK	7.42659
Dutch guilder*	NLG	2.20371
Deutsche mark*	DEM	1.95583
Euro	EUR	1
Finnish markka	FIM	5.94573
French franc*	FRF	6.55957
Greek drachma*	GRD	340.750
Irish pound*	IEP	0.787564
Italian lira*	ITL	1936.27
Japanese yen or Chinese yuan	JPY	118.377
Luxembourg franc*	LUF	40.3399
Norwegian kroner	NOK	7.34864
Portuguese escudo*	PTE	200.482
Spanish peseta*	ESP	166.386
Swedish krona	SEK	9.20906
Swiss franc	CHF	1/4634

Example: Specifying Currency Codes and Rates in a Master File

The following Master File for a comma-delimited currency data source specifies two rates for each currency, ACTUAL and BUDGET:

```
FILE = CURRCODE, SUFFIX = COM,$
FIELD = CURRENCY_ID,,    FORMAT = A3,    ACTUAL = A3 ,$
FIELD = ACTUAL, ALIAS =, FORMAT = D12.6, ACTUAL = A12 ,$
FIELD = BUDGET, ALIAS =, FORMAT = D12.6, ACTUAL = A12 ,$
```

The following is sample data for the currency data source defined by this Master File:

```
FRF,  6.55957,  6.50000,$
USD,  0.974298,  1.00000,$
BEF,  40.3399, 41.00000,$
```

Identifying Fields That Contain Currency Data

After you have created your currency data source, you must identify the fields in your data sources that represent currency values. To designate a field as a currency-denominated value (a value that represents a number of units in a specific type of currency) add the CURRENCY attribute to one of the following:

- The FIELD specification in the Master File.
- The left side of a DEFINE or COMPUTE.

Syntax: How to Identify a Currency Value

Use the following syntax to identify a currency-denominated value.

In a Master File

```
FIELD = currfield,, FORMAT = numeric_format, ..., CURR =
{curr_id|codefield} ,$
DEFINE currfield/numeric_format CURR curr_id = expression ;$
DEFINE FILE filename
currfield/numeric_format CURR curr_id = expression ;
END
COMPUTE currfield/numeric_format CURR curr_id = expression ;
```

In a DEFINE in the Master File

```
DEFINE currfield/numeric_format CURR curr_id = expression ;$
```

In a DEFINE FILE command

```
DEFINE FILE filename
currfield/numeric_format CURR curr_id = expression ;
END
```

In a COMPUTE command

```
COMPUTE currfield/numeric_format CURR curr_id = expression ;
```

where:

currfield

Is the name of the currency-denominated field.

numeric_format

Is a numeric format. Depending on the currency denomination involved, the recommended number of decimal places is either two or zero. Do not use I or F format.

CURR

Indicates that the field value represents a currency-denominated value. CURR is an abbreviation of CURRENCY, which is the full attribute name.

curr_id

Is the three-character currency ID associated with the field. In order to perform currency conversions, this ID must either be the value EUR or match a CURRENCY_ID value in your currency data source.

codefield

Is the name of a field, qualified if necessary, that contains the currency ID associated with *currfield*. The code field should have format A3 or longer and is interpreted as containing the currency ID value in its first three bytes. For example:

```
FIELD = PRICE,, FORMAT = P12.2C, ..., CURR = TABLE.FLD1,$
   .
   .
   .
FIELD = FLD1,, FORMAT = A3, ...,$
```

The field named FLD1 contains the currency ID for the field named PRICE.

filename

Is the name of the file for which this field is defined.

expression

Is a valid expression.

Example: Identifying a Currency-Denominated Field

The following Master File contains the description of a field named PRICE that is denominated in U.S. dollars.

```
FILE=CURRDATA,SUFFIX=COM,$
FIELD=PRICE, FORMAT=P17.2 , ACTUAL=A5, CURR=USD,$
.
.
.
```

Activating the Currency Data Source

Before you can perform currency conversions, you must specify the currency data source by setting the EUROFILE parameter with the SET command. By default, the EUROFILE parameter is not set.

The SET command can be issued at the FOCUS command prompt, in a procedure, or in any supported profile. It cannot be set within a report request.

After a data source is activated, you can access a different currency data source by reissuing the SET command.

Note: The EUROFILE parameter must be set alone. For example, appending an additional SET parameter will cause the additional parameter setting to be lost.

Syntax: How to Activate Your Currency Data Source

```
SET EUROFILE = {ddname|OFF}
```

where:

ddname

Is the name of the Master File for the currency data source. The ddname must refer to a data source known to and accessible by your application in read-only mode.

OFF

Deactivates the currency data source and removes it from memory.

Reference: EUROFILE Error Messages and Notes

- Issuing the SET EUROFILE command when the currency data source Master File does not exist generates the following error message:

  ```
  (FOC205) THE DESCRIPTION CANNOT BE FOUND FOR FILE NAMED: ddname
  ```

- Issuing the SET EUROFILE command when the currency Master File specifies a FOCUS data source and the associated FOCUS data source does not exist generates the following error message:

```
(FOC036) NO DATA FOUND FOR THE FOCUS FILE NAMED: name
```

Processing Currency Data

After you have created your currency data source, identified the currency-denominated fields in your data sources, and activated your currency data source, you can perform currency conversions.

Each currency ID in your currency data source generates a virtual conversion function whose name is the same as its currency ID. For example, if you added BEF to your currency data source, a virtual BEF currency conversion function will be generated.

The euro function, EUR, is supplied automatically with your application. You do not need to add the EUR currency ID to your currency data source.

The result of a conversion is calculated with very high precision, 31 to 36 significant digits, depending on platform. The precision of the final result is always rounded to two decimal places. In order to display the result to the proper precision, its format must allow at least two decimal places.

Syntax: How to Process Currency Data

In a procedure

```
DEFINE FILE filename
result/format [CURR curr_id] = curr_id(infield, rate1 [,rate2]);
END
```

or

```
COMPUTE result/format [CURR curr_id] = curr_id(infield, rate1 [,rate2]);
```

In a Master File

```
DEFINE result/format [CURR curr_id] = curr_id(infield, rate1 [,rate2]);$
```

where:

filename

Is the name of the file for which this field is defined.

result

Is the converted currency value.

format

Is a numeric format. Depending on the currency denomination involved, the recommended number of decimal places is either two or zero. The result will always be rounded to two decimal places, which will display if the format allows at least two decimal places. Do not use an Integer or Floating Point format.

curr_id

Is the currency ID of the result field. This ID must be the value EUR or match a currency ID in your currency data source. Any other value generates the following message:

```
(FOC263)  EXTERNAL FUNCTION OR LOAD MODULE NOT FOUND: curr_id
```

Note: The CURR attribute on the left side of the DEFINE or COMPUTE identifies the result field as a currency-denominated value which can be passed as an argument to a currency function in subsequent currency calculations. Adding this attribute to the left side of the DEFINE or COMPUTE does not invoke any format or value conversion on the calculated result.

infield

Is a currency-denominated value. This input value will be converted from its original currency to the *curr_id* denomination. If the *infield* and *result* currencies are the same, no calculation is performed and the *result* value is the same as the *infield* value.

rate1

Is the name of a rate field from the currency data source. The *infield* value is divided by the *rate1* value of the currency to produce the equivalent number of euros.

If *rate2* is not specified in the currency calculation and triangulation is required, this intermediate result is then multiplied by the *result* currency *rate1* value to complete the conversion.

In certain cases, you may need to provide different rates for special purposes. In these situations you can specify any field or numeric constant for *rate1* as long as it indicates the number of units of the *infield* currency denomination that equals one euro.

rate2

Is the name of a rate field from the currency data source. This argument is only used for those cases of triangulation in which you need to specify different rate fields for the *infield* and *result* currencies. It is ignored if the euro is one of the currencies involved in the calculation.

The number of euros that was derived using *rate1* is multiplied by the *result* currency *rate2* value to complete the conversion.

In certain cases, you may need to provide different rates for special purposes. In these situations you can specify any field or numeric constant for *rate2* as long as it indicates the number of units of the *result* currency denomination that equals one euro.

Reference: Currency Calculation Error Messages

Issuing a report request against a Master File that specifies a currency code not listed in the active currency data source generates the following message:

```
(FOC1911) CURRENCY IN FILE DESCRIPTION NOT FOUND IN DATA
```

A syntax error or undefined field name in a currency conversion expression generates the following message:

```
(FOC1912)  ERROR IN PARSING CURRENCY STATEMENT
```

Example: Using the Currency Conversion Function

Assume that the currency data source contains the currency IDs USD and BEF, and that PRICE is denominated in Belgian francs as follows:

```
FIELD = PRICE, ALIAS=, FORMAT = P17.2, CURR=BEF,$
```

- The following example converts PRICE to euros and stores the result in PRICE2 using the BUDGET conversion rate for the BEF currency ID:

  ```
  COMPUTE PRICE2/P17.2 CURR EUR = EUR(PRICE, BUDGET);
  ```

- This example converts PRICE from Belgian francs to US dollars using the triangulation rule:

  ```
  DEFINE PRICE3/P17.2 CURR USD = USD(PRICE, ACTUAL);$
  ```

  First PRICE is divided by the ACTUAL rate for Belgian francs to derive the number of euros rounded to three decimal places. Then this intermediate value is multiplied by the ACTUAL rate for US dollars and rounded to two decimal places.

- The following example uses a numeric constant for the conversion rate:

  ```
  DEFINE PRICE4/P17.2 CURR EUR = EUR(PRICE,5);$
  ```

- The next example uses the ACTUAL rate for Belgian francs in the division and the BUDGET rate for US dollars in the multiplication:

  ```
  DEFINE PRICE5/P17.2 CURR USD = USD(PRICE, ACTUAL, BUDGET);$
  ```

Example: Converting U.S. Dollars to Euros, French Francs, and Belgian Francs

The following is an example of converting U.S. dollars to Euros, French Francs, and Belgian Francs.

1. Create a currency data source that identifies the currency and one or more exchange rates. (See *Creating the Currency Data Source* on page 433 for details.) The following sample data source is named CURRCODE:

```
FILE = CURRCODE, SUFFIX = COM,$
FIELD = CURRENCY_ID,,    FORMAT = A3,    ACTUAL = A3 ,$
FIELD = ACTUAL, ALIAS =, FORMAT = D12.6, ACTUAL = A12 ,$
FIELD = BUDGET, ALIAS =, FORMAT = D12.6, ACTUAL = A12 ,$
```

2. Create a data source that contains the values to be converted. (See *Identifying Fields That Contain Currency Data* on page 436 for details.) The following sample data source is named CURRDATA:

```
FILE=CURRDATA,SUFFIX=COM,$
FIELD=PRICE, FORMAT=P17.2 , ACTUAL=A5, CURR=USD,$
```

3. Create a request that uses the currency data source to convert the currency values contained in the data source containing these values. The following procedure converts PRICE to euros, French francs, and Belgian francs. The numbers on the left correspond to the notes explaining the code.

```
   -* THE FOLLOWING ALLOCATIONS ARE FOR RUNNING UNDER z/OS
1. -* DYNAM ALLOC FILE CURRCODE DA USER1.FOCEXEC.DATA(CURRCODE) SHR REU
2. -* DYNAM ALLOC FILE CURRDATA DA USER1.FOCEXEC.DATA(CURRDATA) SHR REU
    -* THE FOLLOWING ALLOCATIONS ARE FOR RUNNING UNDER WINDOWS NT
1.     FILEDEF CURRCODE DISK GGDEMO/CURRCODE.COM
2.    FILEDEF CURRCODE DISK GGDEMO/CURRDATA.COM
3. SET EUROFILE = CURRCODE
   DEFINE FILE CURRDATA
4. PRICEEUR/P17.2 CURR EUR = EUR(PRICE, ACTUAL);
    END
    TABLE FILE CURRDATA
    PRINT PRICE PRICEEUR AND COMPUTE
5. PRICEFRF/P17.2 CURR FRF = FRF(PRICE, ACTUAL);
    PRICEBEF/P17.2 CURR BEF = BEF(PRICE, ACTUAL);
    END
```

The report request executes as follows:

1. The FILEDEF or DYNAM command informs the operating system of the location of the CURRCODE data source.
2. The FILEDEF command informs the operating system of the location of the CURRDATA data source.
3. The SET command specifies the currency data source as CURRCODE.

4. This line calls the EUR function, which converts U.S. dollars to euros.
5. The next two lines are the conversion functions that convert euros into the equivalent in French and Belgian Francs.

The output is:

```
PRICE          PRICEEUR          PRICEFRF          PRICEBEF
-----          --------          --------          --------
 5.00              4.26             27.97            172.01
 6.00              5.12             33.57            206.42
40.00             34.12            223.78           1376.20
10.00              8.53             55.95            344.06
```

You cannot use the derived euro value PRICEEUR in a conversion from USD to BEF. PRICEEUR has two decimal places (P17.2), not three, as the triangulation rules require.

Querying the Currency Data Source in Effect

You can issue a query to determine what currency data source is in effect. To do this, issue ? SET ALL or ? SET EUROFILE.

Syntax: How to Determine the Currency Data Source in Effect

```
? SET EUROFILE
```

Example: Determining the Currency Data Source in Effect

Assume the currency data source is named CURRCODE.

If you issue the following commands

```
SET EUROFILE = CURRCODE
? SET EUROFILE
```

the output is:

```
EUROFILE              CURRCODE
```

Punctuating Numbers

Countries differ in how they punctuate numbers, and you can reflect these differences in your reports using Continental Decimal Notation (CDN) which is specified with the CDN SET parameter. The CDN SET allows you to choose to punctuate numbers with a combination of commas, decimals, spaces, and single quotation marks.

The CDN SET parameter can be used in a report request but is not supported in DEFINE or COMPUTE commands.

Note: The punctuation specified by the CDN parameter also determines the punctuation used in numbers affected by the CENT-ZERO SET parameter.

Syntax: **How to Determine the Punctuation of Large Numbers**

```
SET CDN = option
```

where:

option

Determines the punctuation used in numeric notation. The options are:

- **ON,** which uses CDN. For example, the number 3,045,000.76 is represented as 3.045.000,76.
- **OFF,** which turns CDN off. For example, the number 3,045,000.76 is represented as 3,045,000.76. This value is the default.
- **SPACE,** which separates groups of three significant digits with a space instead of a comma, and marks a decimal position with a comma instead of a period. For example, the number 3,045,000.76 is represented as 3 045 000,76.
- **QUOTE,** which separates groups of three significant digits with a single quotation mark (') instead of a comma, and marks a decimal position with a comma instead of a period. For example, the number 3,045,000.76 is represented as 3'045'000,76.
- **QUOTEP,** which separates groups of three significant digits with a single quotation mark (') instead of a comma, and marks a decimal position with period. For example, the number 3,045,000.76 is represented as 3'045'000.76.

Example: **Displaying Numbers Using Continental Decimal Notation**

The following table shows how 1234.56 is displayed, depending on the setting of CDN.

CDN Setting	Result
OFF	1,234.56
ON	1.234,56
SPACE	1 234,56
QUOTE	1'234,56

CDN Setting	Result
QUOTEP	1'234.56

Example: **Determining the Punctuation of Large Numbers**

In the following request, CDN is set to ON which punctuates numbers using a period to separate thousands, and a comma to separate decimals.

```
SET CDN = ON
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME CURR_SALEND
```

The output is:

```
LAST_NAME       FIRST_NAME        CURR_SAL
---------       ----------        --------

STEVENS         ALFRED          $11.000,00
SMITH           MARY            $13.200,00
JONES           DIANE           $18.480,00
JONES           DIANE           $17.750,00
BANNING         JOHN            $29.700,00
IRVING          JOAN            $26.862,00
IRVING          JOAN            $24.420,00
ROMANS          ANTHONY         $21.120,00
MCCOY           JOHN            $18.480,00
BLACKWOOD       ROSEMARIE       $21.780,00
MCKNIGHT        ROGER           $16.100,00
MCKNIGHT        ROGER           $15.000,00
CROSS           BARBARA         $27.062,00
CROSS           BARBARA         $25.775,00
```

Selecting an Extended Currency Symbol

You can select a currency symbol for display in report output regardless of the default currency symbol configured for National Language Support (NLS). Use the extended currency symbol format in place of the floating dollar (M) or non-floating dollar (N) display option. When you use the floating dollar (M) or non-floating dollar (N) display option, the currency symbol associated with the default code page is displayed. For example, when you use an American English code page, the dollar sign is displayed.

Note: You can use the SET CURRSYMB command to control which symbol displays for the M and N options.

The extended currency symbol format allows you to display a symbol other than the dollar sign. For example, you can display the symbol for a United States dollar, a British pound, a Japanese yen, or the euro. Extended currency symbol support is available for numeric formats (I, D, F, and P).

Use the following character combinations as the final two characters in any numeric display format:

Display Option	Description	Example
!d	Fixed dollar sign.	D12.2!d
:d	Fixed dollar sign.	D12.2:d
!D	Floating dollar sign.	D12.2!D
:D	Floating dollar sign.	D12.2:D
!e	Fixed euro symbol.	F9.2!e
:e	Fixed euro symbol.	F9.2:e
!E	Floating euro symbol on the left side.	F9.2!E
:E	Floating euro symbol on the left side.	F9.2:E
!F	Floating euro symbol on the right side.	F9.2!F
:F	Floating euro symbol on the right side.	F9.2:F
!l	Fixed British pound sign.	D12.1!l
:l	Fixed British pound sign.	D12.1:l
!L	Floating British pound sign.	D12.1!L
:L	Floating British pound sign.	D12.1:L
!y	Fixed Japanese yen symbol.	I9!y

Display Option	Description	Example
:y	Fixed Japanese yen symbol.	I9:y
!Y	Floating Japanese yen symbol.	I9!Y
:Y	Floating Japanese yen symbol.	I9:Y

Note: The colon (:) is equivalent to the exclamation point (!), however, only the colon is invariant across code pages, so using the colon is recommended.

Reference: **Extended Currency Symbol Formats**

The following guidelines apply:

- A format specification cannot be longer than eight characters.
- The extended currency option must be the last option in the format.
- The extended currency symbol format cannot include the floating (M) or non-floating (N) display option.
- A non-floating currency symbol is displayed only on the first row of a report page. If you use field-based reformatting (as in the example that follows) to display multiple currency symbols in a report column, only the symbol associated with the first row is displayed. In this case, do not use non-floating currency symbols.
- Lowercase letters are transmitted as uppercase letters by the terminal I/O procedures. Therefore, the fixed extended currency symbols can only be specified in a procedure.
- Extended currency symbol formats can be used with fields in floating point, decimal, packed, and integer formats. Alphanumeric, dynamic, and variable character formats cannot be used.

Chapter 9

Designing Windows With Window Painter

The following topics describe how to create FOCUS menus and windows that work with FOCEXECs.

In this chapter:

- Introduction
- Window Files and Windows
- Integrating Windows and the FOCEXEC
- Tutorial: A Menu-Driven Application
- Window Painter Screens
- Transferring Window Files

Introduction

FOCUS Window Painter is a tool that helps you design and create your own menus and screens for attractive and easy-to-use applications.

Many window types and features are available, and you can implement horizontal menus and multi-input windows as part of your FOCUS application. Horizontal menus can also have pull-down menus associated with each menu item.

You can perform a string search in an active window by entering any pattern followed by a blank and pressing *Enter*. Within the pattern:

- An asterisk (*) is a multiple character wildcard.
- A question mark (?) is a single character wildcard.
- An equal sign (=) repeats the last string.

FOCUS tries to locate the line matching the pattern starting from the line following the current line. The search concludes at the line preceding the current line. If no match is found, a beep sounds and the cursor remains at the current position.

The windows you can design with FOCUS Window Painter look just like the menus and screens you see in the FOCUS Talk Technologies, such as TableTalk and PlotTalk, but you can customize each to fit your application. You can design user-friendly menus and display convenient and eye-catching instructions onscreen.

FOCUS Window Painter itself guides you step by step, using windows like those you created.

On the windows you create, you can prompt users to:

- Select menu items from a list.
- Enter data.
- Select from automatically generated lists of available files and field names.
- Register a choice by pressing a function key.

You can also simply display explanations and instructions.

Window Painter is flexible enough to design the many different types of windows you might need for any application written with FOCUS.

You can also upload window files from FOCUS running in one operating environment, such as PC/FOCUS, and edit them using Window Painter for use on another operating environment, such as z/OS.

How Do Window Applications Work?

Window Painter stores the windows you design in window files. Window files work in conjunction with FOCEXEC procedures that use Dialogue Manager.

There are two major parts in any window application, each of which is a step for the developer:

- The windows, created with Window Painter, which users see.
- The Dialogue Manager FOCEXEC.

You can invoke Window Painter to create and edit windows by typing

```
WINDOW [PAINT]
```

at the FOCUS prompt, and pressing *Enter*.

You can invoke the Window facility in your FOCEXEC by including the Dialogue Manager command -WINDOW in the FOCEXEC. The -WINDOW command provides the name of the window file, and the name of the individual window that should be displayed first. When the -WINDOW command is executed by Dialogue Manager, control in the FOCEXEC passes to the Window facility.

The user is moved through the window file by goto values. A goto value tells the Window facility which window to display next.

You specify goto values when creating the windows with Window Painter. When your window is a menu with several items, you may assign a different goto value for each menu item, so that the next window depends on the user's selection.

When you create the windows, you also specify return values. As with goto values, you may assign a different return value to each item on a menu. Return values are collected as the user moves through the windows, and are substituted for "amper variables" which can be used later in the window file or in the FOCEXEC when control passes back. (Amper variables are Dialogue Manager variables of the format &variablename.)

When the selected value is inserted in the FOCEXEC, you may test it with a Dialogue Manager IF...THEN command and branch accordingly to a label in the FOCEXEC. In this way, you move the user through a series of windows, collecting return values for amper variables, using only one command in your FOCEXEC.

You can use windows to collect amper variable values in place of any other method of prompting available through Dialogue Manager.

For a complete discussion of the Dialogue Manager facility, see *Managing Flow of Control in an Application* on page 201. For details of integrating a FOCEXEC with the Window facility using return and goto values, see *Integrating Windows and the FOCEXEC* on page 469.

Window Files and Windows

Windows—that is, menus and screens—are stored in window files. Windows are included in a specified window file as you create and save them during a Window Painter session.

Window files are contained in a partitioned data set (PDS) allocated to ddname FMU. Before any window files can be created, a PDS must be created and ddname FMU must be allocated to it.

Note, however, that creating a PDS is not necessary if you are creating window files to be used only in the current FOCUS session: Window Painter temporarily allocates the PDS. For a full description of allocation requirements, see the appropriate *Guide to Operations* topic in the *FOCUS Overview and Operating Environments* manual

A window file can contain a maximum of 384 windows, and a number of windows may be displayed on the screen at once. All the windows in a single application may be stored together in one window file, or you may create separate window files for different parts of the application such as Help Windows.

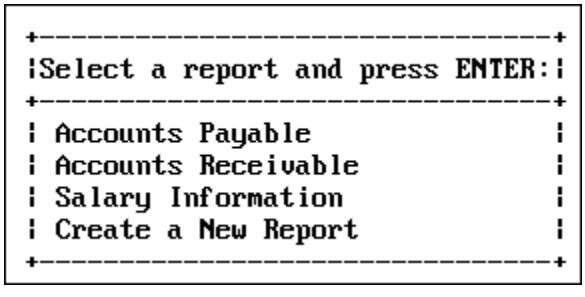
You can make an application more attractive by presenting menus in windows containing titles and other design elements, and can make an application easier to use by displaying function key definitions or other useful information.

Types of Windows You Can Create

Window Painter creates 10 different types of windows, each with its own special uses. These windows are described in the following topics.

Vertical Menus

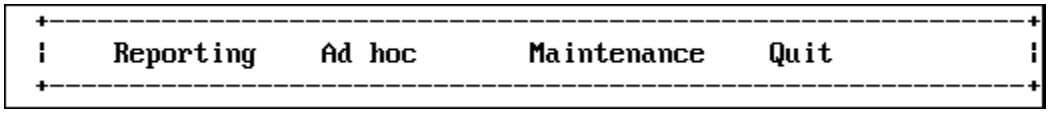
This is a *vertical* menu:



A menu is a window that lets users select an option from a list. These options are called menu items. A vertical menu lists its menu items one below the other. A user can select an item by moving the cursor down the list with the arrow keys and pressing *Enter* when the cursor is on the line of the desired item. A user can select more than one item if the window includes the Multi-Select option, which is part of the Window Options Menu. Help information can be specified for each item in the menu by using the menu-item help feature of help windows. For additional information on Multi-Select and Help windows, see *Window Options Menu* on page 509.

Horizontal Menus

This is a *horizontal* menu:



A horizontal menu displays its menu items on a line, from left to right. You select an item by using *PF11* or the Tab key to move right and *PF10* or Shift+Tab to move left across the line, and pressing *Enter* when the cursor is at the desired item. You can also select an item by employing the search techniques available for FOCUS windows. (Search techniques are not available with pull-down windows).

If you use *PF11* at the last item on the menu, the cursor moves to the first item on the menu. If you use *PF10* at the first item on the menu, the cursor moves to the last item on the menu, unless there is another screen to scroll to.

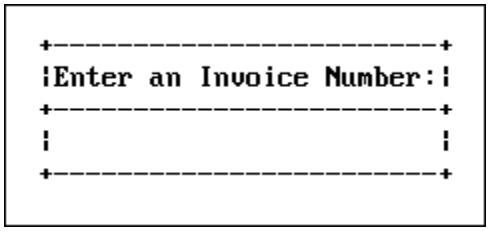
An application can display an associated pull-down menu for an item on a horizontal menu when the cursor is on that item. Choose the pull-down option from the Window Options menu as discussed in *Creating Windows* on page 461. An option to display descriptive text above or below the horizontal menu is also available from the Window Options menu.

You can assign any return value to each item on the menu. When you select a menu item, the corresponding return value is collected.

In a horizontal or vertical menu, you can assign a goto value to each menu item.

Text Input Windows

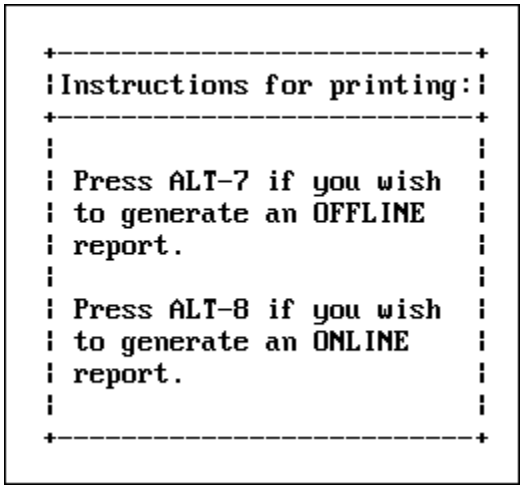
This is a *text input* window:



Amper variables can be used in a Windows application. A text input window prompts the user to supply information needed in a FOCEXEC. It is also possible to display an existing value to be edited. Each text input window accepts one line of input up to 76 characters long. You assign the length and format of the field when you create the window. Additional information about creating a text input window is found in *Window Creation Menu* on page 504.

Text Display Windows

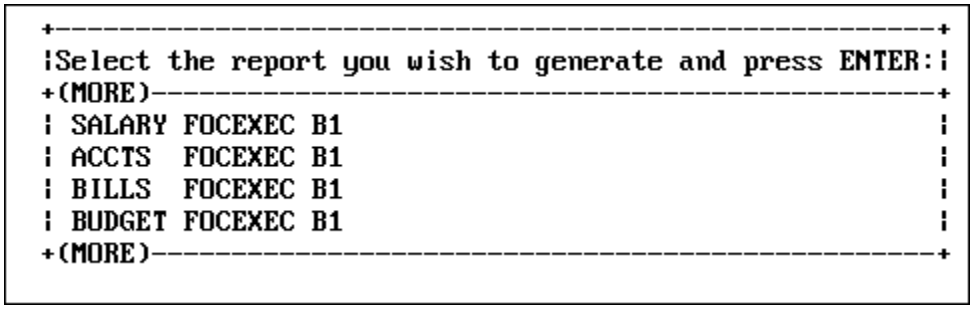
This is a *text display* window:



A text display window lets you present information such as instructions or messages. No selections can be made from a text display window, and no data can be entered in it.

File Names Windows

This is a *file names* window:

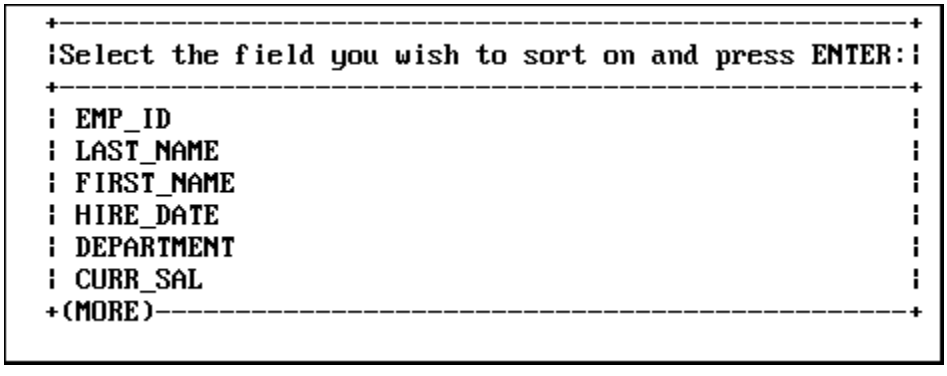


A File Names window presents a list of names of up to 1023 PDS members. The user can select one of these names by moving the cursor and pressing *Enter* when the cursor is on the line of the desired file name. You can specify selection criteria for the displayed file names when the window is created. A user can select more than one file if the window includes the Multi-Select option, which is available on the Window Options Menu.

Note that the maximum number of file (or member) names which can be displayed decreases as the width of the window increases. Narrow windows can display a greater number of names.

Field Names Windows

This is a *field names* window:



A field names window presents a list of all field names from a Master File; the user can select one by moving the cursor and pressing *Enter* when the cursor is on the line of the desired field name. A user can select more than one field if the window includes the Multi-Select option, which is available on the Window Options Menu.

You can use a field names window as the next step after a file names window. That way, you can present a selection of files first, followed by the fields in a selected file.

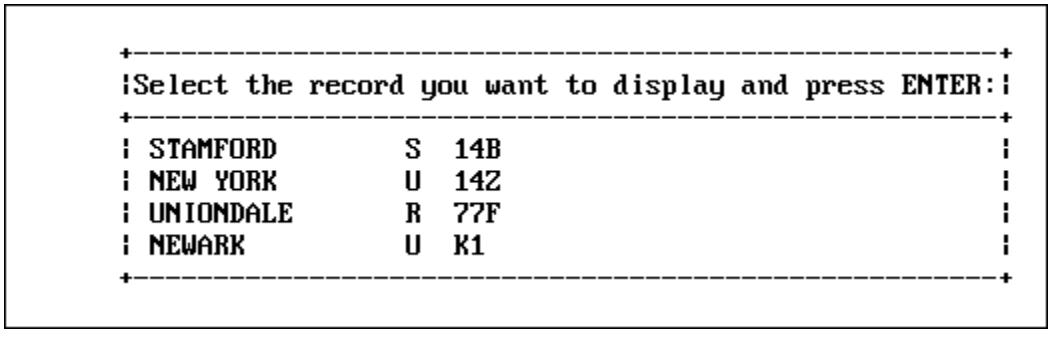
The field names are qualified when duplicates exist. You can use *PF10* and *PF11* to scroll left and right if a field name exceeds the maximum number of characters allowed on a line in a data field window.

Use *PF6* as a three-way toggle to sort the fields in one of the following ways:

1. Display field names in the order in which they appear in the Master File.
2. Display field names in alphabetical order.
3. Display the fully qualified field names in the order in which they appear in the Master File.

File Contents Windows

This is a *file contents* window:



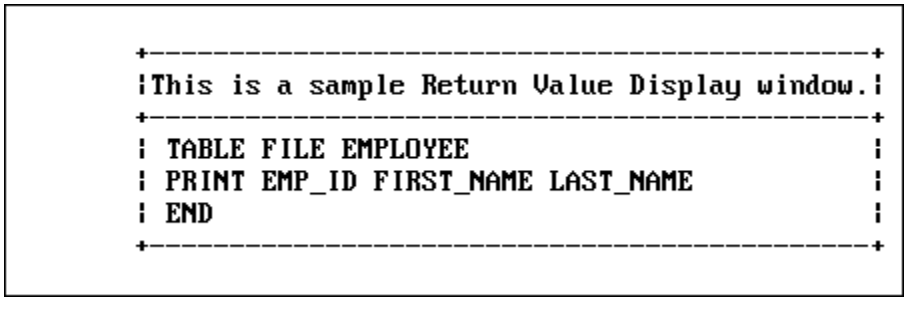
The file contents window displays the contents of a file. There is no limit on the size of a file contents window. The user can select a line of contents by moving the cursor to it and pressing *Enter*. Each line can be up to 77 characters long. A user can select more than one line if the window includes the Multi-Select option, which is described as part of the Window Options Menu in *Window Options Menu* on page 509.

The contents of any member of a PDS (except as noted below) can be displayed. Sequential files can also be displayed in TSO. You are prompted for a file name (the ddname) and a file type (the member name). This information should be entered as "member name ddname".

Note: You cannot display a file with unprintable characters in a file contents window. This includes files such as FOCUS files, HOLD files, SAVB files, FOCCOMP files, and encrypted files.

Return Value Display Windows

This is a *return value display* window:



The return value display window displays amper variables that have been collected from other windows. No selections can be made from a return value display window, and no data can be entered into it.

Return value display windows are very useful for constructing a command (or any string of words or terms) by working through a series of windows. An example of this type of application is seen when you construct a TABLE request using TableTalk.

Each line of the return value display window is stored in a variable called &*windownamexx*, where *windowname* is the name of the window and *xx* is a line number.

Unless you use the Line-break option to place return values on separate lines, all collected return values are placed on the same line until the end of the line is reached. The length of the line is determined by the size of the window created. A description of the Line-break option on the Window Options Menu can be found in *Window Options Menu* on page 509.

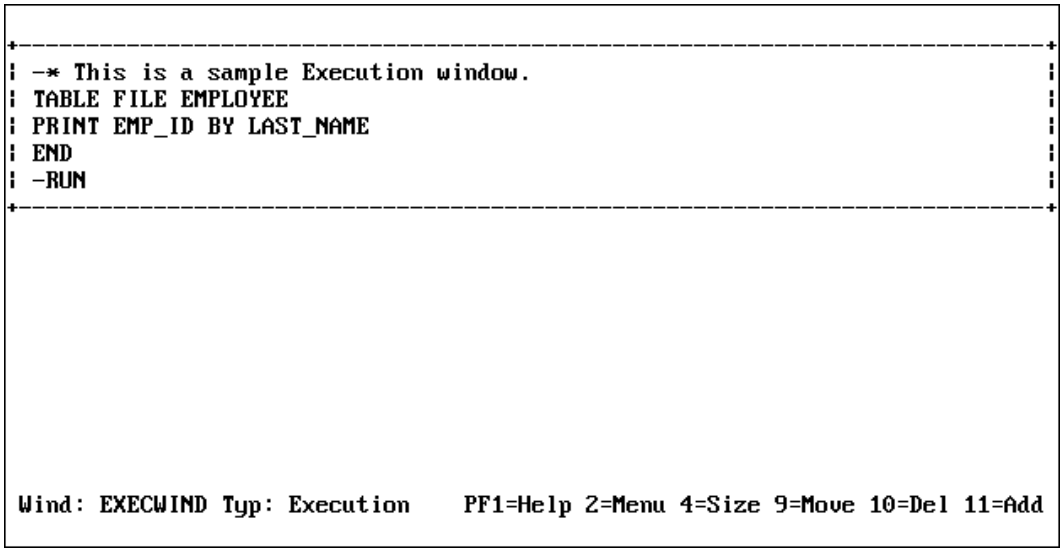
Only one return value display window may be displayed at a time on the screen. It collects a value from any active window (that is, a window from which a selection is being made or to which text is being entered, or an active text display window) if it is on that window's display list. A description of the Display lists option on the Window Options Menu can be found in *Window Options Menu* on page 509.

You can clear the collected values from a return value display window by including it on the hide list of a window that is being used. A description of the Hide lists option on the Window Options Menu can be found in *Window Options Menu* on page 509.

For a Multi-Select window, the return value display window gives the number of selections, not the values selected. The values can be retrieved by using the -WINDOW command with the GETHOLD option.

Execution Windows

This is an *execution* window:



The execution window contains FOCUS commands such as Dialogue Manager commands, and TABLE requests.

You can create an execution window by choosing its option on the Window Creation menu.

When this window is first displayed, it has a width of 77 characters, and no heading. You can place FOCUS commands within it. Note that the commands in an execution window appear just as you type them; commands are not automatically converted to uppercase.

The Window Painter Main Menu contains an option that enables you to run a window in order to see any return values collected. If you were to run (not execute) the execution window from the Window Painter Main Menu, you would see the execution window contents, then any windows called, and finally any return values collected by running the windows.

Note the following rules when using execution windows:

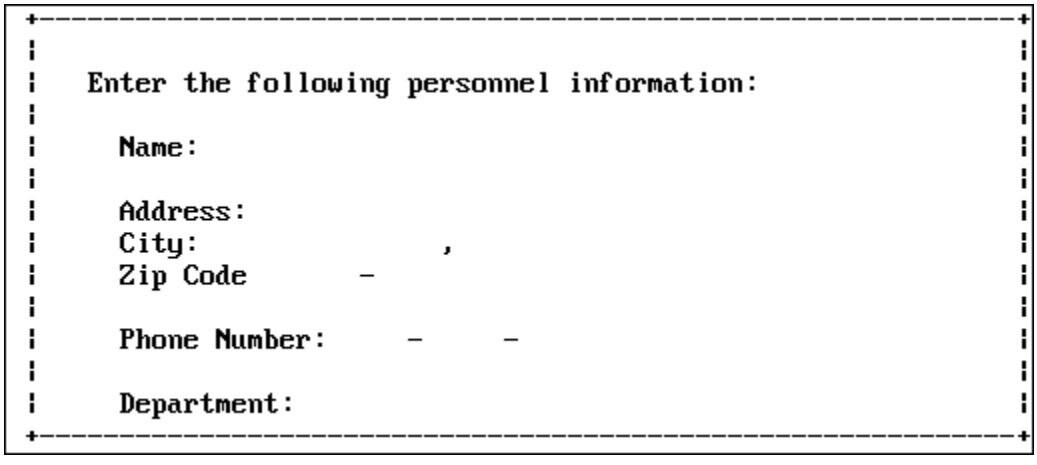
- When you GOTO an execution window, the contents of the window are executed. In all cases, execution begins at the top of the window.
- An execution window is not displayed when executed, although the commands it contains may generate a display.
- An execution window can use an amper variable as a goto value.

- An execution window clears the screen and the Return Value display window.
- Execution windows have no return values.
- Execution windows can contain up to 22 lines.
- Execution windows can use local variables.
- Goto values for execution windows should be assigned at line 1.
- Windows called from within execution windows preempt window goto values. For example, a -WINDOW command issued from within an execution window preempts an assigned goto value.
- The FOCUS commands within an execution window follow normal Dialogue Manager execution (that is, FOCUS commands are stacked, Dialogue Manager commands are executed immediately). Any windows called from the execution window follow the logic determined by the windows themselves. This substantially affects the application's transfer of control.
- Use -RUN for immediate execution; otherwise requests are performed after leaving the window application.

Normally, FOCUS returns to the window designated by the assigned goto value after the contents of the execution window have been executed. However, when a jump is made to a window from inside an execution window, the commands in the execution window following the jump are skipped (along with any attached gotos). This differs from initiating a window from inside Dialogue Manager, which when finished returns you to the command following the GOTO.

Multi-Input Windows

This is a *multi-input* window:



A multi-input window prompts you for information used in the application. A multi-input window may include up to 50 input fields, each of which can be up to 76 characters long. You assign the length, name, and format of the field when you create the window.

Use the Tab key to move the cursor between the fields on a multi-input window.

You can supply help information for each field in a multi-input window by using the Help window option. For information on Help windows, see *Window Options Menu* on page 509.

For a multi-input window, the return value is the name of the input field occupied by the cursor when you pressed *Enter* or a function key. The name that you supply for each input field is assigned to an amper variable with the same name as the field (each input field has a unique name). The variable &WINDOWVALUE contains the value of the input field occupied by the cursor when you pressed *Enter* or a function key.

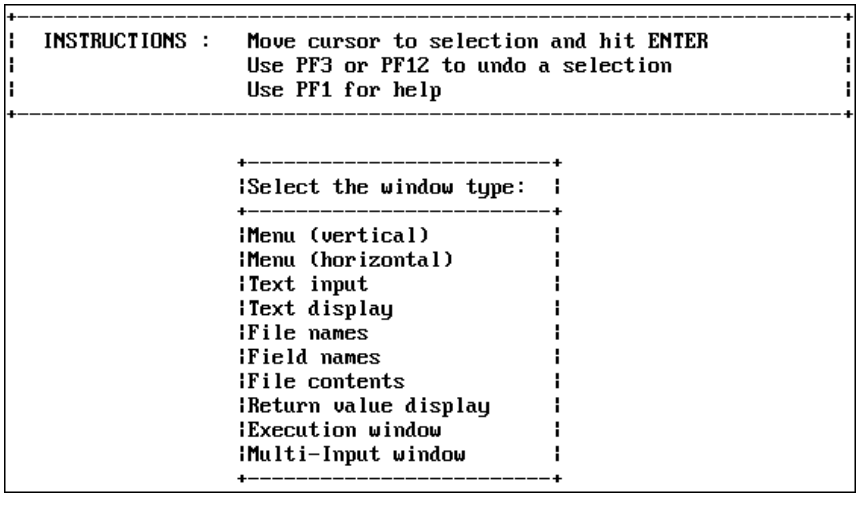
Use a unique name for each field on a multi-input window. To display the field names specified, use the Input Fields option on the Window Options menu.

Creating Windows

The process of creating windows begins with choosing the type of window you want to create from the Window Creation menu. Each type of window requires slightly different instructions. The tutorial in *Tutorial: A Menu-Driven Application* on page 478 describes how to create and implement text display window, vertical menu, and file names windows. This topic describes how to create horizontal menus (with or without associated pull-down menus) and multi-input windows.

Creating a Horizontal Menu

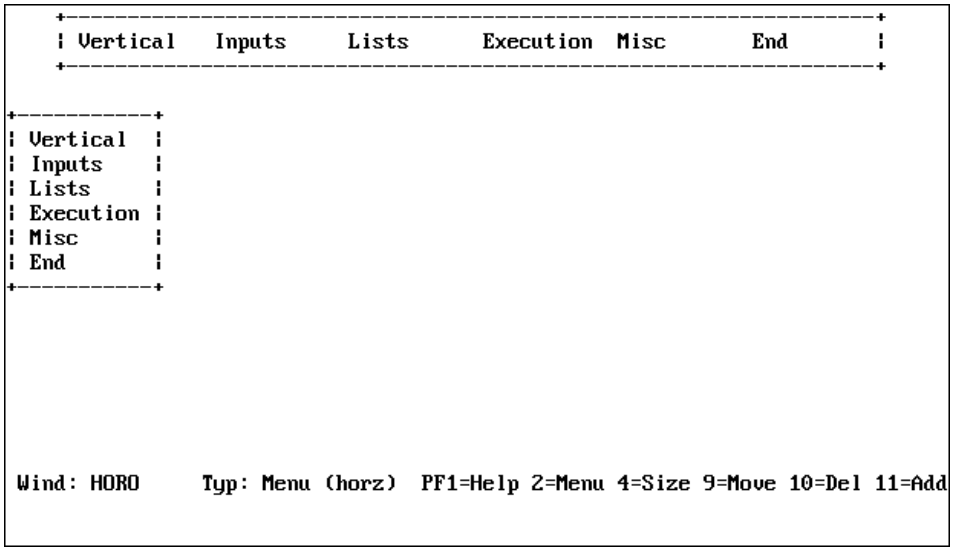
To create a horizontal menu, begin by placing the cursor at the *Menu (horizontal)* option on the Window Creation menu:



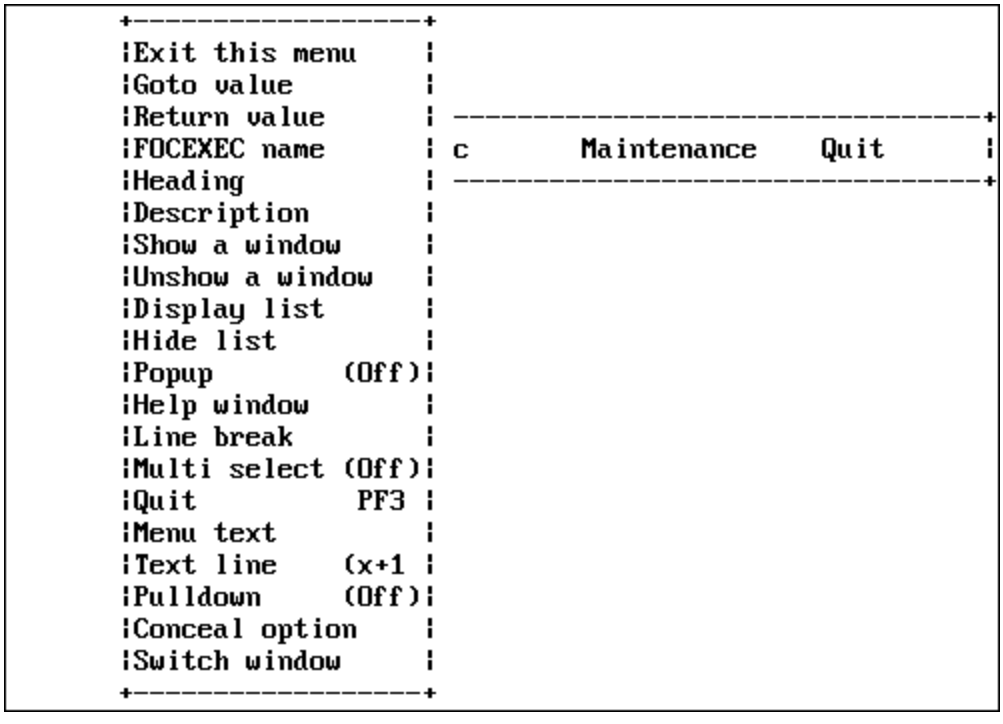
You are prompted to enter a name and brief description for the window, after which you reach the creation screen. On this screen:

1. Move the cursor to the location in which you want the top left corner of the menu to be displayed. Press *Enter*.
2. Next, use the arrow keys to move the cursor down (enough spaces to leave a line for each item you want to display as a menu choice) and to the right (enough spaces to just fit the longest menu item). Press *PF4*. You see two windows: one is for entering information and the other is the corresponding horizontal menu.
3. Enter the menu items in the window containing the cursor. Press *Enter* after each item; the item automatically appears on the horizontal menu.

The following is an example of a completed creation screen:



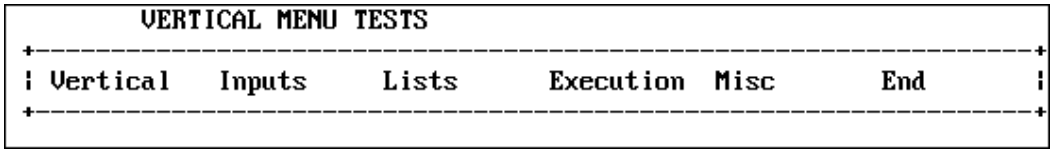
Once you have entered the items on your menu, there are several options you can select for each item. Move the cursor to any item and press *PF2* to display the Window Options menu:



Position the cursor on any option you want to select and press *Enter*.

Two features available for horizontal menus are Menu text and Text line. Menu text is a line of text displayed when the cursor is on a menu item. The line on which the text is displayed is called the text line. You can position the text line one or two lines either above or below the horizontal menu.

The following example illustrates Menu text and Text line. When the cursor is positioned on Vertical in the example below, the following is displayed:



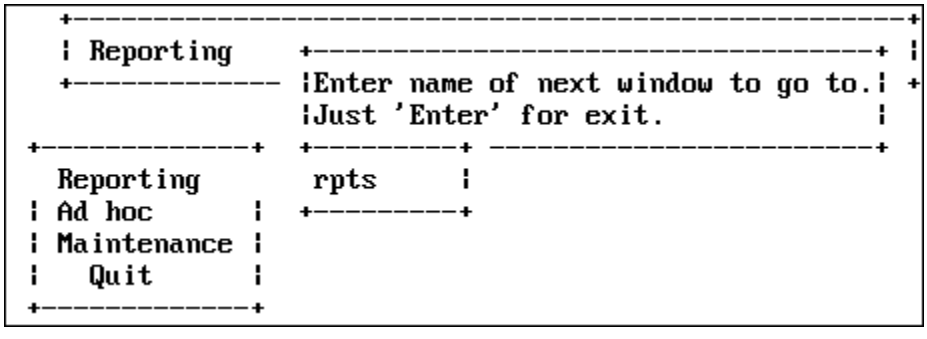
In this example, the Menu text VERTICAL MENU TESTS is positioned at Text line x-1, one line above the menu. To place the Text line two lines above the Menu text, change x-1 to x-2. For Text lines below the menu text, use x+1 or x+2.

You can also select the Pull-down option for a horizontal menu. With this option, you can assign a pull-down menu to be displayed for a horizontal menu item whenever the cursor is positioned on that item.

Pull-down Menus

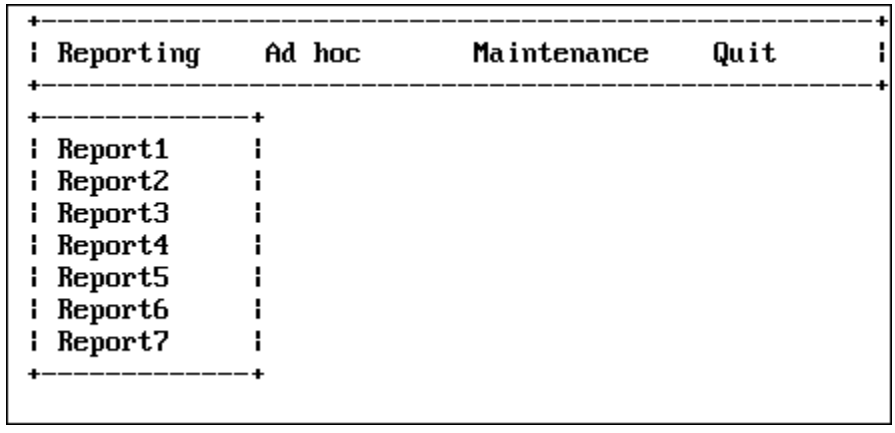
When you set the Pull-down option ON, you can display an associated pull-down menu for an item in a horizontal menu by positioning the cursor on that item. The default is OFF. To change the setting to ON, position the cursor on the Pull-down option and press *Enter*. Note that when Pull-down is set ON, Menu Text is automatically set OFF.

The associated pull-down menu must be a vertical menu. When creating the horizontal menu, you must assign a Goto value to point to the pull-down menu. To do so, position the cursor on the goto value, press *Enter*, and enter the name of the pull-down menu you want to display in the space provided:

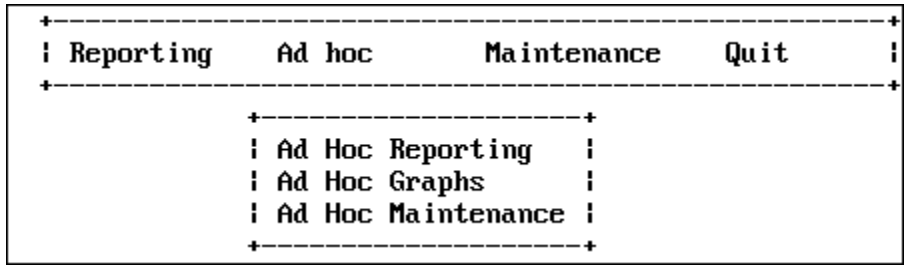


You must create the vertical menu, rpts, as you would any other vertical menu. See *Tutorial: A Menu-Driven Application* on page 478 for examples.

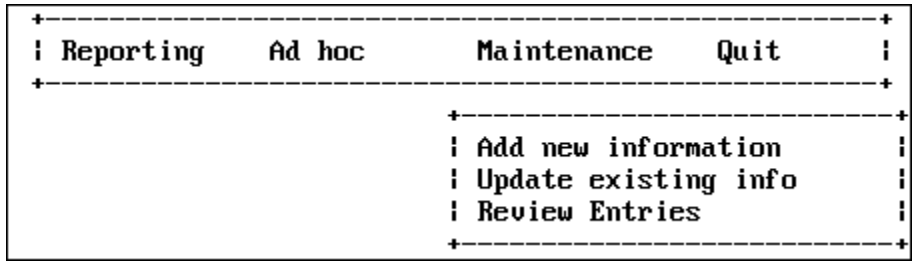
The following example shows a horizontal menu with the Reporting pull-down menu displayed:



The following screen shows the same menu with the Ad hoc pull-down menu displayed:



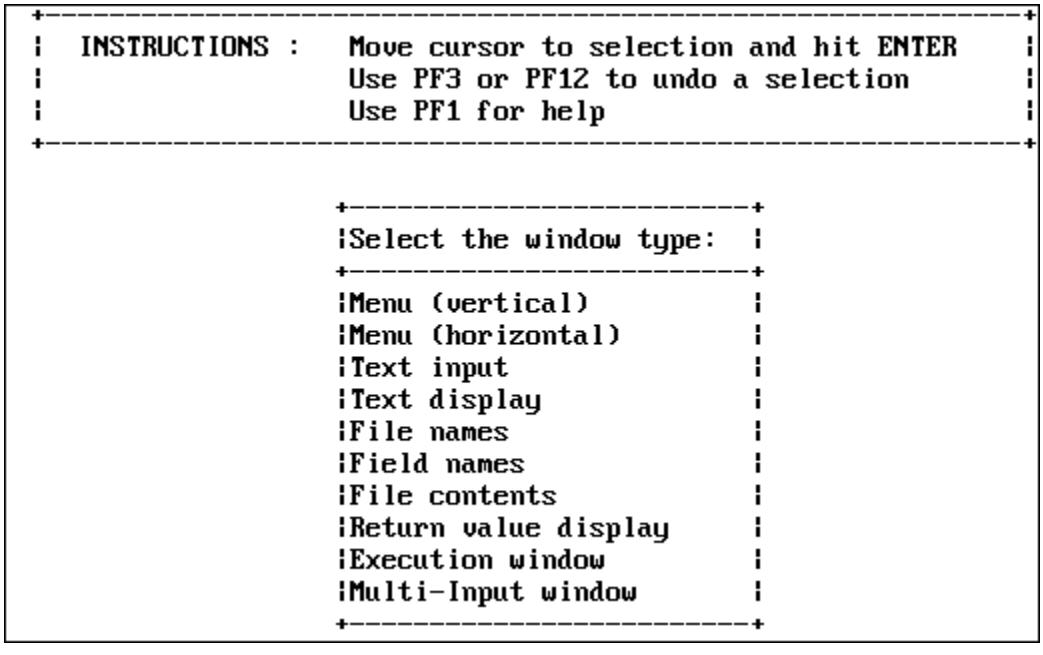
The following screen shows the same menu with the Maintenance pull-down menu displayed:



Note: To move from item to item in a horizontal menu, use *PF10* and *PF11*.

Creating a Multi-Input Window

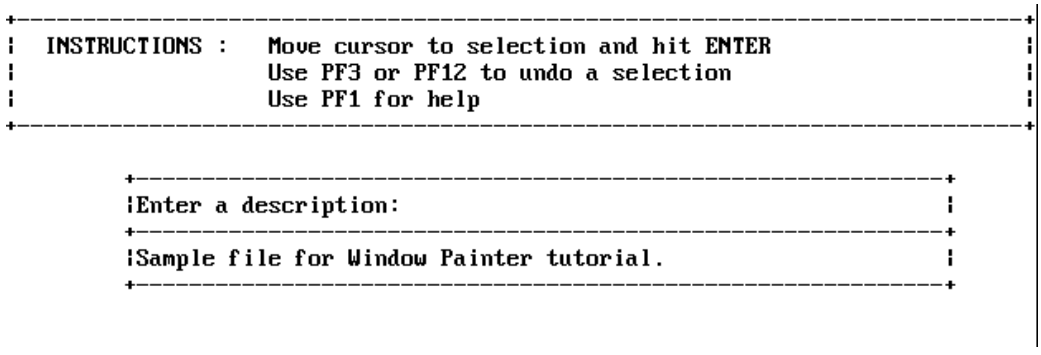
To create a multi-input window, begin by placing the cursor at the *Multi-Input window* option on the Window Creation menu and press *Enter*. You are then prompted for a name, description and heading. Place the window on the screen and size it as desired.



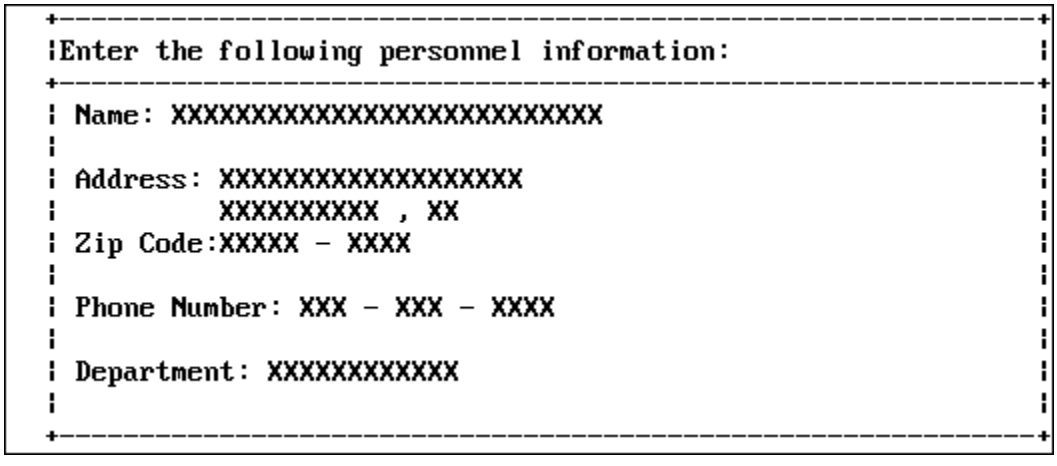
To place entries on the window:

1. Type the text for display.
2. Press *PF6* at the point where the field begins.
3. Space along for the length of the field.
4. Press *PF6* again to signify the end of the input area.
5. Enter name and information for the field.

The following example shows a multi-input window, with *Name:* entered as display text.



This is what the developer's screen looks like after several fields have been included in the multi-input window:



Note: Text fields may be supplied without headings or instructions. For example, see the city and state portion of the address line.

This is how the window appears when run as part of the application:

```
+-------------------------------------------------------------+
|Enter the following personnel information:                   |
+-------------------------------------------------------------+
| Name:                                                       |
|                                                             |
| Address:                                                    |
|                    ,                                        |
| Zip Code       -                                            |
|                                                             |
| Phone Number:     -     -                                   |
|                                                             |
| Department:                                                 |
|                                                             |
+-------------------------------------------------------------+
```

The following screen shows what is returned from the window when it is run inside the Window Painter:

```
+-------------------------------------------------------------+
|Variable      Value                                          |
+-------------------------------------------------------------+
|&WINDOWNAME   MULTI                                          |
|&WINDOWVALUE                                                 |
|&MULTI        NAME                                           |
|&NAME                                                        |
|&STREET                                                      |
|&CITY                                                        |
|&STATE                                                       |
|&ZIP1                                                        |
|&ZIP4                                                        |
|&AREA                                                        |
|&EXCHANGE                                                    |
|&NUMB                                                        |
|&DEPARTMENT                                                  |
|&PFKEY        ENTR                                           |
|&RETCODE             0                                       |
+-------------------------------------------------------------+
```

Note: To move from field to field in a multi-input window, use the Tab key.

Integrating Windows and the FOCEXEC

The windows created with Window Painter are designed for use within an application FOCEXEC. This topic discusses how to integrate the windows into your FOCEXEC.

Syntax: How to Invoke the Window Facility

To invoke the Window facility, insert the following Dialogue Manager command in your FOCEXEC

```
-WINDOW windowfile windowname [PFKEY|NOPFKEY] [GETHOLD] [BLANK|NOBLANK]
[CLEAR|NOCLEAR]
```

where:

windowfile

Identifies the file in which the windows are stored. This is a member name. The member must belong to a PDS allocated to ddname FMU.

windowname

Identifies which window in the file to display first. Can be set in Window Painter or in first window displayed. This is optional.

PFKEY

Enables testing for function key values during window execution.

NOPFKEY

Prevents testing for function key values during window execution.

GETHOLD

Retrieves stored amper variables collected from a Multi-Select window. Does not cause window to be displayed.

BLANK

Clears all previously set amper variable values when the -WINDOW command is encountered. This is the default setting.

NOBLANK

No amper variable values are cleared when the -WINDOW command is encountered.

CLEAR

When FOCUS is being used with the Terminal Operator Environment (described in the *Overview and Operating Environments* manual), the -WINDOW command clears the screen before displaying the first window. The Terminal Operator Environment screen is redisplayed when control is transferred from the Window facility back to the FOCEXEC. This is the default setting.

NOCLEAR

When FOCUS is being used with the Terminal Operator Environment, the window file's windows are displayed directly over the Terminal Operator Environment screens.

Note: NOBLANK is particularly important in applications that use more than one -WINDOW command.

Transferring Control in Window Applications

When the -WINDOW command is encountered, control in the FOCEXEC is transferred to the Window facility. Control remains with the Window facility until one of the following occurs:

- The user makes a selection for which you have assigned no goto value.
- The PFKEY option is in effect and the user presses a function key (the function key must be set to RETURN, HX, CANCEL, or END, as described in the *Testing Function Key Values* on page 475.)

Once control passes back to the FOCEXEC, control only returns to the Window facility if another WINDOW command is encountered.

Example: Window File in an Application FOCEXEC

This example shows an application FOCEXEC and a window file named REPORT which contains three windows: R1, R2, and R3.

The numbers at the left of the example refer to the flow of execution (that is, the order in which the commands and windows are executed).

```
1. -START
2. -WINDOW REPORT R1 PFKEY
   -*
3. -*Control is transferred from the above command
   -*to window R1 in window file REPORT.
   -*
4. -IF &PFKEY EQ PF05 GOTO LABEL1;
   -*
   -*Control returns to the above command from
   -*window R2 in window file REPORT.
   .
   .
   -LABEL1
5. -WINDOW REPORT R3
   -*
6. -*Control is transferred from the above command
   -*to window R3 in window file REPORT.
   -*
7. -IF &R3 EQ EXIT GOTO EXIT;
   -*
   -*Control returns to the above command from
   -*WINDOW R3 in window file REPORT.
   .
   .
   -EXIT
```

Note:

- At Step 3, the user selects an option from Window R1. This option's goto value is R2. Control is transferred to Window R2.
- The user presses a function key in Window R2. Control is transferred to the FOCEXEC, to the command following the -WINDOW command (Step 4).
- At Step 6, the user selects the option to exit; no goto value was set for that option. Control is transferred to the FOCEXEC, to the command following the -WINDOW command (Step 7).

The flow of control has certain implications for the design of your window applications:

- Any time you pass control back to the FOCEXEC, the window or menu option must have no goto value, or else must prompt the user to press a function key (as described in *Testing Function Key Values* on page 475).
- At some point in the window session, control should return to the FOCEXEC so that the accumulated return values can be substituted for amper variables, and the variables then used in the FOCEXEC.
- Any time you pass control from the FOCEXEC to the Window facility you must insert the -WINDOW command in the FOCEXEC.

- Note that it is not necessary to create a new window file for each -WINDOW command; you can simply enter the same file again at any window.
- To test for a function key value in the middle of a series of windows, remember that pressing the function key automatically returns control to the FOCEXEC; an -IF test command should follow the -WINDOW command, and a second -WINDOW command should be placed after the -IF command to transfer control back to the window file.
- If you want to clear an existing set of variable values, return control to the FOCEXEC and execute another -WINDOW command with the BLANK option in effect.

To back up a step during window execution, the user may press *PF12* or *PF24*. This does not cause control to pass to the FOCEXEC. However, you can force Dialogue Manager to return control to a FOCEXEC by a PF key setting as described in *Testing Function Key Values* on page 475.

Return Values

When the user responds to your window prompt by entering text, selecting an item from a menu, or pressing a function key, this response is the return value that fills in an amper variable in your FOCEXEC.

There are two ways in which amper variables are most commonly used in FOCEXECs:

- To collect values to plug into a FOCUS procedure such as a TABLE or GRAPH request so it can run.
- To test the value returned in a variable, and branch accordingly to a different part of the FOCEXEC or to another FOCEXEC.

The return value collected can be a character string, a number, the name of a file, a procedure name, or part of a FOCUS command.

A return value amper variable in the FOCEXEC has the same name as the window in which it is collected; that is:

```
&windowname
```

For example, the return value collected by the window MAIN supplies a value for the variable &MAIN.

- In vertical menu and horizontal menu windows, you assign any return value to each item on the menu. If the user selects that option, that return value is collected.
- In text input windows, the return value is the text that the user types.

- In text display windows, you can assign one return value to the entire window. Unlike other return values, a text display window return value is collected as soon as control passes to the window, without the user selecting anything.
- Return value display windows display return values collected from other types of windows. These return values can be displayed one per line, or several together on a single line. Although this type of window does not have a return value, each line has a corresponding amper variable (&*windownamexx*, where *xx* is the line number).
- For a multi-input window, the return value is the name of the input field on which the cursor is positioned when you press *Enter* or a PF key.
- In windows with the Multi-Select option, the return value is the number of items selected.
- In file names, field names, and file contents windows, the return value is, respectively, the file name, field name, or line of file contents that the user selects from the display.

Example: Return Value in a Menu-Driven Application

Assume that you have written a menu-driven application that enables a user to report from any one of a list of files. You have created a series of windows for this application, one of which is a file names window named FILE designed to collect a return value for &FILE. The window displays a list of all the user's files that meet certain file-identification criteria specified when you created the window.

Your FOCEXEC contains these lines:

```
-START
-WINDOW EXAMPLE FILE
.
.
.
TABLE FILE &FILE
```

When the user moves the cursor to SALES and presses ENTER, SALES is collected to be substituted for &FILE in the FOCEXEC:

```
TABLE FILE SALES
```

Goto Values

When creating your windows, you also assign goto values telling the Window facility which window to display next. These values allow you to move the user through a series of windows, collecting return values for amper variables, without adding lines to your FOCEXEC.

- In vertical menu and horizontal menu windows, you assign a goto value for each menu item.

- In all other windows, you assign a single goto value.
- You can use an amper variable as a GOTO value.

As described in *Transferring Control in Window Applications* on page 470, if you assign no goto value to a menu option or window, control passes back to the FOCEXEC when the user selects that option or presses *Enter* at that window.

It is important not to confuse these goto values with the Dialogue Manager -GOTO command. The goto value points your application to a new window in the window file; the -GOTO command transfers control to a label in your FOCEXEC.

Returning From a Window to Its Caller

You can return from a window to its caller via the <ESCAPE> option. If you enter this string as the goto value of a window, control returns to the previous window upon completion of the current window, you must enter the right and left carets as part of the goto value.

Window System Variables

We have already discussed return values: these are specific to each window. Two other Window facility variables, &WINDOWNAME and &WINDOWVALUE, are specific to the -WINDOW session (not to each window) and receive values when the Window facility passes control from a window file back to the FOCEXEC.

&WINDOWNAME

&WINDOWNAME is an amper variable containing the name of the last window that was displayed before the Window facility transferred control back to the FOCEXEC.

This variable can be used in many ways. For example, if the goto values/function key prompts in a window file allow a user to leave the window file from several different windows, you can test &WINDOWNAME in the FOCEXEC to determine which window the user was in last (and, therefore, which path the user navigated through the window file).

&WINDOWVALUE

&WINDOWVALUE is an amper variable containing the return value from the last window that was displayed before the Window facility transferred control back to the FOCEXEC. If the user selected a line for which no return value was set (for example, a blank line between two menu options in a vertical menu window), then &WINDOWVALUE contains the line number of the line that was selected.

This variable can be used in many ways. For example, if the goto values/function key prompts allow a user to leave the window file from several different windows, and you need to know the return value of the last window the user was in before she or he left the file by pressing a function key, you can test &WINDOWVALUE.

Testing Function Key Values

To test for function key values, you must specify the PFKEY option on the -WINDOW command line. When the PFKEY option is set and a user presses a function key during window execution, the name of that key is stored in the amper variable &PFKEY.

For example, if the user presses *PF1*, the 4-character value of &PFKEY is PF01. If *PF2*, the value is PF02, and so forth. If the user presses *Enter*, the value is ENTR. The value of &PFKEY is reset each time the user presses a function key.

Note that if the PFKEY option is specified, the Window facility's default PF key actions are overridden by the general FOCUS PF key settings. This means that when you specify the PFKEY option, if you still want the standard Window facility PF key actions to be available to window users (for example, *PF1* = HELP, *PF3* = UNDO), you must use the SET command in your application FOCEXEC, followed by a -RUN command, to explicitly set those actions.

For example, if you specify the PFKEY option but you want to retain all of the Window facility's default PF key actions using the same PF keys, you need to include the following commands before the -WINDOW command in your application FOCEXEC:

```
SET PF01=HELP
SET PF03=UNDO
SET PF04=TOP
SET PF05=BOTTOM
SET PF06=SORT
SET PF07=BACKWARD
SET PF08=FORWARD
SET PF09=SELECT
SET PF10=LEFT
SET PF11=RIGHT
SET PF12=UNDO
-RUN
```

When you specify the PFKEY option, any PF key which you want to test for in the application FOCEXEC must be set to RETURN. (HX, CANCEL, and END also function as RETURN within the Window facility, and can be used in place of it.)

For example, if you design your application so that a user can press *PF2* to choose an additional menu option, and therefore you want to test &PFKEY for the value PF02 in your application FOCEXEC, then you must include the following SET command before the -WINDOW command in your application FOCEXEC:

```
SET PF02=RETURN
```

The SET PF command is discussed in *Customizing Your Environment*, and in the *Maintaining Databases* manual.

You can list the current general FOCUS PF key settings by issuing the ? PFKEY command. The ? PFKEY command is discussed in *Testing and Debugging With Query Commands*.

The variable &PFKEY can be tested just like any other amper variable. Note that the name of the variable is always &PFKEY; it is not linked to a window name like other amper variables collected through windows.

You may test the PFKEY variable repeatedly throughout the FOCEXEC. Additional SET commands are not required.

One of the advantages of using the &PFKEY variable is that it enables you to collect two return values from a single menu. You might, for example, create a window called FILES, which prompts the user to enter the name of a file, then press *PF7* to produce a graph or *PF8* to produce a report. Both the file name as &FILES and the function key value as &PFKEY would be collected as return values.

It is always important to remember that pressing a function key immediately returns control to the FOCEXEC if that key was set to RETURN (or to HX, CANCEL, or END).

Note: If the cursor is on a menu that has a FOCEXEC associated with it, the FOCEXEC is executed and the GOTO value associated with the menu choice is assumed. The PFKEY is ignored.

In the example above, if the user presses a function key before typing the file name, the &FILES variable is not collected. If the key was set to something other than RETURN, HX, CANCEL, or END, then the action it was set to is invoked, and control remains within the Window facility.

Executing a Window From the FOCUS Prompt

You can execute a window directly from the FOCUS command prompt.

Syntax: How to Execute a Window From the FOCUS Prompt

```
EX 'windowfile FMU' [windowname] [PFKEY|NOPFKEY] [BLANK|NOBLANK]
[CLEAR|NOCLEAR]
```

where:

windowfile

Is the file containing the windows. It must have ddname FMU, and appear within single quotation marks.

windowname

Identifies the first window to be executed. If a window name is not specified, FOCUS executes the default start window, or the first window created.

PFKEY

Tells FOCUS you will test for function key values during execution.

NOPFKEY

Tells FOCUS you will not test for function key values during execution.

BLANK

Clears previously set amper variables when the window is called. This is the default setting.

NOBLANK

Retains previously set amper variables.

CLEAR

When FOCUS is being used with the Terminal Operator Environment, the screen is cleared when the EX command is encountered. The Terminal Operator Environment screen is restored when the last window in the chain has been executed. This is the default setting.

NOCLEAR

When FOCUS is being used with the Terminal Operator Environment, the screen is not cleared when the EX command is encountered, and any windows are displayed within the Terminal Operator Environment screens.

For example, to execute the window MAIN in the window file REPORT, you could issue EX 'REPORT FMU' MAIN from the FOCUS command prompt, which is equivalent to issuing -WINDOW REPORT MAIN from Dialogue Manager.

Tutorial: A Menu-Driven Application

This tutorial describes a menu-driven system that clerical personnel can use to produce sales reports and graphs at your chain of retail stores. The system must fulfill three major requirements:

- **Ease of use.** Your system must let employees be productive without extensive training.
- **Functionality.** The system has to work properly with only a few steps.
- **Appearance.** There should be continuity between screens, and a general unity of design. The reports and graphs produced must be attractive and easy to read.

The application prompts the user to select reporting or creating a graph.

Then, the user may opt to execute an existing FOCUS request or to create a new one. A user who chooses to execute an existing request is shown an automatically generated list of FOCEXECs from which to pick. A user who chooses to create a new request is placed in either TableTalk or PlotTalk, depending on whether reporting or creating a graph was chosen in the first step.

While the report or graph is being generated, a corresponding message is displayed on the terminal screen. And, after the output is displayed, the user can choose to generate another report or graph, or else to exit.

The following figure illustrates the logic of the application FOCEXEC.

```
-START
-WINDOW SAMPLE MAIN
-*
-*Control is transferred from the above command
-*to window MAIN in window file SAMPLE.
-*
-IF &MAIN ...
-*
-*Control returns to the above command
-*from option "Exit?" in window MAIN,
-*from option "New Request?" in window EXECTYPE,
-*and from every selection in window EXECNAME.
-*
.
.
.
-GOTO START
-EXIT
```

Window	If option selected is...	Then go to:
MAIN	Report? Graph?Exit?	window EXECTYPEwindow EXECTYPEback to FOCEXEC
EXECTYPE	Existing Request?New Request?	window EXECNAMEback to FOCEXEC
EXECNAME	The options in this window are a list of report and graph requests from which the user can select.	Control is transferred back to the FOCEXEC.

Creating the Application FOCEXEC

A FOCEXEC called SAMPLE drives this application.

Begin by using the TED editor to create the FOCEXEC file SAMPLE. At the FOCUS prompt, type

```
TED SAMPLE
```

Type in the following FOCEXEC. Note that the numbers on the left refer to explanatory notes. Do not type them in your FOCEXEC file, but read the notes as you go along. All commands that begin with a hyphen, such as -WINDOW, are Dialogue Manager commands, and must begin in the first column. Dialogue Manager is discussed in *Managing Flow of Control in an Application* on page 201.

Notice that this application determines variable values in two ways: there are variables for which values are collected by windows, and variables which are set within the FOCEXEC using the -SET command.

```
-START
1.  -WINDOW SAMPLE MAIN
2.  -IF &MAIN EQ XXIT GOTO EXIT;
    -IF &MAIN EQ RPT GOTO GENERATE;
    -IF &MAIN EQ GRPH GOTO GENERATE;
    -GOTO START
    -****************** GENERATE ********************
3.  -GENERATE
4.  -IF &EXECTYPE EQ EXIST GOTO RPTEX ELSE GOTO NEWRPT;
5.  -RPTEX
6.  EX &EXECNAME
7.  -SET &FORMAT=IF &MAIN EQ RPT THEN REPORT
    -ELSE IF &MAIN EQ GRPH THEN GRAPH;
8.  -TYPE GENERATING &FORMAT
9.  -RUN
10. -GOTO START
11. -NEWRPT
12. -SET &PROCNAME=IF &MAIN EQ RPT THEN TABLETALK
    -ELSE IF &MAIN EQ GRPH THEN PLOTTALK;
13. &PROCNAME
14. -RUN
15. -GOTO START
    -********************** EXIT **********************
16. -EXIT
```

1. The -WINDOW command transfers control to the Window facility. SAMPLE is the name of the window file this application uses and we will create it in this tutorial. MAIN is the window where the procedure begins.

   Control does not return to the next line of the FOCEXEC until a window is processed for which no goto value has been assigned, in this case, EXECTYPE or EXECNAME.

2. The return value collected for &MAIN—collected from the window MAIN—is tested. The FOCEXEC branches to a label depending on its value.

   If the return value for &MAIN is RPT or GRPH, the FOCEXE branches to -GENERATE; if XXIT, to -EXIT. Each return value corresponds to a selection on the menu window MAIN.

3. This label beings to GENERATE section of the FOCEXEC

4. The value collected for &EXECTYPE (from window EXECTYPE) is tested and the FOCEXEC branches accordingly. Note that this value was collected from the window EXECTYPE while the Window facility was in control, without a prompt from Dialogue Manager.

5. This label begins the RPTEX section of the FOCEXEC.

6. The FOCUS command that executes an existing report is stacked. The value of &EXECNAME—the name of the existing report—was collected while the window file was in control. The single quotation marks around &EXECNAME tell FOCUS to treat the value—which may contain more than one word—as part of a single file identification.

7. The value of the variable &FORMAT is set according to the return value from the MAIN window. If the value was RPT, &FORMAT is set to REPORT; if the value is GRPH, &FORMAT is set to GRAPH.

8. A message containing the value of &FORMAT is displayed for the user while the stacked FOCUS request is executing.
9. -RUN executes the stacked command(s).
10. When the request output has been displayed, the FOCEXEC branches back to -START, where the user can choose to exit or to create another report or graph. All amper variable values collected in the previous round are cleared when the -WINDOW command is encountered.
11. This label begins the section NEWRPT.
12. This command sets the value of &PROCNAME to TABLETALK if the value of &MAIN is RPT, to PLOTTALK if the value is GRPH.
13. This line stacks the command TABLETALK or PLOTTALK.
14. -RUN executes the stacked command.
15. This commands returns to -START, as in note 10.
16. This command ends FOCEXEC execution.

Creating the Window File

The -WINDOW command SAMPLE FOCEXEC tells FOCUS to look for a window file named SAMPLE and a window named MAIN. The complete list of windows used in this application is:

BORDER	A text display window used as a background display for the other windows.
BANNER	A text display window that introduces the application.
MAIN	A vertical menu from which the user can choose to create a graph or a report, or exit the application.
EXECTYPE	A vertical menu from which the user chooses to execute an existing procedure or create a new one.
EXECNAME	A file names window displaying all FOCEXEC files, from which the user can select one to execute. This window is seen only if the user opts to execute an existing report in EXECTYPE.

All these windows are included in the window file named SAMPLE. Start by building that window file.

Before you can use Window Painter to create a window file, a PDS must be allocated with ddname FMU, LRECL 4096, and RECFM F. BLKSIZE 4096 is recommended.

You can reach the FOCUS Window Painter Entry Menu by typing

```
WINDOW [PAINT]
```

at the FOCUS prompt, and pressing *Enter*.

The Entry Menu is the first screen you see:

```
 INSTRUCTIONS :   Move cursor to selection and hit ENTER
                  Use PF3 or PF12 to undo a selection
                  Use PF1 for help
```

```
Select the window file:

New File     Create a new file
CPDNTT       CP DN and Timetrack system
```

Since you are creating a new window file, choose NEW FILE, and press *Enter*. The next screen you see prompts you to name the window file.

Since the FOCEXEC looks for a window file named SAMPLE, type

```
SAMPLE
```

and press *Enter*.

```
 INSTRUCTIONS :   Move cursor to selection and hit ENTER
                  Use PF3 or PF12 to undo a selection
                  Use PF1 for help
```

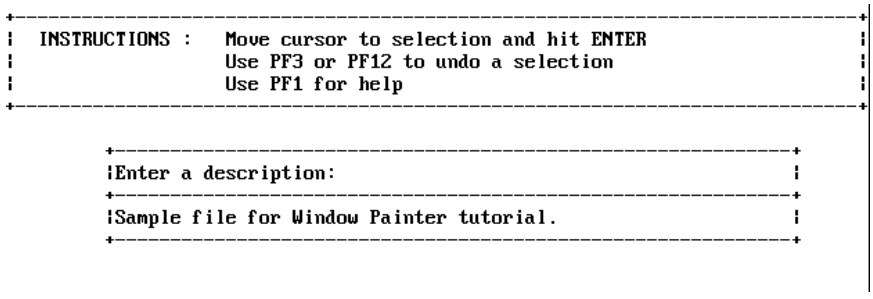
```
Enter the window file name:

SAMPLE
```

A screen appears asking for a description of the window file.

Type

```
Sample file for Window Painter tutorial
```

and press *Enter*.

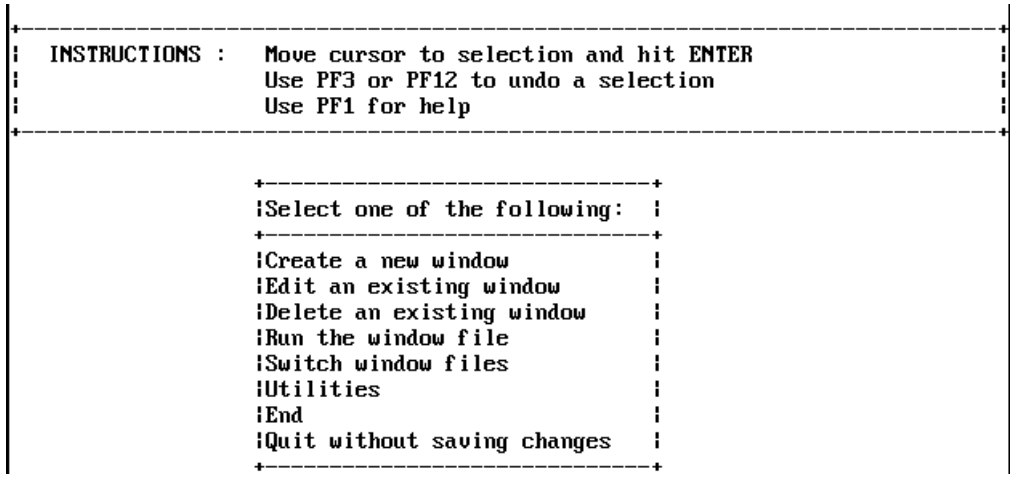


Creating the Text Display Window Named BORDER

Now you are ready to create the first window. The Window Painter Main Menu screen appears. Select

```
Create a new window
```

and press *Enter*.



The Window Creation Menu asks what kind of window you want to create.

```
+-------------------------------------------------------------------------+
|  INSTRUCTIONS :   Move cursor to selection and hit ENTER                |
|                   Use PF3 or PF12 to undo a selection                   |
|                   Use PF1 for help                                      |
+-------------------------------------------------------------------------+

                  +-------------------------+
                  |Select the window type:  |
                  +-------------------------+
                  |Menu (vertical)          |
                  |Menu (horizontal)        |
                  |Text input               |
                  |Text display             |
                  |File names               |
                  |Field names              |
                  |File contents            |
                  |Return value display     |
                  |Execution window         |
                  |Multi-Input window       |
                  +-------------------------+
```

The BORDER window is the first window you create for the application. BORDER supplies a background border for other windows. It is a text display window, so select

```
Text display
```

and press *Enter*.

Next, you are asked to name the window. Type

```
BORDER
```

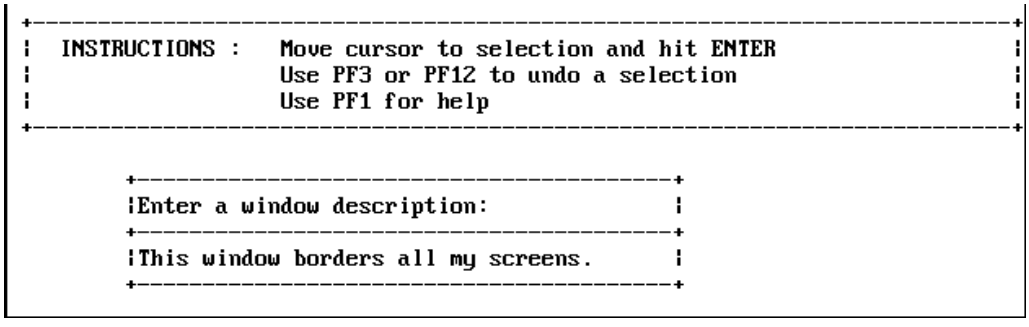
and press *Enter*.

```
+-------------------------------------------------------------------------+
|  INSTRUCTIONS :   Move cursor to selection and hit ENTER                |
|                   Use PF3 or PF12 to undo a selection                   |
|                   Use PF1 for help                                      |
+-------------------------------------------------------------------------+

                        +-----------------------------+
                        |Enter a name for the window: |
                        +-----------------------------+
                        |BORDER                       |
                        +-----------------------------+
```

The Window Description Screen appears next. This description does not appear when the window is displayed, but becomes part of the document file that Window Painter creates describing all windows in the file. Since the document file is very useful when writing your FOCEXEC, it is a good idea to enter a functional description here. To describe this window, type

```
This window borders all my screens.
```

and press *Enter*. The ability to annotate screens in this manner is very useful when selecting windows to edit.



The Window Heading Screen comes next. Since you do not want a heading displayed on this window, simply press *Enter* to bypass it.

The Window Design Screen displayed now is nearly blank, with a cursor for you to position where you want the upper left-hand corner of BORDER to be. Leave the cursor where it is and press *Enter*.

A small box appears around the cursor: this is the window. Make the window larger. Using the arrow keys, move the cursor to the right edge of the screen, on the line just above the status line: this is the new lower right corner of the window. Now press *PF4* to resize the window. (PF4 functions as the SIZE key in the Window Design Screen.) The window has been resized so that its lower right corner is where you positioned the cursor: the window now fills the entire screen.

When resizing a window, remember that the window's lower right corner refers to the lower right corner of the window border, which is shown as a plus sign (+) on the screen. It is this corner that you are moving when you resize the window. On the other hand, the last row of the window refers to the last row that can contain data or text: this is the row immediately above the bottom border.

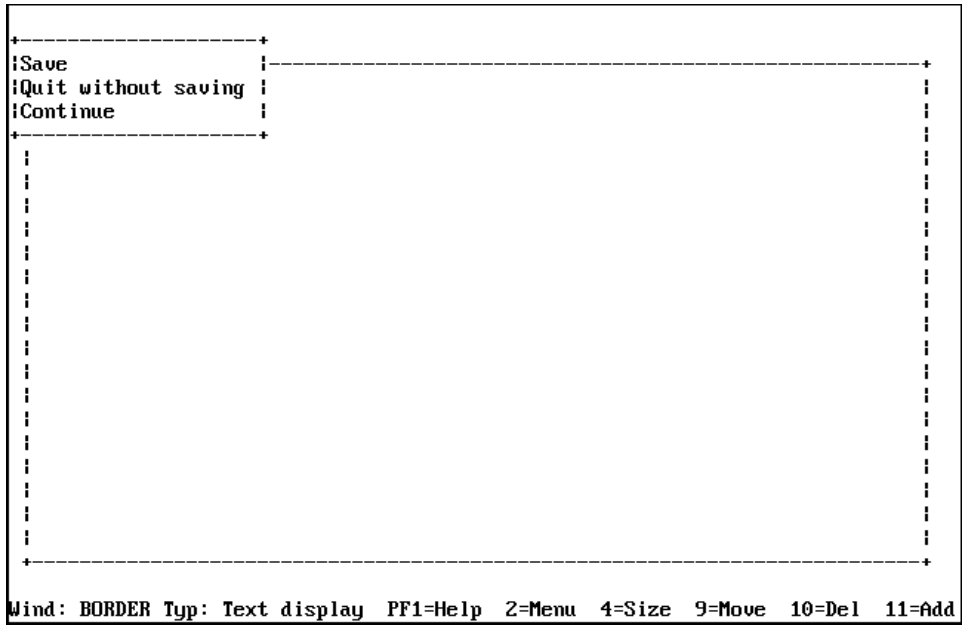
This window's border forms the background border for the other windows in this application.

If you need help using the keyboard while in the Window Design Screen, press *PF1* (the Window Painter Help key) to see the following display:

```
+-------------------------------------------------------------------------+
¦                                                                         ¦
¦          Help: Text display and Return value display windows             ¦
¦                                                                         ¦
¦ Use the arrow keys to move the cursor around on the screen.             ¦
¦ To enter text for a line, simply type that text in the window,          ¦
¦ for text display.                                                       ¦
¦                                                                         ¦
¦ PF01/PF13 : Help.                                                       ¦
¦ PF02/PF14 : Main options menu.                                          ¦
¦ PF03/PF15 : Quit the Menu Design Screen.                                ¦
¦ PF04/PF16 : Resize the window.                                          ¦
¦             If you find that you do not have enough room in the window to ¦
¦             type the text you want, move the cursor to where you want the ¦
¦             new lower-right-hand corner to be, and press PF04 or PF16.  ¦
¦ PF05/PF17 : Set a window to go to if the current line is selected.      ¦
¦ PF06/PF18 : Set a return value for the current line.                    ¦
¦ PF09/PF21 : Move the window.                                            ¦
¦             To move the window, place the cursor where you want the new  ¦
¦             upper-left-hand corner to be, and press PF09 or PF21.       ¦
¦ PF10/PF22 : Delete the line that the cursor is on.                      ¦
¦ PF11/PF23 : Insert a line at the cursor position.                       ¦
¦                                                                         ¦
+-------------------------------------------------------------------------+
```

Press *Enter* to continue.

Now that the window is complete. Press *PF3* and save the window.



Press *Enter* to select Save. You return to the Main Menu.

Creating the Text Display Window Named BANNER

BANNER is also a text display window, but is smaller than BORDER and contains text that identifies this application.

From the Window Painter Main Menu, select

```
Create a new window
```

and press *Enter*. Select

```
Text Display
```

and press *Enter*. The name of this window is

```
BANNER
```

and its description is:

```
Banner for application MAIN menu.
```

Enter this name and description just as you did for the BORDER window. When prompted for a heading, press *Enter*.

At the Window Design Screen, use the arrow keys to move the cursor two spaces to the right, and press *Enter*. Now position the cursor 64 more spaces to the right and two rows down, and press *PF4* to resize the window.

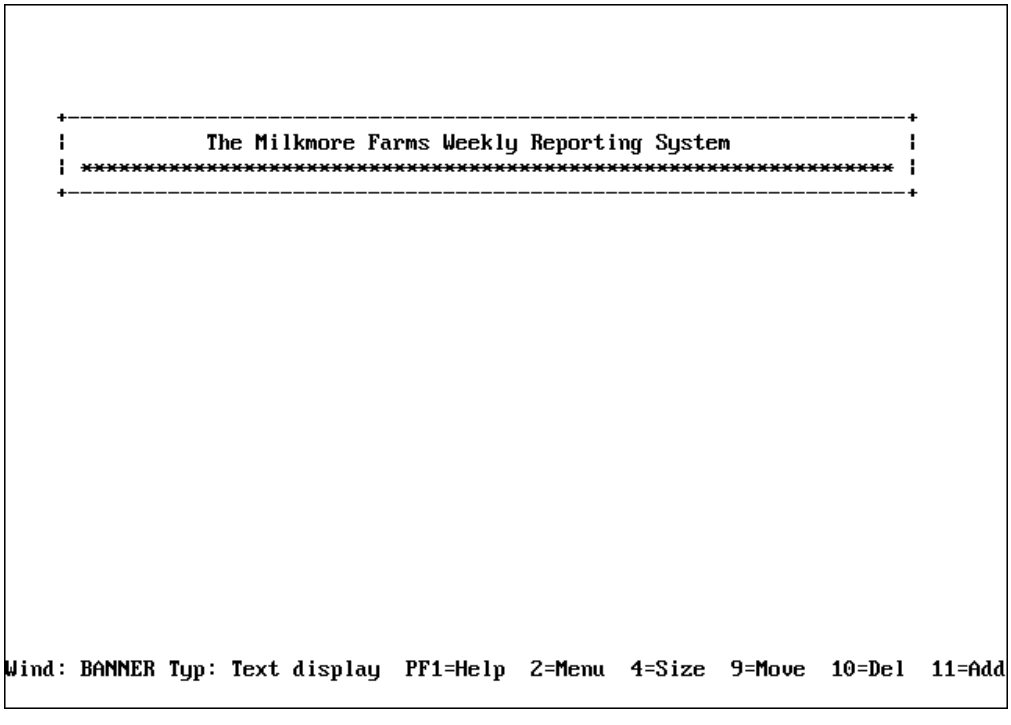
Enter text to be displayed in the window. Reposition the cursor on the first line within the window, 10 spaces to the right of the window's left border, and type:

```
The Milkmore Farms Weekly Reporting System
```

Type a line of asterisks (*) across the window's second line. (Begin at the second column within the window, because the first column of every window is protected.)

Center the banner in the width of the screen. Estimate where the upper left corner of the window would be if the window were centered. Position the cursor there, and then press *PF9*. The window moves to its new location. Repeat the process if you need to center it more precisely.

The window should look like this:



Press *PF3* and save the window.

Creating the Vertical Menu Window Named MAIN

You will now create the MAIN vertical menu window, which collects the amper variable &MAIN. Select

```
Create a new window
```

and press *Enter*.

BORDER and BANNER are text display windows, from which no options may be selected. Since MAIN, however, is a menu from which a selection must be made, choose

```
Menu (vertical)
```

and press *Enter*. Name the window:

```
MAIN
```

On the Description screen, type

```
User can report, graph, or exit.
```

and press *Enter*.

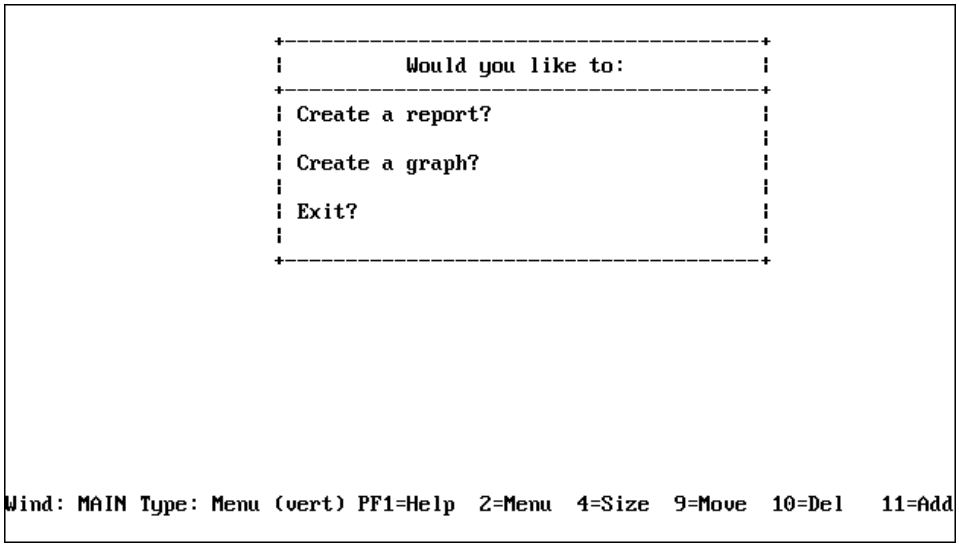
When prompted for a heading, type 10 spaces, then

```
Would you like to:
```

and press *Enter*.

On the Window Design Screen, move the cursor five rows from the top and 20 columns from the left, and press *Enter*. The window is created wide enough to contain the heading. Now position the cursor six rows below the window's bottom edge, and 10 columns to the right of its right edge. Press *PF4* and the window is resized.

Type the following menu options as they appear below:

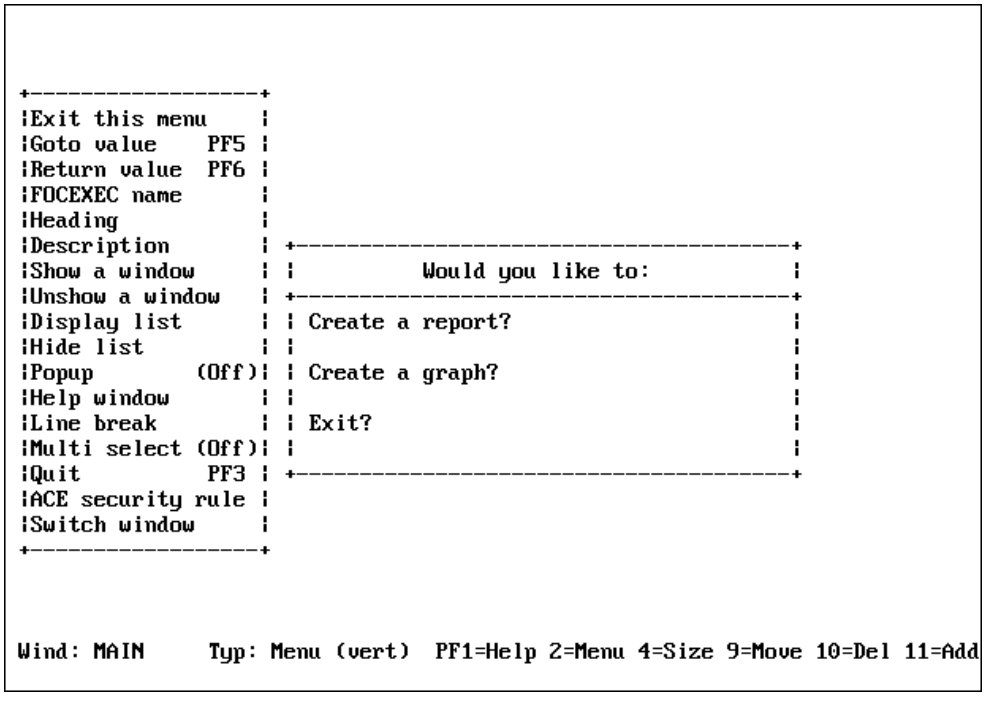


You assign goto and return values for each menu option. To assign either value to an option, the cursor must first be on that option.

Move your cursor back to

```
Create a report?
```

and press *PF2* to display the pop-up Window Options Menu.



Assigning a goto value tells the Window facility to display another window when this item is selected during execution.

In the next window of this application, the user is prompted to either execute an existing report or create a new one. The window that displays the prompt is called EXECTYPE, so the goto value of the first two menu options is EXECTYPE.

Move the cursor to

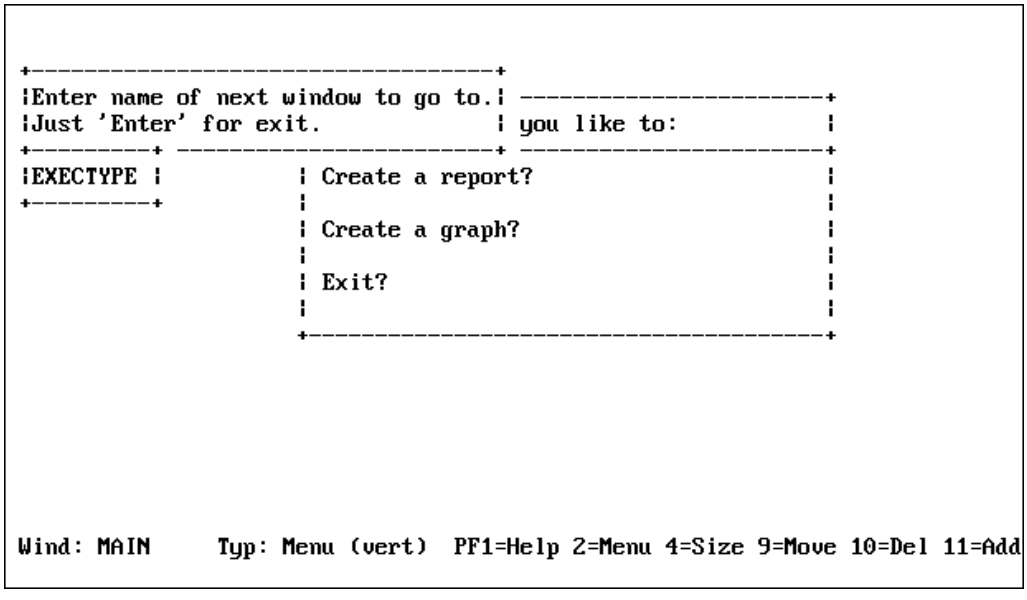
```
Goto value
```

and press *Enter*.

In the space provided, type

```
EXECTYPE
```

and press *Enter*.



The return value collected by this window—&MAIN—is tested in the FOCEXEC:

```
-START
-WINDOW SAMPLE MAIN
-IF &MAIN EQ XXIT     GOTOEXIT;
-IF &MAIN EQ RPT      GOTO GENERATE;
-IF &MAIN EQ GRPH     GOTO GENERATE;
.
.
.
```

Now move the cursor to

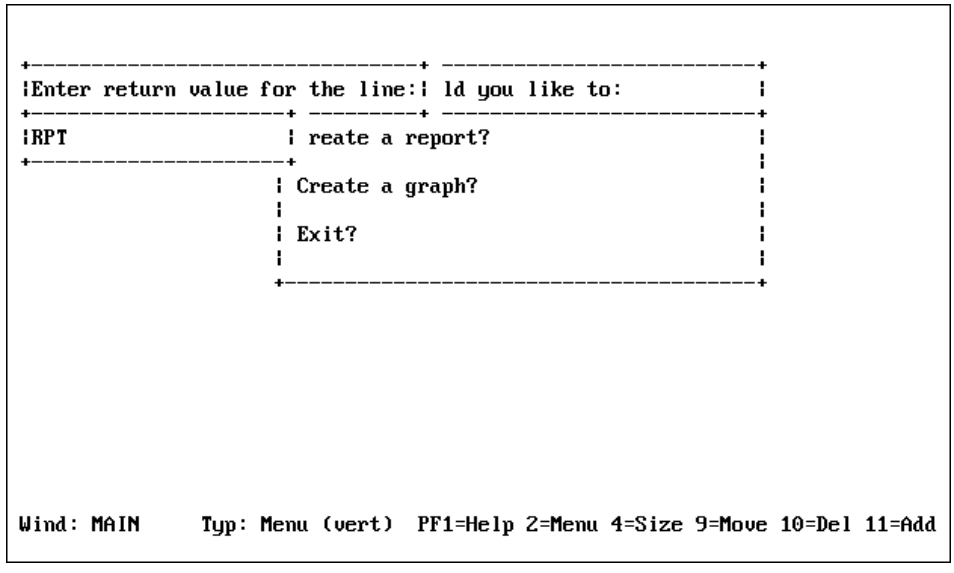
```
Return value
```

and press *Enter*.

Type the value

```
RPT
```

as shown, and press *Enter*.



Exit the Window Options Menu by moving the cursor to

```
Exit this menu
```

and pressing *Enter*.

Set the values for:

```
Create a graph?
```

Move the cursor to the second menu item, and press *PF2*.

Repeat the steps you just performed, assigning the goto value

```
EXECTYPE
```

and the return value:

```
GRPH
```

Leave the Window Options menu and move the cursor to

```
EXIT?
```

For this option, you do not assign a goto value. Since it exits to the FOCEXEC, there is no other window to be displayed.

Repeat the steps to assign the return value:

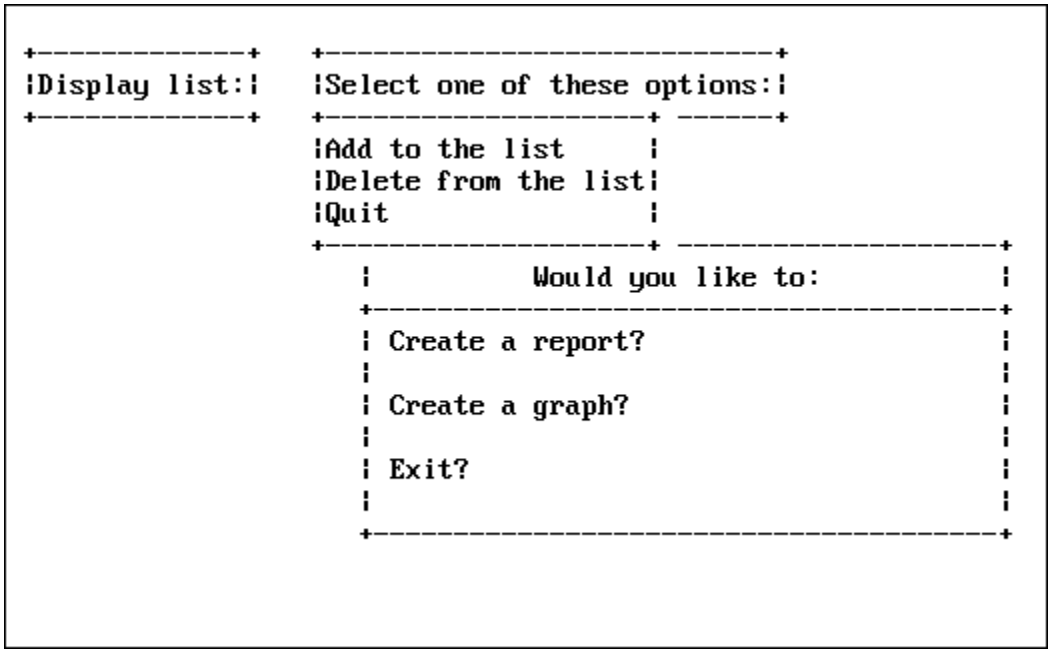
```
XXIT
```

With the Window Options Menu still on the screen, move the cursor to

```
Display list
```

and press *Enter*.

The display list may specify up to 16 windows to be displayed when this window is visible during execution. Since you want BORDER and BANNER to be displayed with MAIN, you must add each to the list.



Select:

```
Add to the list
```

A list of windows appears, from which you select by moving the cursor and pressing *Enter*. The windows must be selected in the order in which they should appear, because they are overlaid one on top of another when displayed. Select BORDER and BANNER for MAIN's display list, being certain to select BORDER first so that it is displayed behind BANNER.

When you have finished, choose *Quit* to return to the Window Options Menu.

Quit the Window Options Menu and press *PF3* to save MAIN.

Before moving on, look at what you have done so far. Select

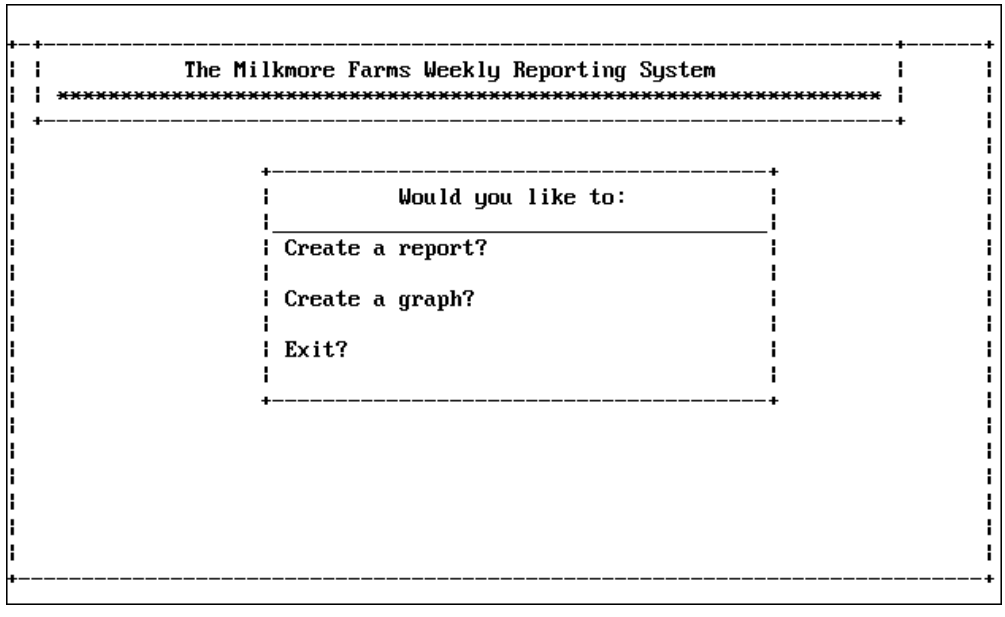
```
Run the window file
```

and press *Enter*.

Select

```
MAIN
```

as the starting screen. Press *Enter*, and the following appears:



Position the cursor on the "Create a report" line. When you press *Enter* to continue the display, you see an error message because EXECTYPE—the goto value—has not been created yet. Ignore it, and press *Enter* to continue. You see a screen displaying amper variables for this window and the values. Press *Enter* to return to the Main Menu.

Creating the Vertical Menu Window Named EXECTYPE

So far you have created two text display windows and a vertical menu. The next window we create is also a vertical menu.

Select

```
Create a new window
```

from the Main Menu, and choose

```
Menu (vertical)
```

from the Window Creation Menu. Enter

```
EXECTYPE
```

as the window name.

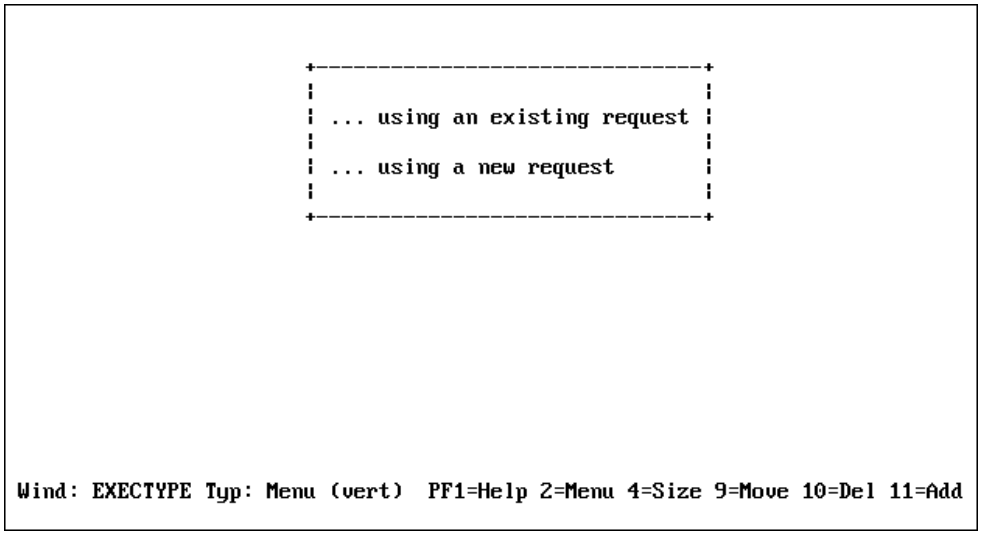
When prompted for a description, type

```
Create a new FOCEXEC or run existing one
```

and press *Enter*. When prompted for a heading, press *Enter*.

When the Window Design Screen appears, move the cursor 12 rows down the screen and 22 columns to the right, and press *Enter*. Now reposition the cursor four rows beneath the bottom edge of the window and 32 columns to the right of the right edge of the window, and press *PF4* to resize it.

Type the following two menu options as they appear below:



When you created the MAIN window, you used the Window Options Menu to set each return value and goto value. There is an easier way to set return and goto values using the PF6 and PF5 keys.

Pressing *PF5* prompts you successively for a Return value, a GOTO value, and a FOCEXEC name. When prompted for the Return value, enter EXIST and press *PF5*. You are prompted for A GOTO value. Press *Enter*, and you are prompted for a FOCEXEC name. Press *Enter*.

If you select

```
... using an existing request.
```

from the EXECTYPE menu, the file names window EXECNAME displays next. EXECNAME contains a list of existing FOCEXEC files from which you may choose.

Move the cursor to the second menu item.

Consider the return and goto values for this option.

If you choose to create a new report or graph request, EXECNAME is not displayed. Rather, control must pass back to the FOCEXEC, which executes these lines:

```
.
.
.
-IF &EXECTYPE EQ EXIST GOTO RPTEX ELSE GOTO NEWRPT;
.
.
.
-NEWRPT
-SET &PROCNAME=IF &MAIN EQ RPT THEN TABLETALK
ELSE IF &MAIN EQ GRPH THEN PLOTTALK;
&PROCNAME
-RUN
```

For control to pass to the FOCEXEC if this option is chosen, do not assign a goto value to it. Remember that during execution, control passes to the FOCEXEC when an option without a goto value is selected.

The return value may be anything other than EXIST. For now, press *PF6*, and enter

```
NEXIST
```

Rather than create display and hide lists for EXECTYPE, make a pop-up window. A pop-up window is displayed like any other window, but disappears when the user presses *Enter*. EXECTYPE pops up in front of MAIN.

Press *PF2* to display the Window Options Menu, move the cursor to

```
Popup(Off)
```

and press *Enter*. (Off) changes to (On).

Exit the Window Options Menu, press *PF3*, and save the window.

Creating the File Names Window Named EXECNAME

Your final window is the file names window that displays a list of existing FOCUS report requests. On the Window Creation Menu, select:

```
File names
```

Name the window

```
EXECNAME
```

and type in the description:

```
Select an existing FOCEXEC from list.
```

Enter an explanatory heading:

```
Select the request you want to execute and press ENTER:
```

You are prompted for file-identification criteria. Type

```
* FOCEXEC
```

and press *Enter*.

```
+-------------------------------------------------------------------------------+
|  INSTRUCTIONS :   Move cursor to selection and hit ENTER                      |
|                   Use PF3 or PF12 to undo a selection                         |
|                   Use PF1 for help                                            |
+-------------------------------------------------------------------------------+
        +-----------------------------------------+
        |Enter the file name criteria (e.g. * MASTER)|
        |or & variable name containing the criteria: |
        +-----------------------------------------+
        |* FOCEXEC                                |
        +-----------------------------------------+
```

When the application is executed, this selects all members of ddname FOCEXEC.

On the Window Design Screen, move the cursor two rows down and press *Enter*. Use PF9 to center the window on the screen. Resize the window: reposition the cursor two columns to the right of the window's right edge and 10 rows below the window's bottom edge, and press *PF4*.

Since only BORDER should be displayed with this window, add BANNER, MAIN, and EXECTYPE to the hide list and add BORDER to the display list.

When the user selects a file name from this window during execution, that file name is automatically collected as the return value. You cannot set the return value any other way for this type of window.

In the FOCEXEC, that return value is plugged into the line

```
EX &EXECNAME
```

and the report or graph request is executed.

In order for this to happen, you must return control to the FOCEXEC assigning no goto value to this window.

To change the file identification criteria of a file names window (or of a field names or file contents window) after it has been created, change the "return value." Although these two window types cannot have actual return values set when the window is created or edited, the "return value" that can be set is actually the window's file identification criteria. You can change the file identification criteria just as you would change the actual return value of a vertical menu window.

Exit from the Window Options Menu, press *PF3*, and save the window. The window file is complete. Exit from Window Painter.

Executing the Application

To execute the SAMPLE FOCEXEC, at the FOCUS prompt, type

```
EX SAMPLE
```

and press *Enter*. When prompted to choose a new or existing FOCEXEC, select

```
... using a new request.
```

unless you have created one in an earlier FOCUS session. The application executes PlotTalk or TableTalk. If you save the request you create, you can try the SAMPLE FOCEXEC again, and execute the new request by selecting:

```
... using an existing request.
```

This completes the tutorial.

Window Painter Screens

The creation of windows is itself an automated window-driven process. There are six major screens:

- The Entry Menu
- The Main Menu
- The Window Creation Menu
- The Window Design Screen
- The Window Options Menu

- The Utilities Menu

These screens assist you whenever you create or edit windows.

Invoking Window Painter

To invoke Window Painter, type the WINDOW PAINT command at the FOCUS prompt and press *Enter*.

Syntax: How to Invoke Window Painter

```
WINDOW [PAINT [filename]]
```

where:

`PAINT`

Is optional.

`filename`

Is the name of the window file that you want to work with. This is a member name. The member must belong to ddname FMU.

If you do not specify file name, you begin your Window Painter session at the Entry Menu where you can choose a window file to use or create a new window file. If you do specify file name, you skip the Entry Menu and begin your Window Painter session at the Main Menu working with the window file you specified.

If the file name does not exist, you are asked if you want to create a new file. If not, the Window Painter Entry Menu is displayed.

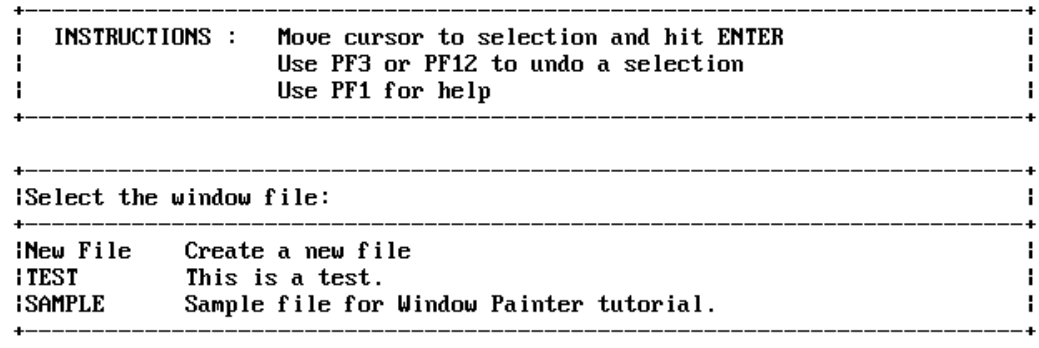
Entry Menu

You can reach the Window Painter Entry Menu by typing

```
WINDOW [PAINT]
```

at the FOCUS prompt, and pressing *Enter*.

The Entry Menu is the first screen you see:

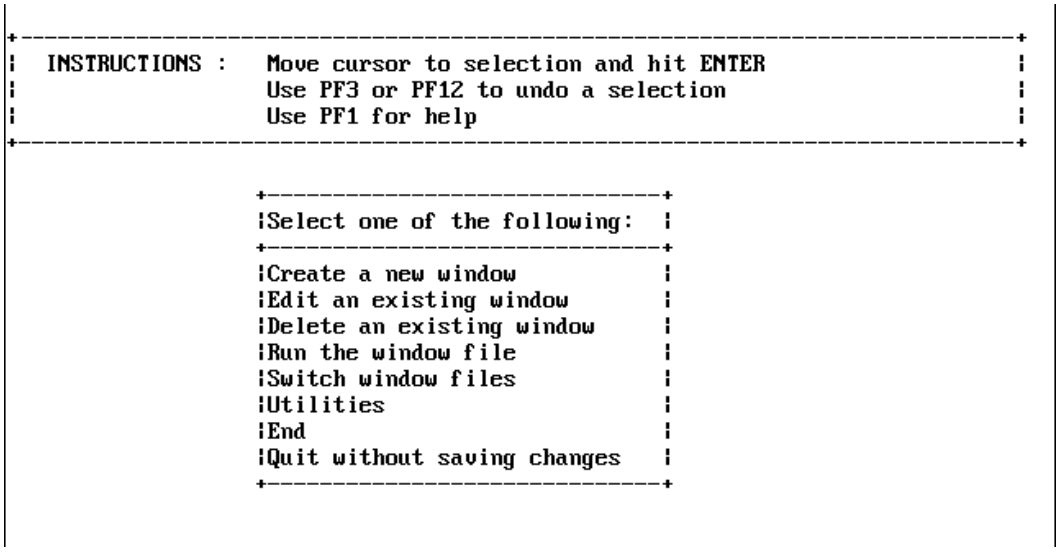


The Entry Menu invites you to choose a window file in which to work. If you are creating windows for a new application, you should start a new window file. If you are maintaining or creating windows for an existing application, use the window file that corresponds to your application.

When you become comfortable working with windows, you can write FOCEXECs that include branching between window files. Refer to *Transferring Control in Window Applications* on page 470 for a detailed discussion on branching and transferring control.

Main Menu

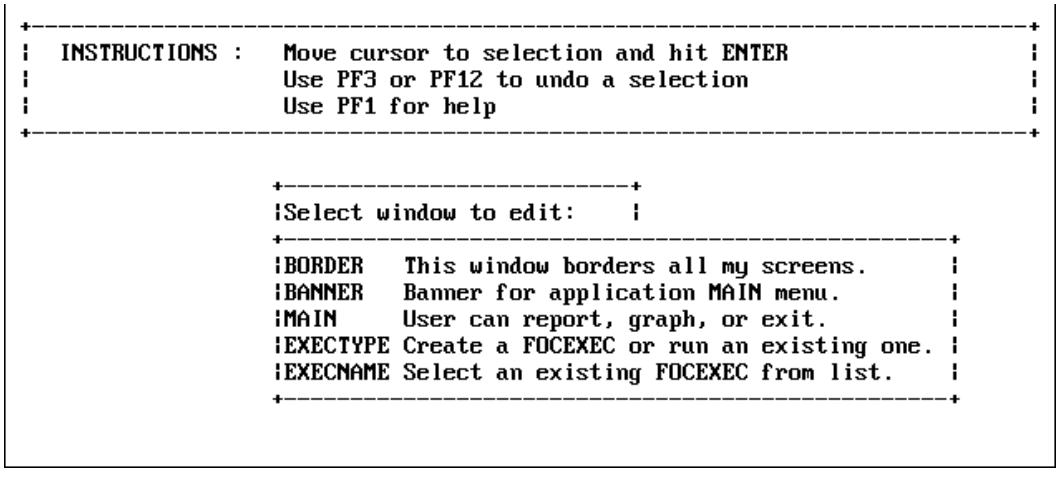
Once you have selected a window file from the Entry Menu, or entered the WINDOW PAINT command with the file name option, the Main Menu appears:



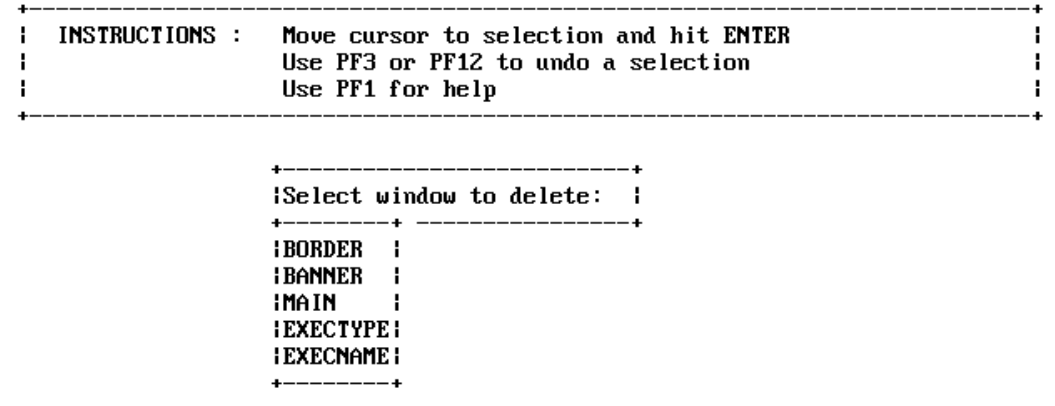
The following table summarizes the options on the Main Menu, along with illustrations of screens that appear when you select the options:

Menu Option	Description
Create a new window	Brings up the Window Creation Menu. You can select the type of window to create.

Menu Option	Description
Edit an existing window	Brings up a list of windows in your current window file. You can select the one to edit.



Menu Option	Description
Delete an existing window	Brings up a list of windows in your current window file. You can select the one to delete.



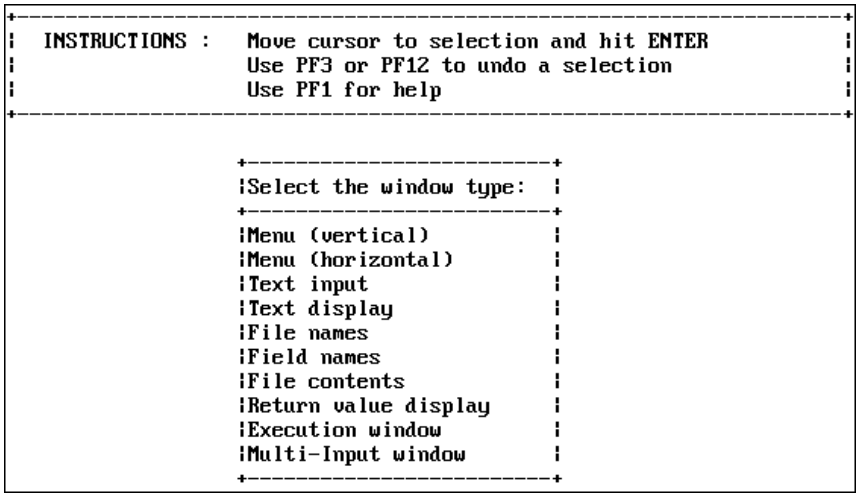
Menu Option	Description
Run the window file	Brings up a list of windows in your current window file. You can select the one from which to start running the window file. After the window file is run, the windows' amper variable values are displayed. The display includes the first 20 characters of each value. This option shows you how your windows work without executing the FOCEXEC. Use this option to test your window file.
Switch Window files	Returns you to the Window Painter Entry Menu, from which you can select another window file. The previous window file is saved whenever you switch window files.
Utilities	Brings up the Utilities Menu, which is discussed in *Utilities Menu* on page 521.
End	Returns you to native FOCUS. All work saved during the Window Painter session is kept.
Quit without saving	Returns you to native FOCUS. All work saved during the Window Painter session is discarded.

Window Creation Menu

You can reach the Window Creation Menu by selecting

```
Create a New Window
```

from the Main Menu. The following screen appears:



You need to select the type of window to create. You are asked to enter an 8-character name and an optional 40-character description. These are for your use only and do not appear in the window during execution.

For a vertical menu, horizontal menu, text input, text display, file names, field names, file contents, multi-input, or return value display window, you are prompted to supply a 60-character heading.

For a text input window, you are prompted to choose the format of the text entry field (alphanumeric, with all text translated to uppercase; alphanumeric, with no case translation; or numeric). Later, in the Window Design Screen, you can make the length of the text entry field shorter than the window's header length by typing a single character in the window immediately following the last desired field position, or by typing characters continuously from the first field position to the last desired field position.

For a file names, field names, or file contents window, you are prompted to produce file-identification criteria that can consist of an amper variable, a complete file identification, or (for file names windows) a file specification which includes an asterisk (for example, * MASTER).

The asterisk is used as a wildcard character indicating that any character or sequence of characters can occupy that position. The asterisk can be used as the member name but not in the ddname.

If an amper variable is used, you can prompt for the file identification criteria at run time.

File-identification criteria must specify the member name first and the ddname second.

If you are creating a field names window, your file-identification criterion is the name of a Master File.

In addition, you can create execution windows containing FOCUS commands such as Dialogue Manager commands or TABLE requests. You are prompted for the window name and heading. Once a window has been specified, the Window Design screen opens.

For complete information about the types of windows you can create in Window Painter, see *Types of Windows You Can Create* on page 452.

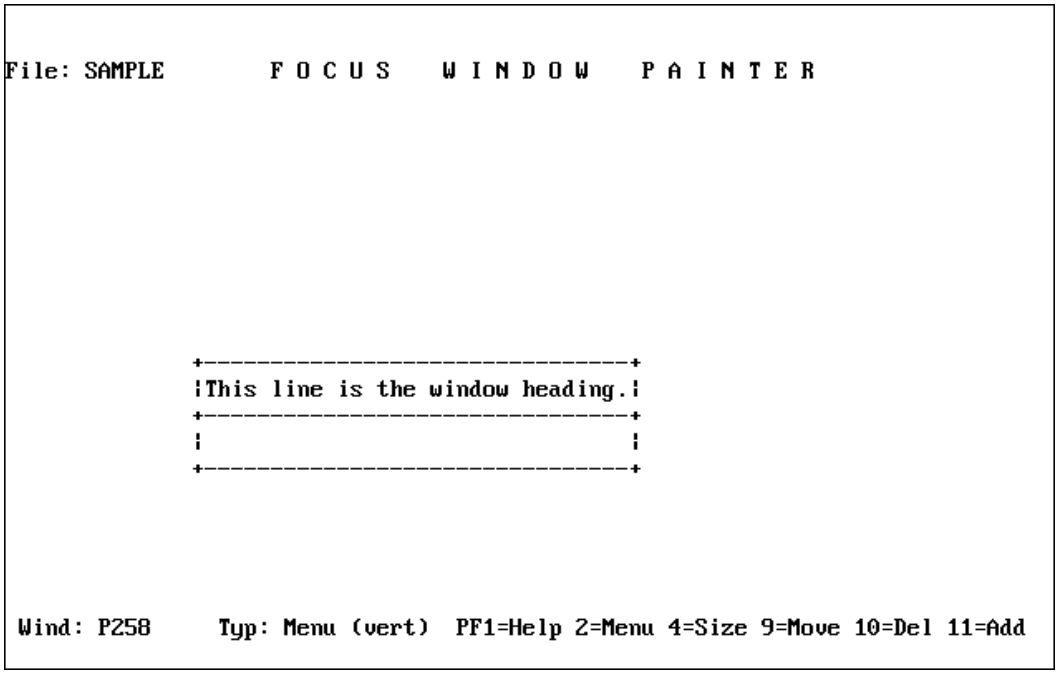
The next screen displayed is the Window Design Screen, discussed in the next section. This screen enables you to enter information, and position and size your window.

Window Design Screen

In this screen you design the appearance and functionality of your windows. It appears during the window creation process, when you press *Enter* after typing the heading of your window.

The Window Design Screen consists of a blank screen, a cursor, and text asking you to move the cursor to the starting position for the window. This starting position becomes the upper left corner of the window. Use the cursor arrow keys to move the cursor to the place where you want the upper left corner of the window to be, and press *Enter*.

The window appears with its heading at the top. You can enlarge it, type text in it, and move it around the screen.



The Window Design Screen allows you to use the keyboard to manipulate the window you are creating.

The following chart summarizes Window Design Screen key functions in all window types.

PF Key	Function
PF1	Displays a window of help information.
PF2	Displays the Window Options menu. This menu is discussed in *Window Options Menu* on page 509.
PF3	Displays the exit menu. You can select: ❑ Exiting from the Window Design Screen while saving your work. ❑ Quitting from the Screen without saving your work. ❑ Continuing your work.

PF Key	Function
PF4	Resizes the window. First move the cursor to the desired position of the window's lower right corner. When you press *PF4*, the window's upper left corner remains in the same position; the window's lower right corner moves to the current cursor position. If the window size is reduced, nothing in the window is deleted; all window contents beyond the window border can be seen by scrolling the window.
PF5	Gets the Return value, the GOTO value, and the FOCEXEC name for the active window.
PF6	Sets the return value of the line that the cursor is on.
PF7	Scrolls the window up if the window contents extend beyond the top border.
PF8	Scrolls the window down if the window contents extend beyond the bottom border.
PF9	Moves the window. First move the cursor to the desired position of the window's upper left corner. When you press *PF9*, the window's upper left corner (the + in the border) moves to the current cursor position. The rest of the window moves accordingly.
PF10	Deletes the line of window contents identified by the current cursor position. If the window contents do not extend beyond the window borders, the window itself is reduced by one line.
PF11	Adds one line of window contents beneath the line identified by the current cursor position. If the window contents do not extend beyond the window borders, the window itself increases by one line.
PF12	Provides the same function as the PF3 key.
PF13 - PF24	These keys provide the same functions as the corresponding keys PF1 - PF12.

If a window's contents extend beyond a top or bottom border, then the message

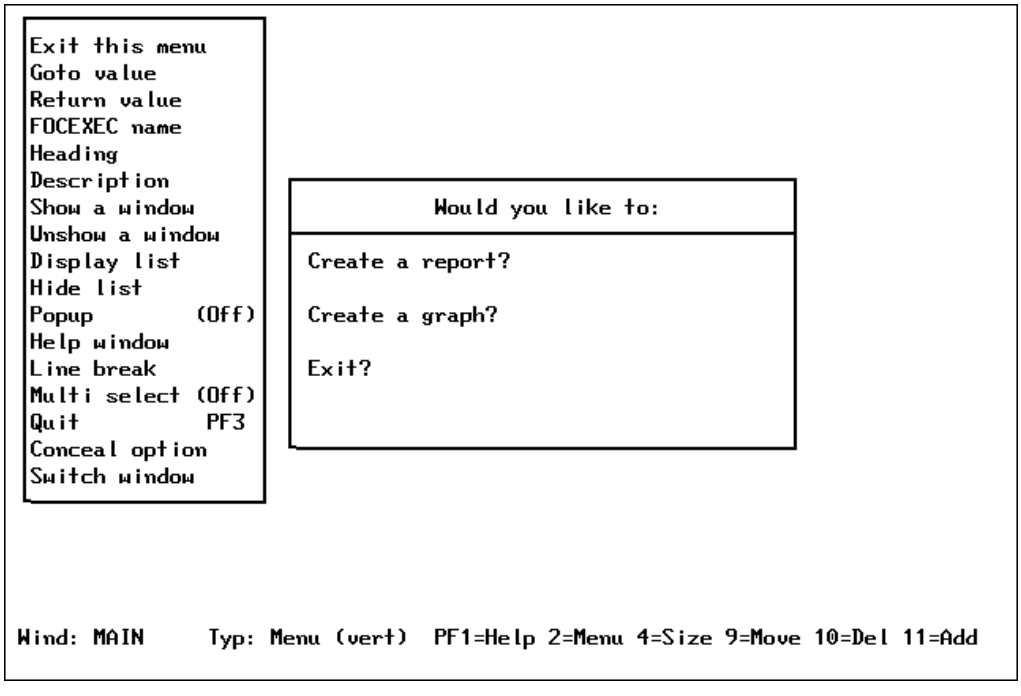
```
(MORE)
```

is displayed on that border to remind you of more lines of contents hidden beyond that border. You can view these lines by scrolling toward the border. When the window is used in an application, the user can also scroll the window to see all of the contents.

The display line at the bottom of the Window Design Screen shows instructions or information. When you first see the Window Design Screen, the display line tells you to move the cursor and press *Enter*. The display line shows the name of the window file, and the name and type of window being created; it also tells which keys to press for the HELP function, the SIZE function, and the Window Options Menu.

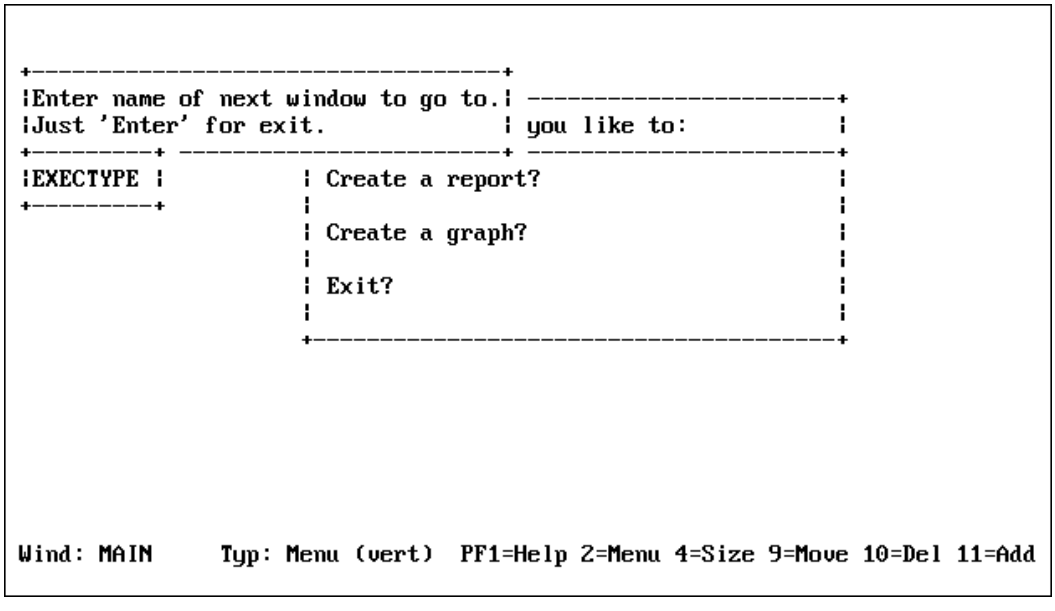
Window Options Menu

When the Window Design Screen is displayed, pressing *PF2* brings up the following Window Options Menu:

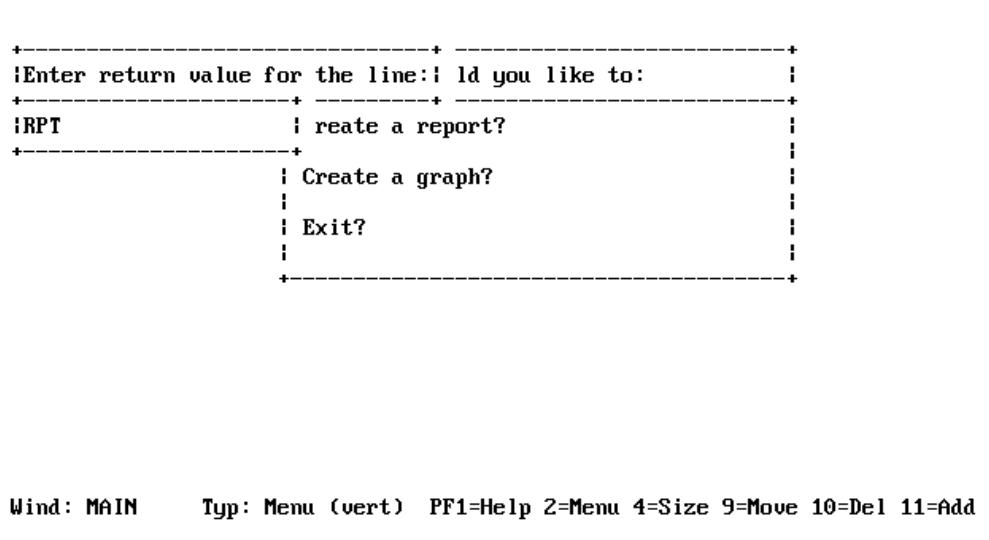


The following table summarizes the options on this menu, along with illustrations of screens that appear when you select some of the options:

Menu Option	Description
Goto value	Selecting this option allows you to specify the next window in the path from this selection field or window. You are asked to supply the name of the window. (It does not matter whether or not this window exists. You can create it later, but remember the name chosen.) In menu windows, goto values are assigned to each menu item. In other windows, there is a single goto value for the entire window. To assign a goto value, your cursor must be on the proper line when the Window Options Menu is brought up. Select Goto value from the Window Options Menu. You are prompted to enter the name of the window that is the target of the goto. Type the name in the space provided and press *Enter* again. The goto value is assigned.

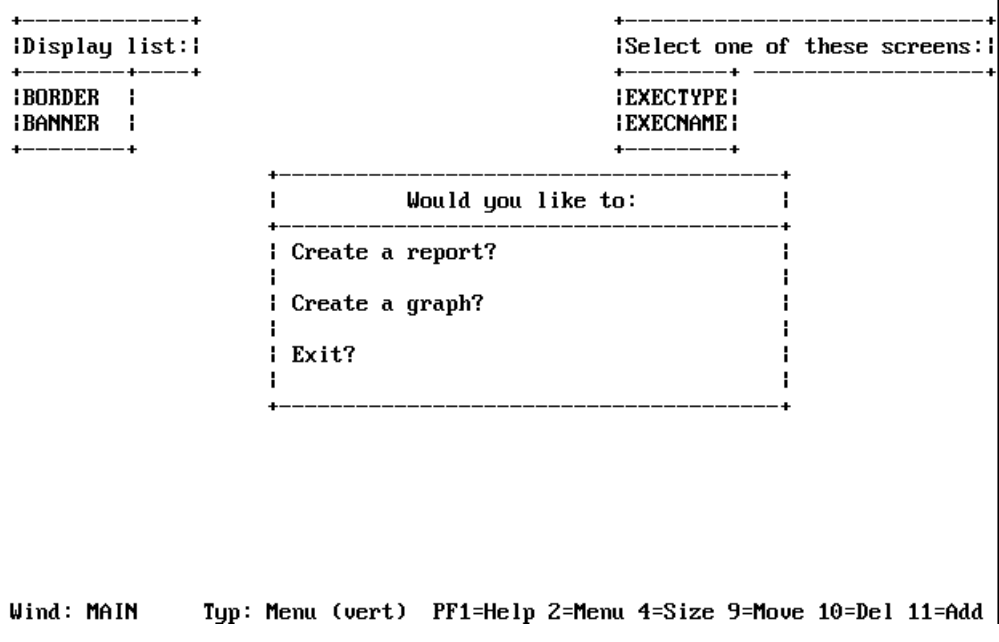


Menu Option	Description
Return value	The return value supplies a value for an amper variable. If the user selects this field during execution, the return value you have assigned is plugged into the amper variable in your FOCEXEC. Return values are assigned to each menu item in menu windows, and one per window for other window types. The only exceptions are the multi-input window, where the return value is the name of the input field occupied by the cursor when you pressed *Enter* or a PF key, and the return value display window, which does not have a return value but instead displays other windows' return values. The return value for a Multi-Select window is the number of selections. To assign a return value, your cursor must be on the proper line. Select Return value from the Window Options Menu and you are prompted to enter a return value. Note that for file names, field names, and file contents windows, the value that you enter is the file-identification criterion for that window. Type the value in the space provided and press *Enter* again to assign the return value .

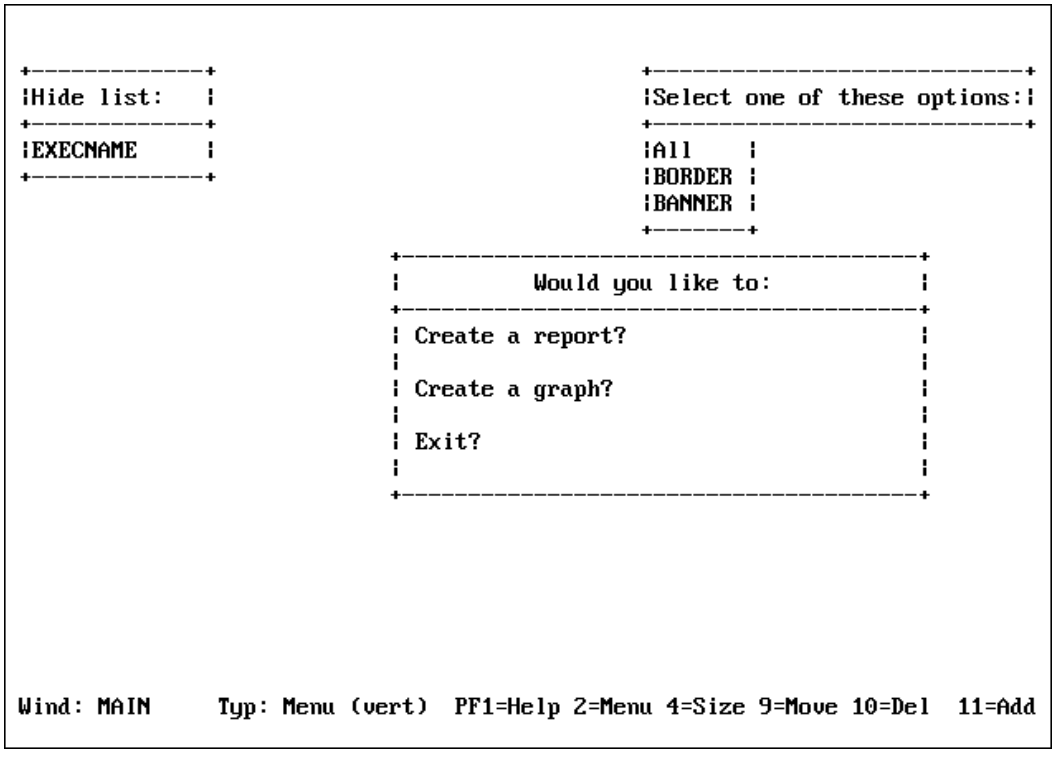


Menu Option	Description
FOCEXEC name	Attaches a FOCEXEC to each menu selection of the window. The FOCEXEC is executed when the menu item is selected.
Heading	Changes the heading of any window you are working on. You can also add or remove a heading.
Description	Changes the description of any window you are working on.
Show a window	Used only during window editing, brings another window onto the screen for reference. You cannot edit the second window.
Unshow a window	Removes the shown window from the display.

Menu Option	Description
Display list	Enables you to specify a list of up to 16 windows that are visible when this window is displayed during execution. Note that if part of a window on the display list extends beyond the window border or does not fit on the screen, it cannot be scrolled. As many as 16 windows can be displayed on the screen at one time. This applies to all windows on the screen (that is, a window displayed during execution, windows displayed when executed previously and not hidden afterward, and windows displayed because specified on a display list). The window facility interprets each window heading as a separate window: if all of the windows have headings, 16 can be displayed on the screen at one time.



Menu Option	Description
Hide list	Allows you to specify windows that does not appear when this window is displayed during execution. You can specify up to 16 specific windows or all windows in the window file. If you select "All", all the windows are hidden except those in the display list. If you do not hide a window that was displayed, it remains on the screen until another window that includes it on a hide list is displayed during execution.



Menu Option	Description
Popup (Off/On)	Makes the window disappear when the user presses *Enter* during execution. Defaults to OFF, which leaves the window on screen. Set Popup to OFF with text display windows as they do not work even if set to ON.

Menu Option	Description

Help window	Allows you display information about a window or a menu item when a user presses *PF1* (the Window facility HELP key) during execution. The information displayed is text within a specified Help window.
	
	Note that if the PFKEY option is specified in the -WINDOW command, you have to explicitly set a PF key as the HELP key, as described in *Testing Function Key Values* on page 475.
	
	When selecting the Help window option, you are asked to supply the name of the Help window file that contains the Help window. Next, you are asked to supply the name of the Help window itself. The Help window can be an existing window, or one that you created.
	
	If the Help window displays field names, it qualifies duplicates with the segment name.
	
	You can use any window type for a Help window. A text display window is easiest, except when supplying different help information for each item in a vertical menu, horizontal menu (that is, item-specific help).
	
	To assign item-specific help, use a file contents window that displays a file containing text in the following format
	
	`=>HELPFILE`
	`=> menu item`
	`this is the Help message you want the user to see.`
	
	where:
	
	`=>`
	
	Is entered with an equal sign (=) and a greater-than sign (>).
	
	`HELPFILE`
	
	Must be uppercase.
	
	`menu item`
	
	Is the exact text of the menu item. Any blank spaces that precede this text in the menu must also precede this text here in the Help file. Note that at least one blank space always precedes the menu item text in a vertical menu, horizontal menu, or multi-input window.

Menu Option	Description

Help window (*continued*)	For example, if the first three lines of a vertical menu are
	`(1) Generate a sales report`
	`(2) Generate a stock report`
	and there are three blank spaces between the left border of the window and the beginning of the text, the file containing help text could look like this:
	`=>HELPFILE`
	`=>   (1) Generate a sales report`
	`This option displays a list of existing sales report requests, and lets you select one of these requests to execute.`
	`=>   (2) Generate a stock report`
	`This option displays a list of existing stock report requests, and lets you select one of these requests to execute.`
	The lines immediately following the menu item text are displayed when the user positions the cursor on the menu item and presses *PF1*.
	In some cases you may assign topic-specific help, but want the help text for some of the topics to be contained in a separate file. In this case, on the line following the menu item text, replace the help message with the file identification of the file containing that menu item's help message.
	Use this file-identification format:
	`FILENAME= membername ddname`

Menu Option	Description

Help window *(continued)*	To assign one set of instructions that can be used for multiple menu items, use the following syntax:
	`=>DEFAULT`
	`This text appears when you have not written topic-specific help.`
	The DEFAULT text must be the last section in the Help file.
	Lines beginning with an asterisk (*) are comment lines that are not displayed.
	What follows is an example of a topic-specific Help file for the Main Menu used in the tutorial.
	`=>HELPFILE`
	`*Help file for tutorial/Main Menu`
	`=> Create a report?`
	`Choose this option if you wish to create a new report.`
	`=> Create a graph?Select this option if you wish to create pie charts, bar charts or other graphics.`
	`=> Exit?`
	`If you wish to leave the application, choose this option.`

Menu Option	Description

Line-break	Formats the contents of the return value display window. This option is set when designing the windows from which you collect the return value(s) to be displayed.
	
	When you select this option, you see:
	
	`None`
	`New line before value`
	`New line after value`
	`Both`
	
	where:
	
	`None`
	
	Places return value directly after preceding value. If there is not enough room on this line, return value is placed on the next line.
	
	`New line before value`
	
	Places return value on the next line.
	
	`New line after value`
	
	Places return value on the same line as preceding value. Places next return value on next line.
	
	`Both`
	
	Places return value on a line by itself.

Menu Option	Description

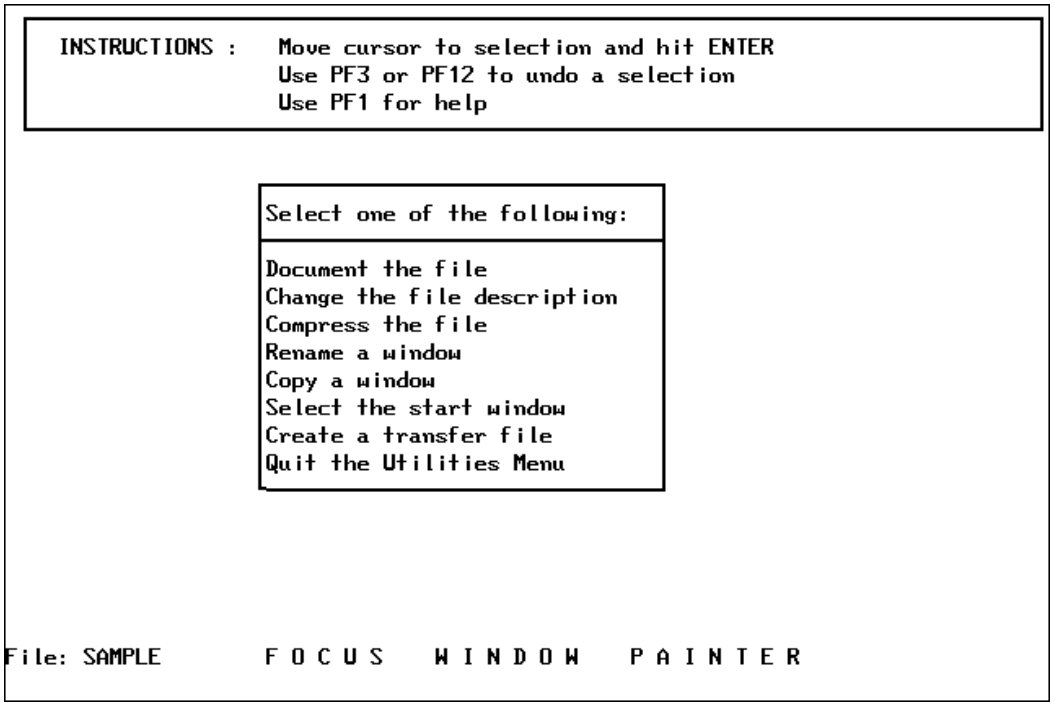
Multi-Select	Enables you to select multiple items from one window. The number of items you select is collected as the return value from that window; each selected item's return value is stored in a temporary file in memory. You can later retrieve these stored values for use in a FOCEXEC. Values for up to 8 windows can be stored at one time.
	
	When you select this option, you see:
	
	`-Select Multi(On )`
	
	During execution, the user selects individual values by pressing *PF9*. After all selections have been made, the user presses *Enter*.
	Note that when the -WINDOW command is issued with the PFKEY option, the PF9 key cannot be used to make selections unless a SET command is issued before the -WINDOW command. For example:
	
	`SET PF09=SELECT`
	
	You can also set a different PF key for selecting multiple items.
	A Multi-Select window can have no more than one goto value. Although in a vertical menu window you can assign a different goto value to each menu item, only the value assigned to the first item is effective.
	The return value collected for a window using the Multi-Select option is the number of values selected by the user.
	To retrieve the individual values, issue a special WINDOW call, as follows
	
	`-WINDOW` *`windowfile windowname`* `GETHOLD`
	
	where:
	
	`windowfile`
	
	Is the name of the window file.
	
	`windowname`
	
	Is the name of the Multi-Select window.
	
	`GETHOLD`
	
	Is the special parameter that retrieves one value at a time from the temporary file.

Menu Option	Description
Multi-Select (*continued*)	The value is assigned to the variable &windowname. The GETHOLD option requires at least two -WINDOW commands in your FOCEXEC. The first -WINDOW command (without the GETHOLD option) transfers control to the Window facility where a Multi-Select window is used. The second and subsequent -WINDOW commands use the GETHOLD option to retrieve the stored amper variables collected in a particular Multi-Select window. For each value to be retrieved, you need a -WINDOW command with the GETHOLD option. Each value is stored in &windowname. To use this value, assign it to another variable. For example, if the return value has the value 4, issue the special -WINDOW command four times; each time you would collect the value from &windowname. Alternatively, you could perform a loop. Note that -WINDOW with the GETHOLD option does not transfer control from the FOCEXEC to the Window facility.
Quit	Returns you to the Window Painter Entry Menu.
Input fields	Input fields pertain to Multi Input Windows. Selecting the field takes you to that field.
Menu text	Specifies a line of descriptive text, up to 60 characters long, for items on a horizontal menu. Use the Text line option to position the text.
Text line (x+1)	On a horizontal menu, positions descriptive text one or two lines above or below the menu. Valid values are x+1 or x+2 to place the text above the horizontal menu, x-1 or x-2 to place the text below the horizontal menu. Use the Menu text option to define the descriptive text.

Menu Option	Description
Pulldown (off/on)	If the setting is ON, placing the cursor on an item in a horizontal menu can display an associated pull-down menu. The default setting is OFF. Turn the setting ON by positioning the cursor on this option and pressing *Enter*. The pull-down menu must be a vertical menu and must be assigned as the goto value for the horizontal menu item. Note that setting Pull-down ON automatically shuts off Menu Text.
Switch window	Enables you to work on and move between two windows. When you select this option, you can create a new window or edit an existing window without returning to the Main Menu.

Utilities Menu

If you select the Utilities option from the Window Painter Main Menu, the Utilities Menu is displayed:



The following table summarizes the options on this menu, along with illustrations of screens that appear when you select some of the options:

Menu Option	Description
Document the file	When you select this utility, Window Painter creates documentation of the window file. You can display the document on the screen using TED or another system editor, or send it to a printer or disk file. This option creates a member of the TRF PDS; that PDS must have already been allocated. However, creating a PDS is not necessary if you are only going to use the documentation file during the current FOCUS session: Window Painter temporarily allocates the PDS. This document contains detailed information about all the windows in the window file. It shows you the kinds of windows, the structure and format, and any options you have assigned from the Window Options Menu, including return and goto values. The text you enter when prompted for a window file description or individual window description is part of this document. The document is especially useful when creating a FOCEXEC, since it provides return and goto values in addition to other information. **Note:** If you create another file with the same name, the file is not overwritten. It is appended.

```
-* WINDOW FILE NAME=SAMPLE
-* DESCRIPTION='Sample file for windows tutorial'
-* WINDOW NAME=MAIN,    TYPE=Menu (vertical)
-* DESCRIPTION='User can report, graph, or exit.'
-* ROW= 6,COLUMN=23,HEIGHT= 7,WIDTH=38,WINDOW= 7,POPUP= 0,BORDER= 2,HEADLEN=28,
-* RETURN=None
-* MULTI=Off
-* HEADING:
-*      Would you like to:
-* WINDOW DATA:                GOTOS:          VALUES:
-*  1.'                        ','             ','             '
-*  2.' Create a report?       ','EXECTYPE     ','RPT          '
-*  3.'                        ','             ','             '
-*  4.' Create a graph?        ','EXECTYPE     ','GRPH         '
-*  5.'                        ','             ','             '
-*  6.' Exit?                  ','             ','XXIT         '
-* DISPLAY LIST:
-* BORDER
```

Menu Option	Description
Change the file description	Changes the description of the current window.
Compress the file	This utility is provided to help you save space in memory. It allows space made available by deleted or edited windows to be reused.
Rename a window	When you select this utility, you see a list of the windows in the current window file. You can change the name of any of these windows.
Copy a window	This function copies a window from one window file to another, or duplicates it within the same file. The copy function is useful when you create a new application, or need to add windows to an existing application, and want the windows to look like those you have already created. You can copy any window and edit it to conform to the new application.
Select the start window	Enables you to choose a default start window. This window is the first to be entered if a specific window is not selected upon startup. If a default start window is not explicitly chosen, FOCUS selects the first window created to be the start window.
Create a transfer file	Creates a file to be transferred for use with the Window facility in another FOCUS environment. This option creates a member of the TRF PDS; that PDS must have already been allocated.
Quit the utilities menu	Returns you to the Main Menu.

Transferring Window Files

If you use FOCUS in more than one operating environment, you can transfer an existing window file from one environment to be used in another environment. For example, if you have a fully-developed window application in PC/FOCUS, and you want to develop a similar application in mainframe FOCUS, you can transfer the PC/FOCUS window file to mainframe FOCUS.

You can transfer a window file to a new environment in four simple steps:

1. Create a transfer file from the original window file using Window Painter.
2. Transfer the new file to the new environment using FTP.
3. Edit the transferred file in TED, if necessary.
4. Compile the transferred file using the WINDOW COMPILE command.

These steps are described in the following topics.

Creating a Transfer File

The window files that you design in Window Painter are compiled files; before a window file can be transferred to another environment, a user-readable source code version must be created. This user-readable file is called a transfer file, and is created using the transfer file option of Window Painter.

- This Window Painter option automatically creates a new member of the PDS allocated to ddname TRF; the PDS must already have been allocated (with LRECL between 80 and 132 and RECFM FB). However, it is not necessary to create the PDS if you use the transfer file during the current FOCUS session: Window Painter temporarily allocates the PDS.
- For information about the transfer files created by FOCUS Window Painter in other operating environments, see the appropriate FOCUS Users Manual for those environments.

To convert a window file to a transfer file, go to the Window Painter Utilities Menu and select:

```
Create a transfer file
```

You are prompted for the name of the new transfer file. Enter a member name; it can have the same name as the window file, or an entirely new name.

Note that you should not give the transfer file a name already assigned to a window documentation file. Also, you should not give the transfer file a name already assigned to an existing transfer file unless you want to merge the two files. See the appropriate operating environment topic in the *Overview and Operating Environments* manual for more information about duplicate window transfer and window documentation file names.

You are asked to select which window(s) you want to transfer. Select

```
All
```

to transfer all of the windows in the current window file, or select any single window in the file. This is the last step in creating a transfer file.

Note that you can merge transfer files: if a transfer file already exists for your window file, and you only need to add a new window to it, you can give the new transfer file the same name as the old one, and select the new window. Window Painter merges the source code for the new window into the existing file, so that you have a single complete transfer file.

Transferring the File to the New Environment

Once the transfer file exists, it can be transferred to the new environment using FTP.

Editing the Transfer File

Window facility features introduced in one FOCUS release may not be fully supported in earlier releases. Because different operating environments may be running different releases of FOCUS, the transfer file created by the FOCUS Window facility in one environment may contain features not fully supported by the Window facility in another environment.

If your transfer file contains Window facility features not fully supported in the new environment, you may need to remove or fine-tune those features. If the new environment supports features are not supported in the original environment, you can add those features to the transfer file. Adding, removing, and fine-tuning features can be done by simply editing the transfer file.

The Format of the Transfer File

The transfer file is a user-readable source code listing of all of the windows and features that were included from the original window file. You can remove or fine-tune an unsupported feature by simply editing or deleting the appropriate line in the transfer file. You can accomplish this by using TED or any other editor.

Each transfer file contains:

- One set of window file attributes describing the file.
- For each window defined in the file, one set of window attributes describing that window.
- For each line in each window, one set of attributes describing that line.

If any attribute is not specified in the transfer file, it defaults to a value of zero or blank (depending on whether the value is normally numeric or alphanumeric).

Reference: **Transfer File Syntax: Window File Attributes**

Attribute	Description
FILENAME	The name of the original window file.
DESCRIPTION	A comment field describing the file.
WINDOWNAME	The name of the window.
TYPE	The type of window: 1. Vertical menu 2. Text input window 3. Text display window 4. Horizontal menu 5. File names window 6. Field names window 7. File contents window 8. Return value display window 9. Execution window 10.Multi-input window
COMMENT	A comment field describing the window.
TRANSLATE	Type of input for text input windows (Type 2). 0 Allow mixed-case input. 1 Allow numeric input only. 2 Translate input to uppercase.
ROW	The row number of the upper left corner of the window.
COLUMN	The column number of the upper left corner of the window.

Attribute	Description
HEIGHT	The height of the window data (the number of lines of window data, not the height of the actual window frame). If there are more data lines than what fits in the window frame, use the PF7 and PF8 keys to scroll the window.
TEXT LINE	Position of menu text. Values are: +1, +2, -1, -2.
WIDTH	The width of the window frame, not including the border.
INPUT FIELDS	Fields for multi-input windows.
WINDOW	The number of lines in the actual window frame (not the number of lines of window data). This does not include borders.
POPUP	Sets the pop-up feature. 0 This is not be a pop-up window. 1 This is a pop-up window.

Reference: Transfer File Syntax: Window Attributes

Attribute	Description
BORDER	Sets the window border. 0 There is no window border. 1 There is a window border. 2 There is a window border. Options 1 and 2 both result in a basic window border.
HEADLEN	Length of the window heading. If this value is 0, there is no heading.

Attribute	Description
RETURN	Sets the line break feature for use with return value display windows. 0 Line break is not used. 1 New line before this return value. 2 New line after this return value. 3 New line before and after this value.
MULTI	Sets the multi-select feature. 0 This is not a multi-select window. 1 This is a multi-select window.
HEADING	The text of the window heading.
HELP	The name of the help window for this window.
HELPFILE	The name of the window file that contains the help window.
DISPLAY	The name of a window to be displayed at the same time this one is displayed. There can be up to 16 DISPLAY values for each window. This attribute is optional.
HIDE	The name of a window to be hidden when this one is displayed. There can be up to 16 HIDE values for each window. This attribute is optional.

Reference: **Transfer File Syntax: Window Line Attributes**

Attribute	Description
DATA	A line to be displayed in the window (for example, a menu choice in a vertical menu Window, or a line of text in a text display window). The data can include amper variables (including &windowname).

Attribute	Description
GOTO	The name of the window to go to if this line is selected by the user. The value can be an amper variable (including &windowname). If the value is blank, and this line is selected, Windows returns to Dialogue Manager.
VALUE	The return value supplied if this line is selected by the user. This value is placed in the amper variable &windowname, where windowname is the name of the window. For file names windows (TYPE = 5), this is the file selection criteria (including asterisks) of the file names to be displayed. For field names windows (TYPE = 6), this is the name of the Master File whose fields are displayed. For file contents windows (TYPE = 7), this is the name of the file whose contents are to be displayed.

Operating Environment Considerations

When you transfer a window file to a mainframe operating environment from a different environment, differences in hardware and operating software may require that you make changes to the file. These changes are discussed below.

- **Screen position.** Windows should not begin in row 1 or in column 1. If you transfer a window with these row or column positions, truncation occurs. Adjust the ROW and COLUMN attributes if necessary.
- **Screen size.** Windows should not have more than 22 rows or 77 columns. Windows that extend beyond the end of the terminal screen is automatically truncated without any warning message.

  This is important to note if you are transferring a window file from an environment where the screen size differs from that in the mainframe environment. Adjust the ROW and COLUMN attributes if necessary.
- **Window Position.** Column 1 of vertical menu, horizontal menu, multi-input and text display windows cannot be used. Window text must begin to the right of column 1.
- **Function keys.** Windows transferred from other environments may refer to function keys not present in the mainframe environment. Change function key references if necessary.

- **Blank lines.** Blank line are acknowledged by Window Painter.
- **Colors and Border Types.** The use of colored windows and background and multiple border types is not supported.
- **File Naming Conventions.** File naming conventions differ in different operating environments. When transferring a file from some environments, the Window facility automatically translates references to FOCEXECs, Master Files, and error files, as shown below. You must change other file references yourself when you edit the transfer file.

PC or UNIX Extension	Mainframe ddname
.FEX	FOCEXEC
.MAS	MASTER
.ERR	ERRORS

Example: Sample Transfer File

To illustrate the transfer file format, part of the transfer file for the SAMPLE window file is shown below (SAMPLE is described in the tutorial). The MAIN and EXECNAME windows from the file are included in the example.

```
FILENAME=SAMPLE
DESCRIPTION='Sample file for windows tutorial'
WINDOWNAME=MAIN,TYPE=1
COMMENT='User can report, graph, or exit.'
ROW= 6,COLUMN=23,HEIGHT= 7,WIDTH=38,WINDOW= 7,POPUP= 0,BORDER=
2,HEADLEN=28
RETURN=0
MULTI=0
HEADING='Would you like to:'
DATA='   '
$
DATA='         Create a report?'
GOTO='EXECTYPE',VALUE='RPT '
$
DATA='   '
$
DATA='         Create a graph?'
GOTO='EXECTYPE',VALUE='GRPH'
$
DATA='   '
$
DATA='         Exit?'
GOTO='        ',VALUE='XXIT'
$
DATA='   '
$
DISPLAY=BORDER   ,$
DISPLAY=BANNER   ,$
WINDOWNAME=EXECNAME,TYPE=5
COMMENT='Select an existing FOCEXEC from list.'
ROW= 4,COLUMN=11,HEIGHT=11,WIDTH=57,WINDOW=11,POPUP= 0,BORDER=
2,HEADLEN=55,
RETURN=0
MULTI=0
HEADING='Select the request you want to execute and press ENTER:'
DATA='   '
GOTO='        ',VALUE='* FOCEXEC'
$
DISPLAY=BORDER,$
HIDE=BANNER,$
HIDE=MAIN,$
HIDE=EXECTYPE,$
```

Compiling the Transfer File

The transfer file can be executed in its current format, but it may execute slowly, and uses a large amount of memory. You can make your window application more efficient, requiring less time and memory for execution, by compiling it.

You can compile a transfer file using the WINDOW COMPILE command. This produces a new compiled window file, in the same format as the window files produced by Window Painter.

Note that before you can issue this command, a PDS with LRECL 4096 and RECFM F must have already been allocated to ddname FMU. However, you do not need to create this PDS if you are only going to use the transfer file during the current FOCUS session: Window Painter temporarily allocates the PDS.

Syntax: How to Compile a Transfer File

```
WINDOW COMPILE windowfile
```

where:

windowfile

Is the name of the transfer file.

This must be a member name of a member of a PDS allocated to ddname TRF.

The command creates a new member of the PDS allocated to ddname FMU, with the same member name specified in the command.

When a Dialogue Manager -WINDOW command is encountered in a FOCEXEC, FOCUS searches for a compiled window file (an FMU file) with the specified file name. If the compiled file is not found, the transfer file (TRF file) with the same file name is used.

Note that if you compile a transfer file and later make changes to it, you need to recompile the updated transfer file: otherwise, FOCUS continues to use the older, unchanged compiled file.

Appendix A

Master Files and Diagrams

This appendix contains descriptions and structure diagrams for the sample data sources used throughout the documentation.

In this appendix:

- EMPLOYEE Data Source
- JOBFILE Data Source
- EDUCFILE Data Source
- SALES Data Source
- PROD Data Source
- CAR Data Source
- LEDGER Data Source
- FINANCE Data Source
- REGION Data Source
- COURSES Data Source
- EMPDATA Data Source
- EXPERSON Data Source
- TRAINING Data Source
- COURSE Data Source
- JOBHIST Data Source
- JOBLIST Data Source
- LOCATOR Data Source
- PERSINFO Data Source
- SALHIST Data Source
- PAYHIST File
- COMASTER File
- VIDEOTRK, MOVIES, and ITEMS Data Sources
- VIDEOTR2 Data Source
- Gotham Grinds Data Sources
- Century Corp Data Sources

EMPLOYEE Data Source

EMPLOYEE contains sample data about company employees. Its segments are:

`EMPINFO`

Contains employee IDs, names, and positions.

`FUNDTRAN`

Specifies employee direct deposit accounts. This segment is unique.

`PAYINFO`

Contains the employee salary history.

`ADDRESS`

Contains employee home and bank addresses.

`SALINFO`

Contains data on employee monthly pay.

`DEDUCT`

Contains data on monthly pay deductions.

EMPLOYEE also contains cross-referenced segments belonging to the JOBFILE and EDUCFILE files, also described in this appendix. The segments are:

`JOBSEG (from JOBFILE)`

Describes the job positions held by each employee.

`SKILLSEG (from JOBFILE)`

Lists the skills required by each position.

`SECSEG (from JOBFILE)`

Specifies the security clearance needed for each job position.

`ATTNDSEG (from EDUCFILE)`

Lists the dates that employees attended in-house courses.

```
COURSEG (from EDUCFILE)
```

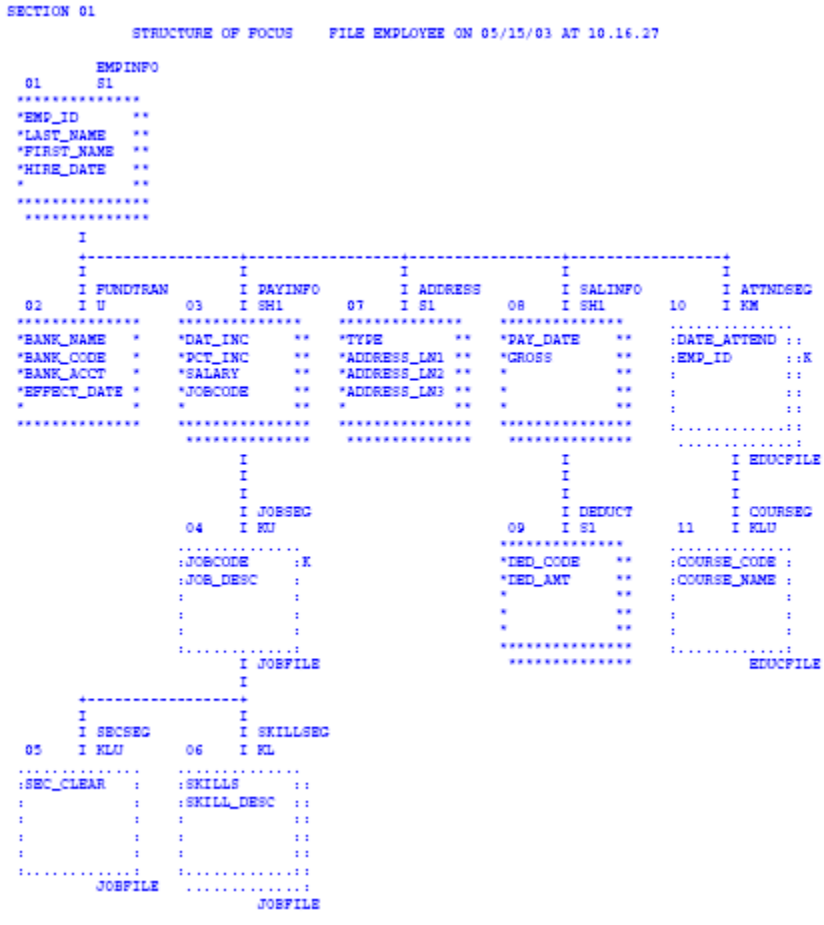
Lists the courses that the employees attended.

EMPLOYEE Master File

```
FILENAME=EMPLOYEE, SUFFIX=FOC
 SEGNAME=EMPINFO,  SEGTYPE=S1
  FIELDNAME=EMP_ID,         ALIAS=EID,     FORMAT=A9,           $
  FIELDNAME=LAST_NAME,      ALIAS=LN,      FORMAT=A15,          $
  FIELDNAME=FIRST_NAME,     ALIAS=FN,      FORMAT=A10,          $
  FIELDNAME=HIRE_DATE,      ALIAS=HDT,     FORMAT=I6YMD,        $
  FIELDNAME=DEPARTMENT,     ALIAS=DPT,     FORMAT=A10,          $
  FIELDNAME=CURR_SAL,       ALIAS=CSAL,    FORMAT=D12.2M,       $
  FIELDNAME=CURR_JOBCODE,   ALIAS=CJC,     FORMAT=A3,           $
  FIELDNAME=ED_HRS,         ALIAS=OJT,     FORMAT=F6.2,         $
 SEGNAME=FUNDTRAN,  SEGTYPE=U,   PARENT=EMPINFO
  FIELDNAME=BANK_NAME,      ALIAS=BN,      FORMAT=A20,          $
  FIELDNAME=BANK_CODE,      ALIAS=BC,      FORMAT=I6S,          $
  FIELDNAME=BANK_ACCT,      ALIAS=BA,      FORMAT=I9S,          $
  FIELDNAME=EFFECT_DATE,    ALIAS=EDATE,   FORMAT=I6YMD,        $
 SEGNAME=PAYINFO,   SEGTYPE=SH1, PARENT=EMPINFO
  FIELDNAME=DAT_INC,        ALIAS=DI,      FORMAT=I6YMD,        $
  FIELDNAME=PCT_INC,        ALIAS=PI,      FORMAT=F6.2,         $
  FIELDNAME=SALARY,         ALIAS=SAL,     FORMAT=D12.2M,       $
  FIELDNAME=JOBCODE,        ALIAS=JBC,     FORMAT=A3,           $
 SEGNAME=ADDRESS,   SEGTYPE=S1,  PARENT=EMPINFO
  FIELDNAME=TYPE,           ALIAS=AT,      FORMAT=A4,           $
  FIELDNAME=ADDRESS_LN1,    ALIAS=LN1,     FORMAT=A20,          $
  FIELDNAME=ADDRESS_LN2,    ALIAS=LN2,     FORMAT=A20,          $
  FIELDNAME=ADDRESS_LN3,    ALIAS=LN3,     FORMAT=A20,          $
  FIELDNAME=ACCTNUMBER,     ALIAS=ANO,     FORMAT=I9L,          $
 SEGNAME=SALINFO,   SEGTYPE=SH1, PARENT=EMPINFO
  FIELDNAME=PAY_DATE,       ALIAS=PD,      FORMAT=I6YMD,        $
  FIELDNAME=GROSS,          ALIAS=MO_PAY,  FORMAT=D12.2M,       $
 SEGNAME=DEDUCT,    SEGTYPE=S1,  PARENT=SALINFO
  FIELDNAME=DED_CODE,       ALIAS=DC,      FORMAT=A4,           $
  FIELDNAME=DED_AMT,        ALIAS=DA,      FORMAT=D12.2M,       $
 SEGNAME=JOBSEG,    SEGTYPE=KU,  PARENT=PAYINFO, CRFILE=JOBFILE,
  CRKEY=JOBCODE,$
 SEGNAME=SECSEG,    SEGTYPE=KLU, PARENT=JOBSEG,   CRFILE=JOBFILE, $
 SEGNAME=SKILLSEG,  SEGTYPE=KL,  PARENT=JOBSEG,   CRFILE=JOBFILE, $
 SEGNAME=ATTNDSEG,  SEGTYPE=KM,  PARENT=EMPINFO,  CRFILE=EDUCFILE,
  CRKEY=EMP_ID,$
 SEGNAME=COURSEG,   SEGTYPE=KLU, PARENT=ATTNDSEG, CRFILE=EDUCFILE,$
```

EMPLOYEE Structure Diagram

The EMPLOYEE structure follows:



JOBFILE Data Source

JOBFILE contains sample data about company job positions. Its segments are:

JOBSEG

Describes what each position is. The field JOBCODE in this segment is indexed.

SKILLSEG

Lists the skills required by each position.

SECSEG

Specifies the security clearance needed, if any. This segment is unique.

JOBFILE Master File

```
FILENAME=JOBFILE,  SUFFIX=FOC
 SEGNAME=JOBSEG,   SEGTYPE=S1
  FIELDNAME=JOBCODE,     ALIAS=JC, FORMAT=A3,     INDEX=I,$
  FIELDNAME=JOB_DESC,    ALIAS=JD, FORMAT=A25             ,$
 SEGNAME=SKILLSEG, SEGTYPE=S1,  PARENT=JOBSEG
  FIELDNAME=SKILLS,      ALIAS=,   FORMAT=A4              ,$
  FIELDNAME=SKILL_DESC, ALIAS=SD, FORMAT=A30              ,$
 SEGNAME=SECSEG,   SEGTYPE=U,   PARENT=JOBSEG
  FIELDNAME=SEC_CLEAR,  ALIAS=SC, FORMAT=A6               ,$
```

JOBFILE Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE JOBFILE ON 05/15/03 AT 14.40.06

          JOBSEG
  01      S1
 **************
 *JOBCODE     **I
 *JOB_DESC    **
 *            **
 *            **
 *            **
 ***************
  **************
        I
        +-----------------+
        I                 I
        I SECSEG          I SKILLSEG
  02    I U         03    I S1
 **************    *************
 *SEC_CLEAR   *    *SKILLS     **
 *            *    *SKILL_DESC **
 *            *    *           **
 *            *    *           **
 *            *    *           **
 **************    **************
                    *************
```

EDUCFILE Data Source

EDUCFILE contains sample data about company in-house courses. Its segments are:

COURSEG

Contains data on each course.

ATTNDSEG

Specifies which employees attended the courses. Both fields in the segment are key fields. The field EMP_ID in this segment is indexed.

EDUCFILE Master File

```
FILENAME=EDUCFILE, SUFFIX=FOC
 SEGNAME=COURSEG,  SEGTYPE=S1
  FIELDNAME=COURSE_CODE,  ALIAS=CC,   FORMAT=A6,          $
  FIELDNAME=COURSE_NAME,  ALIAS=CD,   FORMAT=A30,         $
 SEGNAME=ATTNDSEG,  SEGTYPE=SH2,  PARENT=COURSEG
  FIELDNAME=DATE_ATTEND,  ALIAS=DA,   FORMAT=I6YMD,       $
  FIELDNAME=EMP_ID,       ALIAS=EID,  FORMAT=A9, INDEX=I, $
```

EDUCFILE Structure Diagram

```
SECTION 01
              STRUCTURE OF FOCUS    FILE EDUCFILE ON 05/15/03 AT 14.45.44

          COURSEG
  01      S1
 **************
 *COURSE_CODE **
 *COURSE_NAME **
 *            **
 *            **
 *            **
 ***************
  **************
        I
        I
        I
        I ATTNDSEG
  02    I SH2
 **************
 *DATE_ATTEND **
 *EMP_ID      **I
 *            **
 *            **
 *            **
 ***************
  **************
```

SALES Data Source

SALES contains sample data about a dairy company with an affiliated store chain. Its segments are:

STOR_SEG

Lists the stores buying the products.

DAT_SEG

Contains the dates of inventory.

PRODUCT

Contains sales data for each product on each date. The PROD_CODE field is indexed. The RETURNS and DAMAGED fields have the MISSING=ON attribute.

SALES Master File

```
FILENAME=KSALES,    SUFFIX=FOC
 SEGNAME=STOR_SEG, SEGTYPE=S1
  FIELDNAME=STORE_CODE,  ALIAS=SNO,    FORMAT=A3,    $
  FIELDNAME=CITY,        ALIAS=CTY,    FORMAT=A15,   $
  FIELDNAME=AREA,        ALIAS=LOC,    FORMAT=A1,    $
 SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
  FIELDNAME=DATE,        ALIAS=DTE,    FORMAT=A4MD, $
 SEGNAME=PRODUCT,  PARENT=DATE_SEG, SEGTYPE=S1,
  FIELDNAME=PROD_CODE,   ALIAS=PCODE, FORMAT=A3,    FIELDTYPE=I,$
  FIELDNAME=UNIT_SOLD,   ALIAS=SOLD,  FORMAT=I5,    $
  FIELDNAME=RETAIL_PRICE,ALIAS=RP,    FORMAT=D5.2M,$
  FIELDNAME=DELIVER_AMT, ALIAS=SHIP,  FORMAT=I5,    $
  FIELDNAME=OPENING_AMT, ALIAS=INV,   FORMAT=I5,    $
  FIELDNAME=RETURNS,     ALIAS=RTN,   FORMAT=I3,    MISSING=ON,$
  FIELDNAME=DAMAGED,     ALIAS=BAD,   FORMAT=I3,    MISSING=ON,$
```

SALES Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE SALES ON 05/15/03 AT 14.50.28

         STOR_SEG
 01      S1
 **************
 *STORE_CODE  **
 *CITY        **
 *AREA        **
 *            **
 *            **
 ***************
  **************
       I
       I
       I
       I DATE_SEG
 02    I SH1
 **************
 *DATE        **
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
       I
       I
       I
       I PRODUCT
 03    I S1
 **************
 *PROD_CODE   **I
 *UNIT_SOLD   **
 *RETAIL_PRICE**
 *DELIVER_AMT **
 *            **
 ***************
  **************
```

PROD Data Source

The PROD data source lists products sold by a dairy company. It consists of one segment, PRODUCT. The field PROD_CODE is indexed.

PROD Master File

```
FILE=KPROD, SUFFIX=FOC
 SEGMENT=PRODUCT, SEGTYPE=S1,
  FIELDNAME=PROD_CODE, ALIAS=PCODE, FORMAT=A3,    FIELDTYPE=I, $
  FIELDNAME=PROD_NAME, ALIAS=ITEM,  FORMAT=A15,   $
  FIELDNAME=PACKAGE,   ALIAS=SIZE,  FORMAT=A12,   $
  FIELDNAME=UNIT_COST, ALIAS=COST,  FORMAT=D5.2M, $
```

PROD Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS   FILE PROD   ON 05/15/03 AT 14.57.38
       PRODUCT
01     S1
*************
*PROD_CODE   **I
*PROD_NAME   **
*PACKAGE     **
*UNIT_COST   **
*            **
**************
 *************
```

CAR Data Source

CAR contains sample data about specifications and sales information for rare cars. Its segments are:

`ORIGIN`

Lists the country that manufactures the car. The field COUNTRY is indexed.

`COMP`

Contains the car name.

`CARREC`

Contains the car model.

`BODY`

Lists the body type, seats, dealer and retail costs, and units sold.

SPECS

Lists car specifications. This segment is unique.

WARANT

Lists the type of warranty.

EQUIP

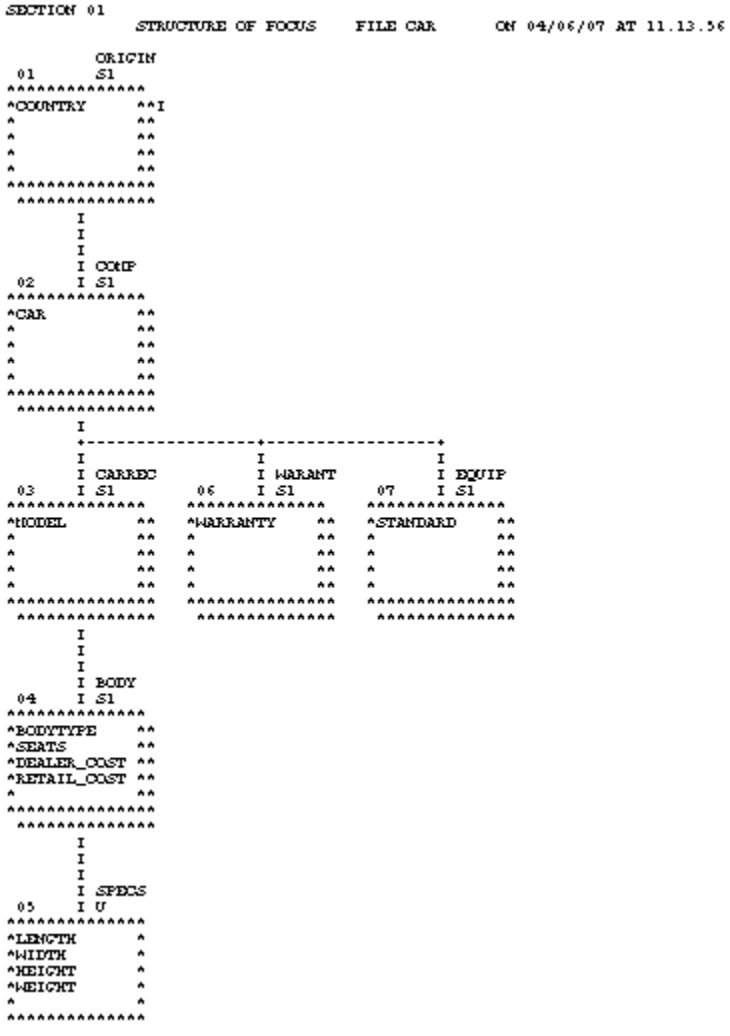
Lists standard equipment.

The aliases in the CAR Master File are specified without the ALIAS keyword.

CAR Master File

```
FILENAME=CAR,SUFFIX=FOC
 SEGNAME=ORIGIN,SEGTYPE=S1
  FIELDNAME=COUNTRY,COUNTRY,A10,FIELDTYPE=I,$
 SEGNAME=COMP,SEGTYPE=S1,PARENT=ORIGIN
  FIELDNAME=CAR,CARS,A16,$
 SEGNAME=CARREC,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=MODEL,MODEL,A24,$
 SEGNAME=BODY,SEGTYPE=S1,PARENT=CARREC
  FIELDNAME=BODYTYPE,TYPE,A12,$
  FIELDNAME=SEATS,SEAT,I3,$
  FIELDNAME=DEALER_COST,DCOST,D7,$
  FIELDNAME=RETAIL_COST,RCOST,D7,$
  FIELDNAME=SALES,UNITS,I6,$
 SEGNAME=SPECS,SEGTYPE=U,PARENT=BODY
  FIELDNAME=LENGTH,LEN,D5,$
  FIELDNAME=WIDTH,WIDTH,D5,$
  FIELDNAME=HEIGHT,HEIGHT,D5,$
  FIELDNAME=WEIGHT,WEIGHT,D6,$
  FIELDNAME=WHEELBASE,BASE,D6.1,$
  FIELDNAME=FUEL_CAP,FUEL,D6.1,$
  FIELDNAME=BHP,POWER,D6,$
  FIELDNAME=RPM,RPM,I5,$
  FIELDNAME=MPG,MILES,D6,$
  FIELDNAME=ACCEL,SECONDS,D6,$
 SEGNAME=WARANT,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=WARRANTY,WARR,A40,$
 SEGNAME=EQUIP,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=STANDARD,EQUIP,A40,$
```

CAR Structure Diagram



LEDGER Data Source

LEDGER contains sample accounting data. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

LEDGER Master File

```
FILENAME=LEDGER, SUFFIX=FOC,$
 SEGNAME=TOP,    SEGTYPE=S2,$
  FIELDNAME=YEAR    ,   , FORMAT=A4, $
  FIELDNAME=ACCOUNT,   , FORMAT=A4, $
  FIELDNAME=AMOUNT ,   , FORMAT=I5C,$
```

LEDGER Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE LEDGER   ON 05/15/03 AT 15.17.08

          TOP
  01      S2
 **************
 *YEAR         **
 *ACCOUNT      **
 *AMOUNT       **
 *             **
 *             **
 ***************
  **************
```

FINANCE Data Source

FINANCE contains sample financial data for balance sheets. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

FINANCE Master File

```
FILENAME=FINANCE, SUFFIX=FOC,$
 SEGNAME=TOP,     SEGTYPE=S2,$
  FIELDNAME=YEAR    ,   , FORMAT=A4,   $
  FIELDNAME=ACCOUNT,   , FORMAT=A4,   $
  FIELDNAME=AMOUNT ,   , FORMAT=D12C,$
```

FINANCE Structure Diagram

```
SECTION 01
           STRUCTURE OF FOCUS    FILE FINANCE  ON 05/15/03 AT 15.17.08

          TOP
 01       S2
 **************
 *YEAR         **
 *ACCOUNT      **
 *AMOUNT       **
 *             **
 *             **
 ***************
  **************
```

REGION Data Source

REGION contains sample account data for the eastern and western regions of the country. It consists of one segment, TOP. This data source is specified primarily for FML examples. Aliases do not exist for the fields in this Master File, and the commas act as placeholders.

REGION Master File

```
FILENAME=REGION, SUFFIX=FOC,$
 SEGNAME=TOP,    SEGTYPE=S1,$
  FIELDNAME=ACCOUNT,  , FORMAT=A4, $
  FIELDNAME=E_ACTUAL, , FORMAT=I5C,$
  FIELDNAME=E_BUDGET, , FORMAT=I5C,$
  FIELDNAME=W_ACTUAL, , FORMAT=I5C,$
  FIELDNAME=W_BUDGET, , FORMAT=I5C,$
```

REGION Structure Diagram

```
SECTION 01
           STRUCTURE OF FOCUS    FILE REGION   ON 05/15/03 AT 15.18.48

          TOP
 01       S1
 **************
 *ACCOUNT      **
 *E_ACTUAL     **
 *E_BUDGET     **
 *W_ACTUAL     **
 *             **
 ***************
  **************
```

COURSES Data Source

COURSES contains sample data about education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

COURSES Master File

```
FILENAME=COURSES,  SUFFIX=FOC,$
 SEGNAME=CRSESEG1, SEGTYPE=S1, $
  FIELDNAME=COURSE_CODE,  ALIAS=CC,    FORMAT=A6,   FIELDTYPE=I, $
  FIELDNAME=COURSE_NAME,  ALIAS=CN,    FORMAT=A30,               $
  FIELDNAME=DURATION,     ALIAS=DAYS,  FORMAT=I3,                $
  FIELDNAME=DESCRIPTION,  ALIAS=CDESC, FORMAT=TX50,              $
```

COURSES Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE COURSES   ON 05/15/03 AT 12.26.05

        CRSESEG1
 01     S1
**************
*COURSE_CODE **I
*COURSE_NAME **
*DURATION    **
*DESCRIPTION **T
*            **
***************
 ***************
```

EMPDATA Data Source

EMPDATA contains sample data about company employees. It consists of one segment, EMPDATA. The PIN field is indexed. The AREA field is a temporary field.

EMPDATA Master File

```
FILENAME=EMPDATA, SUFFIX=FOC
 SEGNAME=EMPDATA, SEGTYPE=S1
  FIELDNAME=PIN,          ALIAS=ID,       FORMAT=A9,  INDEX=I,    $
  FIELDNAME=LASTNAME,     ALIAS=LN,       FORMAT=A15,             $
  FIELDNAME=FIRSTNAME,    ALIAS=FN,       FORMAT=A10,             $
  FIELDNAME=MIDINITIAL,   ALIAS=MI,       FORMAT=A1,              $
  FIELDNAME=DIV,          ALIAS=CDIV,     FORMAT=A4,              $
  FIELDNAME=DEPT,         ALIAS=CDEPT,    FORMAT=A20,             $
  FIELDNAME=JOBCLASS,     ALIAS=CJCLAS,   FORMAT=A8,              $
  FIELDNAME=TITLE,        ALIAS=CFUNC,    FORMAT=A20,             $
  FIELDNAME=SALARY,       ALIAS=CSAL,     FORMAT=D12.2M,          $
  FIELDNAME=HIREDATE,     ALIAS=HDAT,     FORMAT=YMD,             $
$
DEFINE AREA/A13=DECODE DIV (NE 'NORTH EASTERN' SE 'SOUTH EASTERN'
CE 'CENTRAL' WE 'WESTERN' CORP 'CORPORATE' ELSE 'INVALID AREA');$
```

EMPDATA Structure Diagram

```
SECTION 01
           STRUCTURE OF FOCUS    FILE EMPDATA  ON 05/15/03 AT 14.49.09

         EMPDATA
  01     S1
 *************
 *PIN         **I
 *LASTNAME    **
 *FIRSTNAME   **
 *MIDINITIAL  **
 *            **
 ***************
  *************
```

EXPERSON Data Source

The EXPERSON data source contains personal data about individual employees. It consists of one segment, ONESEG.

EXPERSON Master File

```
FILE=EXPERSON    ,SUFFIX=FOC
 SEGMENT=ONESEG, $
  FIELDNAME=SOC_SEC_NO   ,ALIAS=SSN        ,USAGE=A9        ,$
  FIELDNAME=FIRST_NAME   ,ALIAS=FN         ,USAGE=A9        ,$
  FIELDNAME=LAST_NAME    ,ALIAS=LN         ,USAGE=A10       ,$
  FIELDNAME=AGE          ,ALIAS=YEARS      ,USAGE=I2        ,$
  FIELDNAME=SEX          ,ALIAS=           ,USAGE=A1        ,$
  FIELDNAME=MARITAL_STAT ,ALIAS=MS         ,USAGE=A1        ,$
  FIELDNAME=NO_DEP       ,ALIAS=NDP        ,USAGE=I3        ,$
  FIELDNAME=DEGREE       ,ALIAS=           ,USAGE=A3        ,$
  FIELDNAME=NO_CARS      ,ALIAS=CARS       ,USAGE=I3        ,$
  FIELDNAME=ADDRESS      ,ALIAS=           ,USAGE=A14       ,$
  FIELDNAME=CITY         ,ALIAS=           ,USAGE=A10       ,$
  FIELDNAME=WAGE         ,ALIAS=PAY        ,USAGE=D10.2SM   ,$
  FIELDNAME=CATEGORY     ,ALIAS=STATUS     ,USAGE=A1        ,$
  FIELDNAME=SKILL_CODE   ,ALIAS=SKILLS     ,USAGE=A5        ,$
  FIELDNAME=DEPT_CODE    ,ALIAS=WHERE      ,USAGE=A4        ,$
  FIELDNAME=TEL_EXT      ,ALIAS=EXT        ,USAGE=I4        ,$
  FIELDNAME=DATE_EMP     ,ALIAS=BASE_DATE  ,USAGE=I6YMTD    ,$
  FIELDNAME=MULTIPLIER   ,ALIAS=RATIO      ,USAGE=D5.3      ,$
```

EXPERSON Structure Diagram

```
SECTION 01
           STRUCTURE OF FOCUS    FILE EXPERSON  ON 05/15/03 AT 14.50.58

         ONESEG
  01     S1
 **************
 *SOC_SEC_NO  **
 *FIRST_NAME  **
 *LAST_NAME   **
 *AGE         **
 *            **
 ***************
  **************
```

TRAINING Data Source

TRAINING contains sample data about training courses for employees. It consists of one segment, TRAINING. The PIN field is indexed. The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

TRAINING Master File

```
FILENAME=TRAINING, SUFFIX=FOC
 SEGNAME=TRAINING, SEGTYPE=SH3
  FIELDNAME=PIN,          ALIAS=ID,     FORMAT=A9,   INDEX=I,    $
  FIELDNAME=COURSESTART,  ALIAS=CSTART, FORMAT=YMD,              $
  FIELDNAME=COURSECODE,   ALIAS=CCOD,   FORMAT=A7,               $
  FIELDNAME=EXPENSES,     ALIAS=COST,   FORMAT=D8.2, MISSING=ON  $
  FIELDNAME=GRADE,        ALIAS=GRA,    FORMAT=A2,   MISSING=ON, $
  FIELDNAME=LOCATION,     ALIAS=LOC,    FORMAT=A6,   MISSING=ON, $
```

TRAINING Structure Diagram

```
SECTION 01
           STRUCTURE OF FOCUS    FILE TRAINING ON 05/15/03 AT 14.51.28

         TRAINING
  01     SH3
 **************
 *PIN         **I
 *COURSESTART **
 *COURSECODE  **
 *EXPENSES    **
 *            **
 ***************
  **************
```

COURSE Data Source

COURSE contains sample data about education courses. It consists of one segment, CRSELIST.

COURSE Master File

```
FILENAME=COURSE,   SUFFIX=FOC
 SEGNAME=CRSELIST,  SEGTYPE=S1
  FIELDNAME=COURSECODE,  ALIAS=CCOD,   FORMAT=A7,   INDEX=I,    $
  FIELDNAME=CTITLE,      ALIAS=COURSE, FORMAT=A35,              $
  FIELDNAME=SOURCE,      ALIAS=ORG,    FORMAT=A35,              $
  FIELDNAME=CLASSIF,     ALIAS=CLASS,  FORMAT=A10,              $
  FIELDNAME=TUITION,     ALIAS=FEE,    FORMAT=D8.2, MISSING=ON, $
  FIELDNAME=DURATION,    ALIAS=DAYS,   FORMAT=A3,   MISSING=ON, $
  FIELDNAME=DESCRIPTN1,  ALIAS=DESC1,  FORMAT=A40,              $
  FIELDNAME=DESCRIPTN2,  ALIAS=DESC2,  FORMAT=A40,              $
  FIELDNAME=DESCRIPTN2,  ALIAS=DESC3,  FORMAT=A40,              $
```

COURSE Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE COURSE   ON 05/15/03 AT 12.26.05

         CRSELIST
 01      S1
**************
*COURSECODE  **I
*CTITLE      **
*SOURCE      **
*CLASSIF     **
*            **
***************
 **************
```

JOBHIST Data Source

JOBHIST contains information about employee jobs. Both the PIN and JOBSTART fields are keys. The PIN field is indexed.

JOBHIST Master File

```
FILENAME=JOBHIST, SUFFIX=FOC
SEGNAME=JOBHIST, SEGTYPE=SH2
 FIELDNAME=PIN,          ALIAS=ID,       FORMAT=A9,      INDEX=I ,$
 FIELDNAME=JOBSTART,     ALIAS=SDAT,     FORMAT=YMD,              $
 FIELDNAME=JOBCLASS,     ALIAS=JCLASS,   FORMAT=A8,               $
 FIELDNAME=FUNCTITLE,    ALIAS=FUNC,     FORMAT=A20,              $
```

JOBHIST Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE JOBHIST  ON 01/22/08 AT 16.23.46
    JOBHIST
  01      SH2
 **************
 *PIN         **I
 *JOBSTART    **
 *JOBCLASS    **
 *FUNCTITLE   **
 *            **
 ***************
  **************
```

JOBLIST Data Source

JOBLIST contains information about jobs. The JOBCLASS field is indexed.

JOBLIST Master File

```
FILENAME=JOBLIST, SUFFIX=FOC
SEGNAME=JOBSEG,   SEGTYPE=S1
 FIELDNAME=JOBCLASS,     ALIAS=JCLASS,   FORMAT=A8,    INDEX=I ,$
 FIELDNAME=CATEGORY,     ALIAS=JGROUP,   FORMAT=A25,            $
 FIELDNAME=JOBDESC,      ALIAS=JDESC,    FORMAT=A40,            $
 FIELDNAME=LOWSAL,       ALIAS=LSAL,     FORMAT=D12.2M,         $
 FIELDNAME=HIGHSAL,      ALIAS=HSAL,     FORMAT=D12.2M,         $
DEFINE GRADE/A2=EDIT  (JCLASS,'$$$99');$
DEFINE LEVEL/A25=DECODE GRADE (08 'GRADE 8' 09 'GRADE 9' 10
'GRADE 10' 11 'GRADE 11' 12 'GRADE 12' 13 'GRADE 13' 14 'GRADE 14');$
```

JOBLIST Structure Diagram

```
SECTION 01
             STRUCTURE OF FOCUS    FILE JOBLIST  ON 01/22/08 AT 16.24.52
         JOBSEG
  01     S1
 **************
 *JOBCLASS    **I
 *CATEGORY    **
 *JOBDESC     **
 *LOWSAL      **
 *            **
 ***************
  **************
```

LOCATOR Data Source

JOBHIST contains information about employee location and phone number. The PIN field is indexed.

LOCATOR Master File

```
FILENAME=LOCATOR, SUFFIX=FOC
SEGNAME=LOCATOR,   SEGTYPE=S1,
 FIELDNAME=PIN,           ALIAS=ID_NO,    FORMAT=A9,   INDEX=I, $
 FIELDNAME=SITE,          ALIAS=SITE,     FORMAT=A25,           $
 FIELDNAME=FLOOR,         ALIAS=FL,       FORMAT=A3,            $
 FIELDNAME=ZONE,          ALIAS=ZONE,     FORMAT=A2,            $
 FIELDNAME=BUS_PHONE,     ALIAS=BTEL,     FORMAT=A5,            $
```

LOCATOR Structure Diagram

```
SECTION 01
             STRUCTURE OF FOCUS    FILE LOCATOR  ON 01/22/08 AT 16.26.55
         LOCATOR
  01     S1
 **************
 *PIN          **I
 *SITE         **
 *FLOOR        **
 *ZONE         **
 *             **
 ***************
  **************
```

PERSINFO Data Source

PERSINFO contains employee personal information. The PIN field is indexed.

PERSINFO Master File

```
FILENAME=PERSINFO, SUFFIX=FOC
SEGNAME=PERSONAL, SEGTYPE=S1
 FIELDNAME=PIN,           ALIAS=ID,       FORMAT=A9,   INDEX=I,    $
 FIELDNAME=INCAREOF,      ALIAS=ICO,      FORMAT=A35,              $
 FIELDNAME=STREETNO,      ALIAS=STR,      FORMAT=A20,              $
 FIELDNAME=APT,           ALIAS=APT,      FORMAT=A4,               $
 FIELDNAME=CITY,          ALIAS=CITY,     FORMAT=A20,              $
 FIELDNAME=STATE,         ALIAS=PROV,     FORMAT=A4,               $
 FIELDNAME=POSTALCODE,    ALIAS=ZIP,      FORMAT=A10,              $
 FIELDNAME=COUNTRY,       ALIAS=CTRY,     FORMAT=A15,              $
 FIELDNAME=HOMEPHONE,     ALIAS=TEL,      FORMAT=A10,              $
 FIELDNAME=EMERGENCYNO,   ALIAS=ENO,      FORMAT=A10,              $
 FIELDNAME=EMERGCONTACT,  ALIAS=ENAME,    FORMAT=A35,              $
 FIELDNAME=RELATIONSHIP,  ALIAS=REL,      FORMAT=A8,               $
 FIELDNAME=BIRTHDATE,     ALIAS=BDAT,     FORMAT=YMD,              $
```

PERSINFO Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE PERSINFO ON 01/22/08 AT 16.27.24
   PERSONAL
  01      S1
 **************
 *PIN          **I
 *INCAREOF     **
 *STREETNO     **
 *APT          **
 *             **
 ***************
  **************
```

SALHIST Data Source

SALHIST contains information about employee salary history. The PIN field is indexed. Both the PIN and EFFECTDATE fields are keys.

SALHIST Master File

```
FILENAME=SALHIST,  SUFFIX=FOC
SEGNAME=SLHISTRY,  SEGTYPE=SH2
 FIELDNAME=PIN,        ALIAS=ID,     FORMAT=A9,     INDEX=I,  $
 FIELDNAME=EFFECTDATE, ALIAS=EDAT,   FORMAT=YMD,              $
 FIELDNAME=OLDSALARY,  ALIAS=OSAL,   FORMAT=D12.2,            $
```

SALHIST Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE SALHIST  ON 01/22/08 AT 16.28.02
    SLHISTRY
  01      SH2
 **************
 *PIN         **I
 *EFFECTDATE  **
 *OLDSALARY   **
 *            **
 *            **
 ***************
  **************
```

PAYHIST File

The PAYHIST data source contains the employees' salary history. It consists of one segment, PAYSEG. The SUFFIX attribute indicates that the data file is a fixed-format sequential file.

PAYHIST Master File

```
FILENAME=PAYHIST,  SUFFIX=FIX
 SEGMENT=PAYSEG,$
  FIELDNAME=SOC_SEC_NO,  ALIAS=SSN,      USAGE=A9,      ACTUAL=A9, $
  FIELDNAME=DATE_OF_IN,  ALIAS=INCDATE,  USAGE=I6YMTD,  ACTUAL=A6, $
  FIELDNAME=AMT_OF_INC,  ALIAS=RAISE,    USAGE=D6.2,    ACTUAL=A10,$
  FIELDNAME=PCT_INC,     ALIAS=,         USAGE=D6.2,    ACTUAL=A6, $
  FIELDNAME=NEW_SAL,     ALIAS=CURR_SAL, USAGE=D10.2,   ACTUAL=A11,$
  FIELDNAME=FILL,        ALIAS=,         USAGE=A38,     ACTUAL=A38,$
```

PAYHIST Structure Diagram

```
SECTION 01
           STRUCTURE OF FIX    FILE PAYHIST ON 05/15/03 AT 14.51.59

         PAYSEG
  01     S1
 *************
 *SOC_SEC_NO  **
 *DATE_OF_IN  **
 *AMT_OF_INC  **
 *PCT_INC     **
 *            **
 ***************
  **************
```

COMASTER File

The COMASTER file is used to display the file structure and contents of each segment in a data source. Since COMASTER is used for debugging other Master Files, a corresponding FOCEXEC does not exist for the COMASTER file. Its segments are:

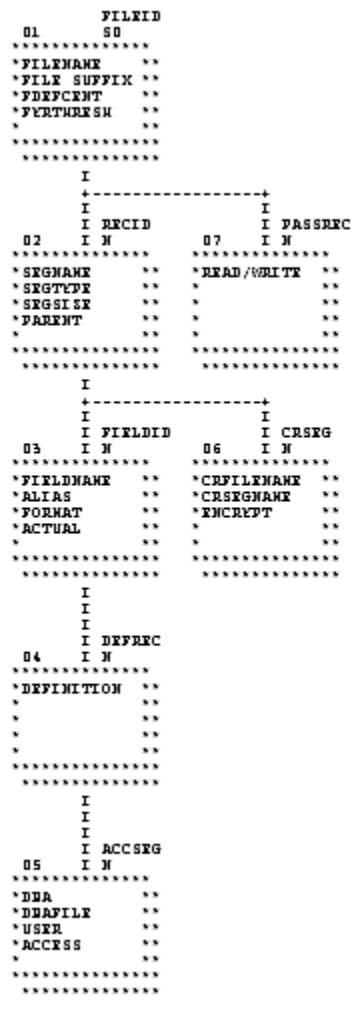
- FILEID, which lists file information.
- RECID, which lists segment information.
- FIELDID, which lists field information.
- DEFREC, which lists a description record.
- PASSREC, which lists read/write access.
- CRSEG, which lists cross-reference information for segments.
- ACCSEG, which lists DBA information.

COMASTER Master File

```
SUFFIX=COM,SEGNAME=FILEID
 FIELDNAME=FILENAME    ,FILE         ,A8 ,      ,$
 FIELDNAME=FILE SUFFIX ,SUFFIX       ,A8 ,      ,$
 FIELDNAME=FDEFCENT    ,FDFC         ,A4 ,      ,$
 FIELDNAME=FYRTHRESH   ,FYRT         ,A2 ,      ,$
SEGNAME=RECID
 FIELDNAME=SEGNAME     ,SEGMENT      ,A8 ,      ,$
 FIELDNAME=SEGTYPE     ,SEGTYPE      ,A4 ,      ,$
 FIELDNAME=SEGSIZE     ,SEGSIZE      ,I4 ,    A4,$
 FIELDNAME=PARENT      ,PARENT       ,A8 ,      ,$
 FIELDNAME=CRKEY       ,VKEY         ,A66,      ,$
SEGNAME=FIELDID
 FIELDNAME=FIELDNAME   ,FIELD        ,A66,      ,$
 FIELDNAME=ALIAS       ,SYNONYM      ,A66,      ,$
 FIELDNAME=FORMAT      ,USAGE        ,A8 ,      ,$
 FIELDNAME=ACTUAL      ,ACTUAL       ,A8 ,      ,$
 FIELDNAME=AUTHORITY   ,AUTHCODE     ,A8 ,      ,$
 FIELDNAME=FIELDTYPE   ,INDEX        ,A8 ,      ,$
 FIELDNAME=TITLE       ,TITLE        ,A64,      ,$
 FIELDNAME=HELPMESSAGE ,MESSAGE      ,A256,     ,$
 FIELDNAME=MISSING     ,MISSING      ,A4 ,      ,$
 FIELDNAME=ACCEPTS     ,ACCEPTABLE   ,A255,     ,$
 FIELDNAME=RESERVED    ,RESERVED     ,A44 ,     ,$
 FIELDNAME=DEFCENT     ,DFC          ,A4 ,      ,$
 FIELDNAME=YRTHRESH    ,YRT          ,A4 ,      ,$
SEGNAME=DEFREC
 FIELDNAME=DEFINITION  ,DESCRIPTION  ,A44,      ,$
SEGNAME=PASSREC,PARENT=FILEID
 FIELDNAME=READ/WRITE  ,RW           ,A32,      ,$
SEGNAME=CRSEG,PARENT=RECID
 FIELDNAME=CRFILENAME  ,CRFILE       ,A8 ,      ,$
 FIELDNAME=CRSEGNAME   ,CRSEGMENT    ,A8 ,      ,$
 FIELDNAME=ENCRYPT     ,ENCRYPT      ,A4 ,      ,$
SEGNAME=ACCSEG,PARENT=DEFREC
 FIELDNAME=DBA         ,DBA          ,A8 ,      ,$
 FIELDNAME=DBAFILE     ,             ,A8 ,      ,$
 FIELDNAME=USER        ,PASS         ,A8 ,      ,$
 FIELDNAME=ACCESS      ,ACCESS       ,A8 ,      ,$
 FIELDNAME=RESTRICT    ,RESTRICT     ,A8 ,      ,$
 FIELDNAME=NAME        ,NAME         ,A66,      ,$
 FIELDNAME=VALUE       ,VALUE        ,A80,      ,$
```

COMASTER Structure Diagram

```
SECTION 01
                STRUCTURE OF EXTERNAL FILE COMASTER ON 05/15/03 AT 14.53.38
```



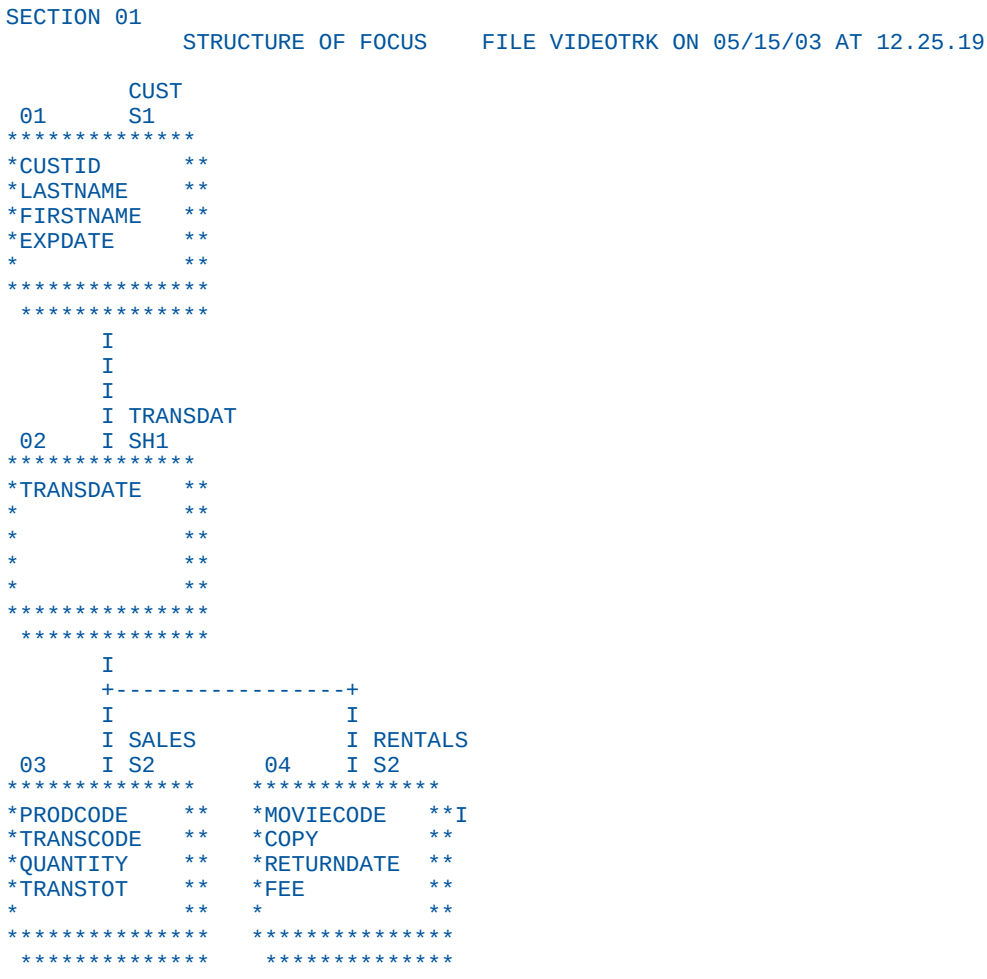
VIDEOTRK, MOVIES, and ITEMS Data Sources

VIDEOTRK contains sample data about customer, rental, and purchase information for a video rental business. It can be joined to the MOVIES or ITEMS data source. VIDEOTRK and MOVIES are used in examples that illustrate the use of the Maintain Data facility.

VIDEOTRK Master File

```
FILENAME=VIDEOTRK, SUFFIX=FOC
 SEGNAME=CUST,     SEGTYPE=S1
  FIELDNAME=CUSTID,     ALIAS=CIN,          FORMAT=A4,    $
  FIELDNAME=LASTNAME,   ALIAS=LN,           FORMAT=A15,   $
  FIELDNAME=FIRSTNAME,  ALIAS=FN,           FORMAT=A10,   $
  FIELDNAME=EXPDATE,    ALIAS=EXDAT,        FORMAT=YMD,   $
  FIELDNAME=PHONE,      ALIAS=TEL,          FORMAT=A10,   $
  FIELDNAME=STREET,     ALIAS=STR,          FORMAT=A20,   $
  FIELDNAME=CITY,       ALIAS=CITY,         FORMAT=A20,   $
  FIELDNAME=STATE,      ALIAS=PROV,         FORMAT=A4,    $
  FIELDNAME=ZIP,        ALIAS=POSTAL_CODE,  FORMAT=A9,    $
SEGNAME=TRANSDAT,  SEGTYPE=SH1,  PARENT=CUST
  FIELDNAME=TRANSDATE,  ALIAS=OUTDATE,      FORMAT=YMD,   $
SEGNAME=SALES,     SEGTYPE=S2,   PARENT=TRANSDAT
  FIELDNAME=PRODCODE,   ALIAS=PCOD,         FORMAT=A6,    $
  FIELDNAME=TRANSCODE,  ALIAS=TCOD,         FORMAT=I3,    $
  FIELDNAME=QUANTITY,   ALIAS=NO,           FORMAT=I3S,   $
  FIELDNAME=TRANSTOT,   ALIAS=TTOT,         FORMAT=F7.2S, $
SEGNAME=RENTALS,   SEGTYPE=S2,   PARENT=TRANSDAT
  FIELDNAME=MOVIECODE,  ALIAS=MCOD,         FORMAT=A6, INDEX=I, $
  FIELDNAME=COPY,       ALIAS=COPY,         FORMAT=I2,    $
  FIELDNAME=RETURNDATE, ALIAS=INDATE,       FORMAT=YMD,   $
  FIELDNAME=FEE,        ALIAS=FEE,          FORMAT=F5.2S, $
```

VIDEOTRK Structure Diagram



MOVIES Master File

```
FILENAME=MOVIES,     SUFFIX=FOC
 SEGNAME=MOVINFO,    SEGTYPE=S1
  FIELDNAME=MOVIECODE,    ALIAS=MCOD,  FORMAT=A6, INDEX=I, $
  FIELDNAME=TITLE,        ALIAS=MTL,   FORMAT=A39,  $
  FIELDNAME=CATEGORY,     ALIAS=CLASS, FORMAT=A8,   $
  FIELDNAME=DIRECTOR,     ALIAS=DIR,   FORMAT=A17,  $
  FIELDNAME=RATING,       ALIAS=RTG,   FORMAT=A4,   $
  FIELDNAME=RELDATE,      ALIAS=RDAT,  FORMAT=YMD,  $
  FIELDNAME=WHOLESALEPR, ALIAS=WPRC,  FORMAT=F6.2, $
  FIELDNAME=LISTPR,       ALIAS=LPRC,  FORMAT=F6.2, $
  FIELDNAME=COPIES,       ALIAS=NOC,   FORMAT=I3,   $
```

MOVIES Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE MOVIES   ON 05/15/03 AT 12.26.05

        MOVINFO
 01     S1
**************
*MOVIECODE   **I
*TITLE       **
*CATEGORY    **
*DIRECTOR    **
*            **
***************
 **************
```

ITEMS Master File

```
FILENAME=ITEMS,    SUFFIX=FOC
 SEGNAME=ITMINFO, SEGTYPE=S1
  FIELDNAME=PRODCODE,  ALIAS=PCOD,  FORMAT=A6,  INDEX=I, $
  FIELDNAME=PRODNAME,  ALIAS=PROD,  FORMAT=A20,          $
  FIELDNAME=OURCOST,   ALIAS=WCOST, FORMAT=F6.2,         $
  FIELDNAME=RETAILPR,  ALIAS=PRICE, FORMAT=F6.2,         $
  FIELDNAME=ON_HAND,   ALIAS=NUM,   FORMAT=I5,           $
```

ITEMS Structure Diagram

```
SECTION 01
            STRUCTURE OF FOCUS    FILE ITEMS    ON 05/15/03 AT 12.26.05

         ITMINFO
 01      S1
**************
*PRODCODE    **I
*PRODNAME    **
*OURCOST     **
*RETAILPR    **
*            **
***************
 **************
```

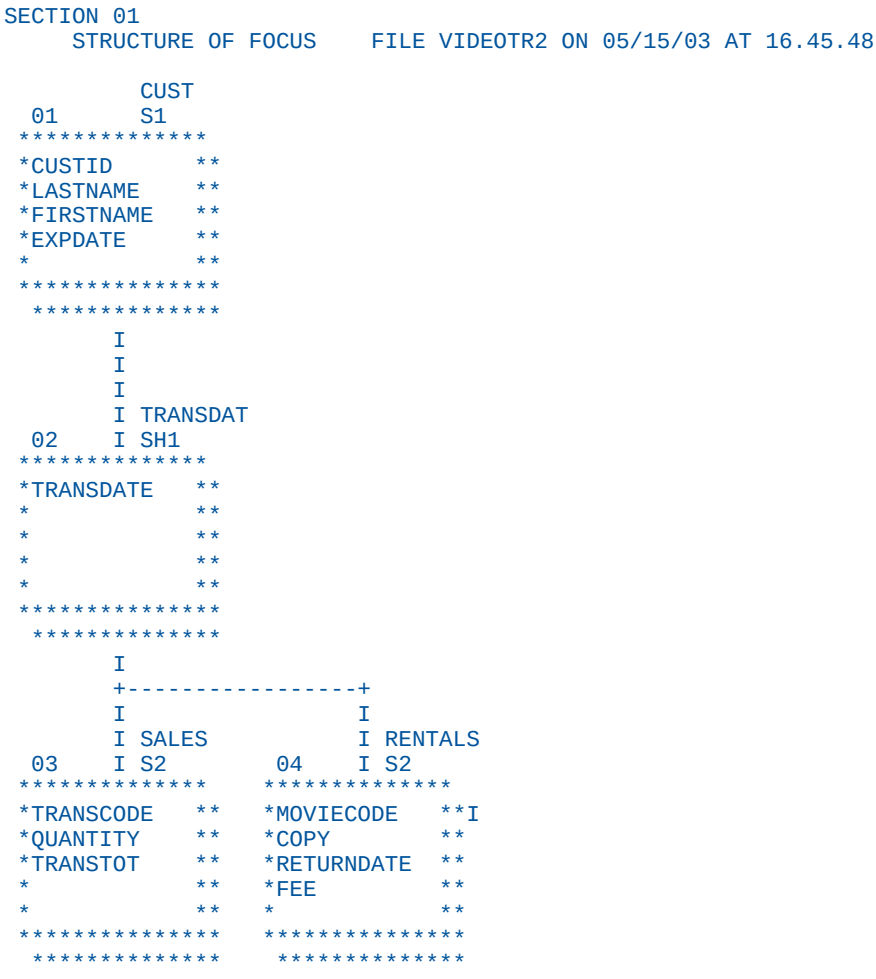
VIDEOTR2 Data Source

VIDEOTR2 contains sample data about customer, rental, and purchase information for a video rental business. It consists of four segments.

VIDEOTR2 Master File

```
FILENAME=VIDEOTR2, SUFFIX=FOC
 SEGNAME=CUST,      SEGTYPE=S1
  FIELDNAME=CUSTID,        ALIAS=CIN,          FORMAT=A4,          $
  FIELDNAME=LASTNAME,      ALIAS=LN,           FORMAT=A15,         $
  FIELDNAME=FIRSTNAME,     ALIAS=FN,           FORMAT=A10,         $
  FIELDNAME=EXPDATE,       ALIAS=EXDAT,        FORMAT=YMD,         $
  FIELDNAME=PHONE,         ALIAS=TEL,          FORMAT=A10,         $
  FIELDNAME=STREET,        ALIAS=STR,          FORMAT=A20,         $
  FIELDNAME=CITY,          ALIAS=CITY,         FORMAT=A20,         $
  FIELDNAME=STATE,         ALIAS=PROV,         FORMAT=A4,          $
  FIELDNAME=ZIP,           ALIAS=POSTAL_CODE,  FORMAT=A9,          $
  FIELDNAME=EMAIL,         ALIAS=EMAIL,        FORMAT=A18,         $
 SEGNAME=TRANSDAT, SEGTYPE=SH1,  PARENT=CUST
  FIELDNAME=TRANSDATE,     ALIAS=OUTDATE,      FORMAT=HYYMDI,      $
 SEGNAME=SALES,    SEGTYPE=S2,   PARENT=TRANSDAT
  FIELDNAME=TRANSCODE,     ALIAS=TCOD,         FORMAT=I3,          $
  FIELDNAME=QUANTITY,      ALIAS=NO,           FORMAT=I3S,         $
  FIELDNAME=TRANSTOT,      ALIAS=TTOT,         FORMAT=F7.2S,       $
 SEGNAME=RENTALS,  SEGTYPE=S2,   PARENT=TRANSDAT
  FIELDNAME=MOVIECODE,     ALIAS=MCOD,         FORMAT=A6, INDEX=I, $
  FIELDNAME=COPY,          ALIAS=COPY,         FORMAT=I2,          $
  FIELDNAME=RETURNDATE,    ALIAS=INDATE,       FORMAT=YMD,         $
  FIELDNAME=FEE,           ALIAS=FEE,          FORMAT=F5.2S,       $
```

VIDEOTR2 Structure Diagram



Gotham Grinds Data Sources

Gotham Grinds is a group of data sources that contain sample data about a specialty items company.

- GGDEMOG contains demographic information about the customers of Gotham Grinds, a company that sells specialty items like coffee, gourmet snacks, and gifts. It consists of one segment, DEMOG01.
- GGORDER contains order information for Gotham Grinds. It consists of two segments, ORDER01 and ORDER02.

- GGPRODS contains product information for Gotham Grinds. It consists of one segment, PRODS01.
- GGSALES contains sales information for Gotham Grinds. It consists of one segment, SALES01.
- GGSTORES contains information for each of Gotham Grinds 12 stores in the United States. It consists of one segment, STORES01.

GGDEMOG Master File

```
FILENAME=GGDEMOG, SUFFIX=FOC
 SEGNAME=DEMOG01, SEGTYPE=S1
  FIELD=ST,       ALIAS=E02, FORMAT=A02, INDEX=I,TITLE='State',
   DESC='State',$
  FIELD=HH,       ALIAS=E03, FORMAT=I09, TITLE='Number of Households',
   DESC='Number of Households',$
  FIELD=AVGHHSZ98,ALIAS=E04, FORMAT=I09, TITLE='Average Household Size',
   DESC='Average Household Size',$
  FIELD=MEDHHI98, ALIAS=E05, FORMAT=I09, TITLE='Median Household Income',
   DESC='Median Household Income',$
  FIELD=AVGHHI98, ALIAS=E06, FORMAT=I09, TITLE='Average Household Income',
   DESC='Average Household Income',$
  FIELD=MALEPOP98,ALIAS=E07, FORMAT=I09, TITLE='Male Population',
   DESC='Male Population',$
  FIELD=FEMPOP98, ALIAS=E08, FORMAT=I09, TITLE='Female Population',
   DESC='Female Population',$
  FIELD=P15TO1998,ALIAS=E09, FORMAT=I09, TITLE='15 to 19',
   DESC='Population 15 to 19 years old',$
  FIELD=P20TO2998,ALIAS=E10, FORMAT=I09, TITLE='20 to 29',
   DESC='Population 20 to 29 years old',$
  FIELD=P30TO4998,ALIAS=E11, FORMAT=I09, TITLE='30 to 49',
   DESC='Population 30 to 49 years old',$
  FIELD=P50TO6498,ALIAS=E12, FORMAT=I09, TITLE='50 to 64',
   DESC='Population 50 to 64 years old',$
  FIELD=P65OVR98, ALIAS=E13, FORMAT=I09, TITLE='65 and over',
   DESC='Population 65 and over',$
```

GGDEMOG Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE GGDEMOG  ON 05/15/03 AT 12.26.05

         GGDEMOG
 01      S1
**************
*ST          **I
*HH          **
*AVGHHSZ98   **
*MEDHHI98    **
*            **
***************
 **************
```

GGORDER Master File

```
FILENAME=GGORDER, SUFFIX=FOC,$
 SEGNAME=ORDER01, SEGTYPE=S1,$
  FIELD=ORDER_NUMBER, ALIAS=ORDNO1,   FORMAT=I6,  TITLE='Order,Number',
   DESC='Order Identification Number',$
  FIELD=ORDER_DATE,   ALIAS=DATE,     FORMAT=MDY, TITLE='Order,Date',
   DESC='Date order was placed',$
  FIELD=STORE_CODE,   ALIAS=STCD,     FORMAT=A5,  TITLE='Store,Code',
   DESC='Store Identification Code (for order)',$
  FIELD=PRODUCT_CODE, ALIAS=PCD,      FORMAT=A4,  TITLE='Product,Code',
   DESC='Product Identification Code (for order)',$
  FIELD=QUANTITY,     ALIAS=ORDUNITS, FORMAT=I8,  TITLE='Ordered,Units',
   DESC='Quantity Ordered',$
SEGNAME=ORDER02, SEGTYPE=KU, PARENT=ORDER01, CRFILE=GGPRODS, CRKEY=PCD,
CRSEG=PRODS01  ,$
```

GGORDER Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE GGORDER  ON 05/15/03 AT 16.45.48

          GGORDER
  01      S1
 **************
 *ORDER_NUMBER**
 *ORDER_DATE  **
 *STORE_CODE  **
 *PRODUCT_CODE**
 *            **
 ***************
  **************
        I
        I
        I
        I ORDER02
  02    I KU
 ..............
 :PRODUCT_ID  :K
 :PRODUCT_DESC:
 :VENDOR_CODE :
 :VENDOR_NAME :
 :            :
 :............:
```

GGPRODS Master File

```
FILENAME=GGPRODS, SUFFIX=FOC
 SEGNAME=PRODS01, SEGTYPE=S1
  FIELD=PRODUCT_ID, ALIAS=PCD, FORMAT=A4, INDEX=I, TITLE='Product,Code',
   DESC='Product Identification Code',$
  FIELD=PRODUCT_DESCRIPTION, ALIAS=PRODUCT, FORMAT=A16, TITLE='Product',
   DESC='Product Name',$
  FIELD=VENDOR_CODE, ALIAS=VCD, FORMAT=A4, INDEX=I, TITLE='Vendor ID',
   DESC='Vendor Identification Code',$
  FIELD=VENDOR_NAME, ALIAS=VENDOR, FORMAT=A23, TITLE='Vendor Name',
   DESC='Vendor Name',$
  FIELD=PACKAGE_TYPE, ALIAS=PACK, FORMAT=A7, TITLE='Package',
   DESC='Packaging Style',$
  FIELD=SIZE, ALIAS=SZ, FORMAT=I2, TITLE='Size',
   DESC='Package Size',$
  FIELD=UNIT_PRICE, ALIAS=UNITPR, FORMAT=D7.2, TITLE='Unit,Price',
   DESC='Price for one unit',$
```

GGPRODS Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE GGPRODS  ON 05/15/03 AT 12.26.05

         GGPRODS
 01      S1
*************
*PRODUCT_ID  **I
*PRODUCT_DESC**I
*VENDOR_CODE **
*VENDOR_NAME **
*             **
**************
 *************
```

GGSALES Master File

```
FILENAME=GGSALES, SUFFIX=FOC
 SEGNAME=SALES01, SEGTYPE=S1
  FIELD=SEQ_NO, ALIAS=SEQ, FORMAT=I5, TITLE='Sequence#',
   DESC='Sequence number in database',$
  FIELD=CATEGORY, ALIAS=E02, FORMAT=A11, INDEX=I, TITLE='Category',
   DESC='Product category',$
  FIELD=PCD, ALIAS=E03, FORMAT=A04, INDEX=I, TITLE='Product ID',
   DESC='Product Identification code (for sale)',$
  FIELD=PRODUCT, ALIAS=E04, FORMAT=A16, TITLE='Product',
   DESC='Product name',$
  FIELD=REGION, ALIAS=E05, FORMAT=A11, INDEX=I, TITLE='Region',
   DESC='Region code',$
  FIELD=ST, ALIAS=E06, FORMAT=A02, INDEX=I, TITLE='State',
   DESC='State',$
  FIELD=CITY, ALIAS=E07, FORMAT=A20, TITLE='City',
   DESC='City',$
  FIELD=STCD, ALIAS=E08, FORMAT=A05, INDEX=I, TITLE='Store ID',
   DESC='Store identification code (for sale)',$
  FIELD=DATE, ALIAS=E09, FORMAT=I8YYMD, TITLE='Date',
   DESC='Date of sales report',$
  FIELD=UNITS, ALIAS=E10, FORMAT=I08, TITLE='Unit Sales',
   DESC='Number of units sold',$
  FIELD=DOLLARS, ALIAS=E11, FORMAT=I08, TITLE='Dollar Sales',
   DESC='Total dollar amount of reported sales',$
  FIELD=BUDUNITS, ALIAS=E12, FORMAT=I08, TITLE='Budget Units',
   DESC='Number of units budgeted',$
  FIELD=BUDDOLLARS, ALIAS=E13, FORMAT=I08, TITLE='Budget Dollars',
   DESC='Total sales quota in dollars',$
```

GGSALES Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE GGSALES  ON 05/15/03 AT 12.26.05

         GGSALES
 01      S1
**************
*SEQ_NO       **
*CATEGORY     **I
*PCD          **I
*PRODUCT      **I
*             **
***************
 **************
```

GGSTORES Master File

```
FILENAME=GGSTORES, SUFFIX=FOC
 SEGNAME=STORES01, SEGTYPE=S1
  FIELD=STORE_CODE, ALIAS=E02, FORMAT=A05, INDEX=I, TITLE='Store ID',
   DESC='Franchisee ID Code',$
  FIELD=STORE_NAME, ALIAS=E03, FORMAT=A23, TITLE='Store Name',
   DESC='Store Name',$
  FIELD=ADDRESS1, ALIAS=E04, FORMAT=A19, TITLE='Contact',
   DESC='Franchisee Owner',$
  FIELD=ADDRESS2, ALIAS=E05, FORMAT=A31, TITLE='Address',
   DESC='Street Address',$
  FIELD=CITY, ALIAS=E06, FORMAT=A22, TITLE='City',
   DESC='City',$
  FIELD=STATE, ALIAS=E07, FORMAT=A02, INDEX=I, TITLE='State',
   DESC='State',$
  FIELD=ZIP, ALIAS=E08, FORMAT=A06, TITLE='Zip Code',
   DESC='Postal Code',$
```

GGSTORES Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE GGSTORES ON 05/15/03 AT 12.26.05

         GGSTORES
 01      S1
**************
*STORE_CODE   **I
*STORE_NAME   **
*ADDRESS1     **
*ADDRESS2     **
*             **
***************
 **************
```

Century Corp Data Sources

Century Corp is a consumer electronics manufacturer that distributes products through retailers around the world. Century Corp has thousands of employees in plants, warehouses, and offices worldwide. Their mission is to provide quality products and services to their customers.

Century Corp is a group of data sources that contain financial, human resources, inventory, and order information. The last three data sources are designed to be used with chart of accounts data.

- CENTCOMP Master File contains location information for stores. It consists of one segment, COMPINFO.
- CENTFIN Master File contains financial information. It consists of one segment, ROOT_SEG.
- CENTHR Master File contains human resources information. It consists of one segment, EMPSEG.
- CENTINV Master File contains inventory information. It consists of one segment, INVINFO.
- CENTORD Master File contains order information. It consists of four segments, OINFO, STOSEG, PINFO, and INVSEG.
- CENTQA Master File contains problem information. It consists of three segments, PROD_SEG, INVSEG, and PROB_SEG.
- CENTGL Master File contains a chart of accounts hierarchy. The field GL_ACCOUNT_PARENT is the parent field in the hierarchy. The field GL_ACCOUNT is the hierarchy field. The field GL_ACCOUNT_CAPTION can be used as the descriptive caption for the hierarchy field.
- CENTSYSF Master File contains detail-level financial data. CENTSYSF uses a different account line system (SYS_ACCOUNT), which can be joined to the SYS_ACCOUNT field in CENTGL. Data uses "natural" signs (expenses are positive, revenue negative).
- CENTSTMT Master File contains detail-level financial data and a cross-reference to the CENTGL data source.

CENTCOMP Master File

```
FILE=CENTCOMP, SUFFIX=FOC, FDFC=19, FYRT=00
  SEGNAME=COMPINFO, SEGTYPE=S1, $
  FIELD=STORE_CODE, ALIAS=SNUM, FORMAT=A6, INDEX=I,
   TITLE='Store Id#:',
   DESCRIPTION='Store Id#', $
  FIELD=STORENAME, ALIAS=SNAME, FORMAT=A20,
   WITHIN=STATE,
   TITLE='Store,Name:',
   DESCRIPTION='Store Name', $
  FIELD=STATE, ALIAS=STATE, FORMAT=A2,
   WITHIN=PLANT,
   TITLE='State:',
   DESCRIPTION=State, $
  DEFINE REGION/A5=DECODE STATE ('AL' 'SOUTH' 'AK' 'WEST' 'AR' 'SOUTH'
  'AZ' 'WEST' 'CA' 'WEST' 'CO' 'WEST' 'CT' 'EAST'
  'DE' 'EAST' 'DC' 'EAST' 'FL' 'SOUTH' 'GA' 'SOUTH' 'HI' 'WEST'
  'ID' 'WEST' 'IL' 'NORTH' 'IN' 'NORTH' 'IA' 'NORTH'
  'KS' 'NORTH' 'KY' 'SOUTH' 'LA' 'SOUTH' 'ME' 'EAST' 'MD' 'EAST'
  'MA' 'EAST' 'MI' 'NORTH' 'MN' 'NORTH' 'MS' 'SOUTH' 'MT' 'WEST'
  'MO' 'SOUTH' 'NE' 'WEST'  'NV' 'WEST' 'NH' 'EAST' 'NJ' 'EAST'
  'NM' 'WEST' 'NY' 'EAST' 'NC' 'SOUTH' 'ND' 'NORTH' 'OH' 'NORTH'
  'OK' 'SOUTH' 'OR' 'WEST' 'PA' 'EAST' 'RI' 'EAST' 'SC' 'SOUTH'
  'SD' 'NORTH' 'TN' 'SOUTH' 'TX' 'SOUTH' 'UT' 'WEST' 'VT' 'EAST'
  'VA' 'SOUTH' 'WA' 'WEST' 'WV' 'SOUTH' 'WI' 'NORTH' 'WY' 'WEST'
  'NA' 'NORTH' 'ON' 'NORTH' ELSE ' ');,
   TITLE='Region:',
   DESCRIPTION=Region, $
```

CENTCOMP Structure Diagram

```
SECTION 01
       STRUCTURE OF FOCUS    FILE CENTCOMP ON 05/15/03 AT 10.20.49

          COMPINFO
  01      S1
 **************
 *STORE_CODE   **I
 *STORENAME    **
 *STATE        **
 *             **
 *             **
 ***************
  **************
```

CENTFIN Master File

```
FILE=CENTFIN, SUFFIX=FOC, FDFC=19, FYRT=00
  SEGNAME=ROOT_SEG, SEGTYPE=S4, $
  FIELD=YEAR, ALIAS=YEAR, FORMAT=YY,
   WITHIN='*Time Period', $
  FIELD=QUARTER, ALIAS=QTR, FORMAT=Q,
   WITHIN=YEAR,
   TITLE=Quarter,
   DESCRIPTION=Quarter, $
  FIELD=MONTH, ALIAS=MONTH, FORMAT=M,
   TITLE=Month,
   DESCRIPTION=Month, $
  FIELD=ITEM, ALIAS=ITEM, FORMAT=A20,
   TITLE=Item,
   DESCRIPTION=Item, $
  FIELD=VALUE, ALIAS=VALUE, FORMAT=D12.2,
   TITLE=Value,
   DESCRIPTION=Value, $
  DEFINE ITYPE/A12=IF EDIT(ITEM,'9$$$$$$$$$$$$$$$$$$$') EQ 'E'
   THEN 'Expense' ELSE IF EDIT(ITEM,'9$$$$$$$$$$$$$$$$$$$') EQ 'R'
   THEN 'Revenue' ELSE 'Asset';,
   TITLE=Type,
   DESCRIPTION='Type of Financial Line Item',$
  DEFINE MOTEXT/MT=MONTH;,$
```

CENTFIN Structure Diagram

```
SECTION 01
       STRUCTURE OF FOCUS    FILE CENTFIN  ON 05/15/03 AT 10.25.52

          ROOT_SEG
  01      S4
 **************
 *YEAR         **
 *QUARTER      **
 *MONTH        **
 *ITEM         **
 *             **
 ***************
  **************
```

CENTHR Master File

```
FILE=CENTHR, SUFFIX=FOC
  SEGNAME=EMPSEG, SEGTYPE=S1, $
  FIELD=ID_NUM, ALIAS=ID#, FORMAT=I9,
   TITLE='Employee,ID#',
   DESCRIPTION='Employee Identification Number', $
  FIELD=LNAME, ALIAS=LN, FORMAT=A14,
   TITLE='Last,Name',
   DESCRIPTION='Employee Last Name', $
  FIELD=FNAME, ALIAS=FN, FORMAT=A12,
   TITLE='First,Name',
   DESCRIPTION='Employee First Name', $
  FIELD=PLANT, ALIAS=PLT, FORMAT=A3,
   TITLE='Plant,Location',
   DESCRIPTION='Location of the manufacturing plant',
   WITHIN='*Location', $
  FIELD=START_DATE, ALIAS=SDATE, FORMAT=YYMD,
   TITLE='Starting,Date',
   DESCRIPTION='Date of employment',$
  FIELD=TERM_DATE, ALIAS=TERM_DATE, FORMAT=YYMD,
   TITLE='Termination,Date',
   DESCRIPTION='Termination Date', $
  FIELD=STATUS, ALIAS=STATUS, FORMAT=A10,
   TITLE='Current,Status',
   DESCRIPTION='Job Status', $
  FIELD=POSITION, ALIAS=JOB, FORMAT=A2,
   TITLE=Position,
   DESCRIPTION='Job Position',  $
  FIELD=PAYSCALE, ALIAS=PAYLEVEL, FORMAT=I2,
   TITLE='Pay,Level',
   DESCRIPTION='Pay Level',
   WITHIN='*Wages',$
  DEFINE POSITION_DESC/A17=IF POSITION EQ 'BM' THEN
   'Plant Manager' ELSE
   IF POSITION EQ 'MR' THEN 'Line Worker' ELSE
   IF POSITION EQ 'TM' THEN 'Line Manager' ELSE
   'Technician';
   TITLE='Position,Description',
   DESCRIPTION='Position Description',
   WITHIN='PLANT',$
  DEFINE BYEAR/YY=START_DATE;
   TITLE='Beginning,Year',
   DESCRIPTION='Beginning Year',
   WITHIN='*Starting Time Period',$
```

```
DEFINE BQUARTER/Q=START_DATE;
 TITLE='Beginning,Quarter',
 DESCRIPTION='Beginning Quarter',
 WITHIN='BYEAR',
DEFINE BMONTH/M=START_DATE;
 TITLE='Beginning,Month',
 DESCRIPTION='Beginning Month',
 WITHIN='BQUARTER',$
DEFINE EYEAR/YY=TERM_DATE;
 TITLE='Ending,Year',
 DESCRIPTION='Ending Year',
 WITHIN='*Termination Time Period',$
DEFINE EQUARTER/Q=TERM_DATE;
 TITLE='Ending,Quarter',
 DESCRIPTION='Ending Quarter',
 WITHIN='EYEAR',$
DEFINE EMONTH/M=TERM_DATE;
 TITLE='Ending,Month',
 DESCRIPTION='Ending Month',
 WITHIN='EQUARTER',$
DEFINE RESIGN_COUNT/I3=IF STATUS EQ 'RESIGNED' THEN 1
 ELSE 0;
 TITLE='Resigned,Count',
 DESCRIPTION='Resigned Count',$
DEFINE FIRE_COUNT/I3=IF STATUS EQ 'TERMINAT' THEN 1
 ELSE 0;
 TITLE='Terminated,Count',
 DESCRIPTION='Terminated Count',$
DEFINE DECLINE_COUNT/I3=IF STATUS EQ 'DECLINED' THEN 1
 ELSE 0;
 TITLE='Declined,Count',
 DESCRIPTION='Declined Count',$
DEFINE EMP_COUNT/I3=IF STATUS EQ 'EMPLOYED' THEN 1
 ELSE 0;
 TITLE='Employed,Count',
 DESCRIPTION='Employed Count',$
DEFINE PEND_COUNT/I3=IF STATUS EQ 'PENDING' THEN 1
 ELSE 0;
 TITLE='Pending,Count',
 DESCRIPTION='Pending Count',$
DEFINE REJECT_COUNT/I3=IF STATUS EQ 'REJECTED' THEN 1
 ELSE 0;
 TITLE='Rejected,Count',
 DESCRIPTION='Rejected Count',$
DEFINE FULLNAME/A28=LNAME||', '|FNAME;
 TITLE='Full Name',
 DESCRIPTION='Full Name: Last, First', WITHIN='POSITION_DESC',$
```

```
 DEFINE SALARY/D12.2=IF BMONTH LT 4 THEN PAYLEVEL * 12321
  ELSE IF BMONTH GE 4 AND BMONTH LT 8 THEN PAYLEVEL * 13827
  ELSE PAYLEVEL * 14400;,
  TITLE='Salary',
  DESCRIPTION='Salary',$
 DEFINE PLANTLNG/A11=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
  LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
  ELSE 'n/a');$
```

CENTHR Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE CENTHR   ON 05/15/03 AT 10.40.34

          EMPSEG
  01      S1
 **************
 *ID_NUM      **
 *LNAME       **
 *FNAME       **
 *PLANT       **
 *            **
 ***************
  **************
```

CENTINV Master File

```
FILE=CENTINV, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=INVINFO, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number:',
   DESCRIPTION='Product Number', $
  FIELD=PRODNAME, ALIAS=PNAME, FORMAT=A30,
   WITHIN=PRODCAT,
   TITLE='Product,Name:',
   DESCRIPTION='Product Name', $
  FIELD=QTY_IN_STOCK, ALIAS=QIS, FORMAT=I7,
   TITLE='Quantity,In Stock:',
   DESCRIPTION='Quantity In Stock', $
  FIELD=PRICE, ALIAS=RETAIL, FORMAT=D10.2,
   TITLE='Price:',
   DESCRIPTION=Price, $
  FIELD=COST, ALIAS=OUR_COST, FORMAT=D10.2,
   TITLE='Our,Cost:',
   DESCRIPTION='Our Cost:', $
  DEFINE PRODCAT/A22 = IF PRODNAME CONTAINS 'LCD'
   THEN 'VCRs' ELSE IF PRODNAME
   CONTAINS 'DVD' THEN 'DVD' ELSE IF PRODNAME CONTAINS 'Camcor'
   THEN 'Camcorders'
   ELSE IF PRODNAME CONTAINS 'Camera' THEN 'Cameras' ELSE IF PRODNAME
   CONTAINS 'CD' THEN 'CD Players'
   ELSE IF PRODNAME CONTAINS 'Tape' THEN 'Digital Tape Recorders'
   ELSE IF PRODNAME CONTAINS 'Combo' THEN 'Combo Players'
   ELSE 'PDA Devices'; WITHIN=PRODTYPE, TITLE='Product Category:' ,$
  DEFINE PRODTYPE/A19 = IF PRODNAME CONTAINS 'Digital' OR 'DVD' OR 'QX'
   THEN 'Digital' ELSE 'Analog';,WITHIN='*Product Dimension',
   TITLE='Product Type:',$
```

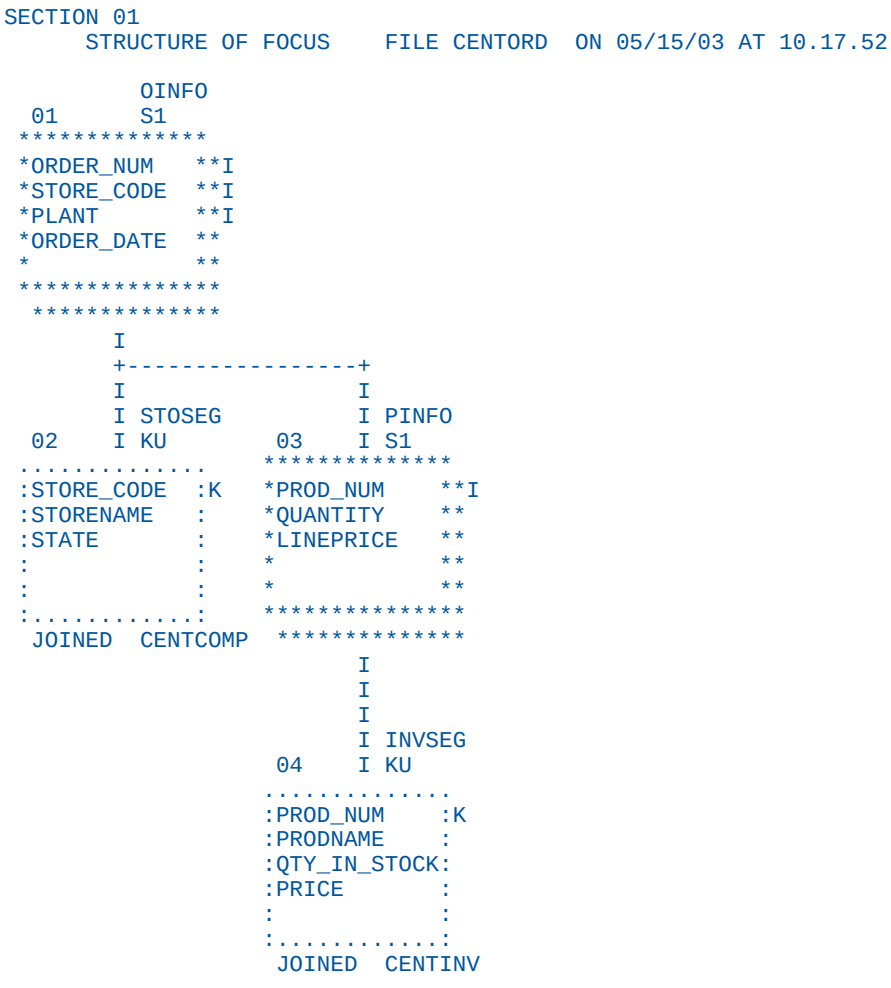
CENTINV Structure Diagram

```
SECTION 01
       STRUCTURE OF FOCUS    FILE CENTINV  ON 05/15/03 AT 10.43.35

          INVINFO
  01      S1
 **************
 *PROD_NUM    **I
 *PRODNAME    **
 *QTY_IN_STOCK**
 *PRICE       **
 *            **
 ***************
  ***************
```

CENTORD Master File

```
FILE=CENTORD, SUFFIX=FOC
 SEGNAME=OINFO, SEGTYPE=S1, $
  FIELD=ORDER_NUM, ALIAS=ONUM, FORMAT=A5, INDEX=I,
   TITLE='Order,Number:',
   DESCRIPTION='Order Number', $
  FIELD=ORDER_DATE, ALIAS=ODATE, FORMAT=YYMD,
   TITLE='Date,Of Order:',
   DESCRIPTION='Date Of Order', $
  FIELD=STORE_CODE, ALIAS=SNUM, FORMAT=A6, INDEX=I,
   TITLE='Company ID#:',
   DESCRIPTION='Company ID#', $
  FIELD=PLANT, ALIAS=PLNT, FORMAT=A3, INDEX=I,
   TITLE='Manufacturing,Plant',
   DESCRIPTION='Location Of Manufacturing Plant',
   WITHIN='*Location',$
  DEFINE YEAR/YY=ORDER_DATE;,
   WITHIN='*Time Period',$
  DEFINE QUARTER/Q=ORDER_DATE;,
   WITHIN='YEAR',$
  DEFINE MONTH/M=ORDER_DATE;,
   WITHIN='QUARTER',$
 SEGNAME=PINFO, SEGTYPE=S1, PARENT=OINFO, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4,INDEX=I,
   TITLE='Product,Number#:',
   DESCRIPTION='Product Number#', $
  FIELD=QUANTITY, ALIAS=QTY, FORMAT=I8C,
   TITLE='Quantity:',
   DESCRIPTION=Quantity, $
  FIELD=LINEPRICE, ALIAS=LINETOTAL, FORMAT=D12.2MC,
   TITLE='Line,Total',
   DESCRIPTION='Line Total', $
  DEFINE LINE_COGS/D12.2=QUANTITY*COST;,
   TITLE='Line,Cost Of,Goods Sold',
   DESCRIPTION='Line cost of goods sold', $
  DEFINE PLANTLNG/A11=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
   LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
   ELSE 'n/a');
 SEGNAME=INVSEG, SEGTYPE=DKU, PARENT=PINFO, CRFILE=CENTINV,
  CRKEY=PROD_NUM,   CRSEG=INVINFO,$
 SEGNAME=STOSEG, SEGTYPE=DKU, PARENT=OINFO, CRFILE=CENTCOMP,
  CRKEY=STORE_CODE, CRSEG=COMPINFO,$
```

CENTORD Structure Diagram



CENTQA Master File

```
FILE=CENTQA, SUFFIX=FOC, FDFC=19, FYRT=00
 SEGNAME=PROD_SEG, SEGTYPE=S1, $
  FIELD=PROD_NUM, ALIAS=PNUM, FORMAT=A4, INDEX=I,
   TITLE='Product,Number',
   DESCRIPTION='Product Number', $
 SEGNAME=PROB_SEG, PARENT=PROD_SEG, SEGTYPE=S1, $
  FIELD=PROBNUM, ALIAS=PROBNO, FORMAT=I5,
   TITLE='Problem,Number',
   DESCRIPTION='Problem Number',
   WITHIN=PLANT,$
  FIELD=PLANT, ALIAS=PLT, FORMAT=A3, INDEX=I,
   TITLE=Plant,
   DESCRIPTION=Plant,
   WITHIN=PROBLEM_LOCATION,$
  FIELD=PROBLEM_DATE, ALIAS=PDATE, FORMAT=YYMD,
   TITLE='Date,Problem,Reported',
   DESCRIPTION='Date Problem Was Reported', $
  FIELD=PROBLEM_CATEGORY, ALIAS=PROBCAT, FORMAT=A20, $
   TITLE='Problem,Category',
   DESCRIPTION='Problem Category',
   WITHIN=*Problem,$
  FIELD=PROBLEM_LOCATION, ALIAS=PROBLOC, FORMAT=A10,
   TITLE='Location,Problem,Occurred',
   DESCRIPTION='Location Where Problem Occurred',
   WITHIN=PROBLEM_CATEGORY,$
  DEFINE PROB_YEAR/YY=PROBLEM_DATE;,
   TITLE='Year,Problem,Occurred',
   DESCRIPTION='Year Problem Occurred',
   WITHIN=*Time Period,$
  DEFINE PROB_QUARTER/Q=PROBLEM_DATE;
   TITLE='Quarter,Problem,Occurred',
   DESCRIPTION='Quarter Problem Occurred',
   WITHIN=PROB_YEAR,$
  DEFINE PROB_MONTH/M=PROBLEM_DATE;
   TITLE='Month,Problem,Occurred',
   DESCRIPTION='Month Problem Occurred',
   WITHIN=PROB_QUARTER,$
  DEFINE PROBLEM_OCCUR/I5 WITH PROBNUM=1;,
   TITLE='Problem,Occurrence'
   DESCRIPTION='# of times a problem occurs',$
  DEFINE PLANTLNG/A11=DECODE PLANT (BOS 'Boston' DAL 'Dallas'
   LA 'Los Angeles' ORL 'Orlando' SEA 'Seattle' STL 'St Louis'
   ELSE 'n/a');$
 SEGNAME=INVSEG, SEGTYPE=DKU, PARENT=PROD_SEG, CRFILE=CENTINV,
  CRKEY=PROD_NUM, CRSEG=INVINFO,$
```

CENTQA Structure Diagram

```
SECTION 01
      STRUCTURE OF FOCUS    FILE CENTQA   ON 05/15/03 AT 10.46.43

         PROD_SEG
 01      S1
 **************
 *PROD_NUM    **I
 *            **
 *            **
 *            **
 *            **
 ***************
  **************
       I
       +-----------------+
       I                 I
       I INVSEG          I PROB_SEG
 02    I KU        03    I S1
 ..............    **************
 :PROD_NUM    :K   *PROBNUM     **
 :PRODNAME    :    *PLANT       **I
 :QTY_IN_STOCK:    *PROBLEM_DATE**
 :PRICE       :    *PROBLEM_CAT>**
 :            :    *             **
 :............:    **************
  JOINED  CENTINV   **************
```

CENTGL Master File

```
FILE=CENTGL ,SUFFIX=FOC
 SEGNAME=ACCOUNTS, SEGTYPE=S1
  FIELDNAME=GL_ACCOUNT, ALIAS=GLACCT, FORMAT=A7,
   TITLE='Ledger,Account', FIELDTYPE=I, $
  FIELDNAME=GL_ACCOUNT_PARENT, ALIAS=GLPAR, FORMAT=A7,
   TITLE=Parent,
   PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
  FIELDNAME=GL_ACCOUNT_TYPE, ALIAS=GLTYPE, FORMAT=A1,
   TITLE=Type,$
  FIELDNAME=GL_ROLLUP_OP, ALIAS=GLROLL, FORMAT=A1,
   TITLE=Op, $
  FIELDNAME=GL_ACCOUNT_LEVEL, ALIAS=GLLEVEL, FORMAT=I3,
   TITLE=Lev, $
  FIELDNAME=GL_ACCOUNT_CAPTION, ALIAS=GLCAP, FORMAT=A30,
   TITLE=Caption,
   PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
  FIELDNAME=SYS_ACCOUNT, ALIAS=ALINE, FORMAT=A6,
   TITLE='System,Account,Line', MISSING=ON, $
```

CENTGL Structure Diagram

```
SECTION 01
       STRUCTURE OF FOCUS    FILE CENTGL   ON 05/15/03 AT 15.18.48

          ACCOUNTS
  01      S1
 **************
 *GL_ACCOUNT  **I
 *GL_ACCOUNT_>**
 *GL_ACCOUNT_>**
 *GL_ROLLUP_OP**
 *            **
 ***************
  **************
```

CENTSYSF Master File

```
FILE=CENTSYSF ,SUFFIX=FOC
 SEGNAME=RAWDATA ,SEGTYPE=S2
  FIELDNAME = SYS_ACCOUNT ,   ,A6     , FIELDTYPE=I,
   TITLE='System,Account,Line', $
  FIELDNAME = PERIOD      ,   ,YYM    , FIELDTYPE=I,$
  FIELDNAME = NAT_AMOUNT  ,   ,D10.0 , TITLE='Month,Actual', $
  FIELDNAME = NAT_BUDGET  ,   ,D10.0 , TITLE='Month,Budget', $
  FIELDNAME = NAT_YTDAMT  ,   ,D12.0 , TITLE='YTD,Actual',   $
  FIELDNAME = NAT_YTDBUD  ,   ,D12.0 , TITLE='YTD,Budget',   $
```

CENTSYSF Structure Diagram

```
SECTION 01
       STRUCTURE OF FOCUS    FILE CENTSYSF   ON 05/15/03 AT 15.19.27

          RAWDATA
  01      S2
 **************
 *SYS_ACCOUNT **I
 *PERIOD      **I
 *NAT_AMOUNT  **
 *NAT_BUDGET  **
 *            **
 ***************
  **************
```

CENTSTMT Master File

```
FILE=CENTSTMT, SUFFIX=FOC
 SEGNAME=ACCOUNTS, SEGTYPE=S1
  FIELD=GL_ACCOUNT, ALIAS=GLACCT,  FORMAT=A7,
   TITLE='Ledger,Account', FIELDTYPE=I, $
  FIELD=GL_ACCOUNT_PARENT, ALIAS=GLPAR, FORMAT=A7,
   TITLE=Parent,
   PROPERTY=PARENT_OF, REFERENCE=GL_ACCOUNT, $
  FIELD=GL_ACCOUNT_TYPE, ALIAS=GLTYPE, FORMAT=A1,
   TITLE=Type,$
  FIELD=GL_ROLLUP_OP, ALIAS=GLROLL, FORMAT=A1,
   TITLE=Op, $
  FIELD=GL_ACCOUNT_LEVEL, ALIAS=GLLEVEL, FORMAT=I3,
   TITLE=Lev, $
  FIELD=GL_ACCOUNT_CAPTION, ALIAS=GLCAP, FORMAT=A30,
   TITLE=Caption,
   PROPERTY=CAPTION, REFERENCE=GL_ACCOUNT, $
 SEGNAME=CONSOL, SEGTYPE=S1, PARENT=ACCOUNTS, $
  FIELD=PERIOD, ALIAS=MONTH, FORMAT=YYM, $
  FIELD=ACTUAL_AMT, ALIAS=AA, FORMAT=D10.0, MISSING=ON,
   TITLE='Actual', $
  FIELD=BUDGET_AMT, ALIAS=BA, FORMAT=D10.0, MISSING=ON,
   TITLE='Budget', $
  FIELD=ACTUAL_YTD, ALIAS=AYTD, FORMAT=D12.0, MISSING=ON,
   TITLE='YTD,Actual', $
  FIELD=BUDGET_YTD, ALIAS=BYTD, FORMAT=D12.0, MISSING=ON,
   TITLE='YTD,Budget', $
```

CENTSTMT Structure Diagram

```
SECTION 01
     STRUCTURE OF FOCUS    FILE CENTSTMT ON 05/15/03 AT 14.45.44

          ACCOUNTS
 01       S1
 **************
 *GL_ACCOUNT  **I
 *GL_ACCOUNT_>**
 *GL_ACCOUNT_>**
 *GL_ROLLUP_OP**
 *            **
 ***************
  **************
        I
        I
        I
        I CONSOL
 02     I S1
 **************
 *PERIOD      **
 *ACTUAL_AMT  **
 *BUDGET_AMT  **
 *ACTUAL_YTD  **
 *            **
 ***************
  **************
```

Appendix B

Error Messages

To see the text or explanation for any error message, you can display it online in your FOCUS session or find it in a standard FOCUS ERRORS file. All of the FOCUS error messages are stored in eight system ERRORS files.

- For z/OS, the ddname is ERRORS.

In this appendix:

- Accessing Error Files
- Displaying Messages

Accessing Error Files

For z/OS, the error files are the following members in the ERRORS PDS:

- FOT004
- FOG004
- FOM004
- FOS004
- FOA004
- FSQLXLT
- FOCSTY
- FOB004

Displaying Messages

To display the text and explanation for any message, issue the following query command at the FOCUS command level

```
? n
```

where:

n

Is the message number.

The message number and text appear, along with a detailed explanation of the message (if one exists). For example, issuing the following command

```
? 210
```

displays the following:

```
(FOC210)    THE DATA VALUE HAS A FORMAT ERROR:
An alphabetic character has been found where all numerical digits are
required.
```

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, FOCUS, iWay, Omni-Gen, Omni-HealthData, and WebFOCUS are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (https://www.tibco.com/patents) for details.

Copyright © 2021. TIBCO Software Inc. All Rights Reserved.

Index

-? &\[string] command *245*

-? &\[variablename] commands *204*, *245*

-? command *308*

-? SET SETCOMMAND &myvar *204*

-* command *202*, *209*, *308*

-CLOSE command *202*, *262*, *308*

-CLOSE ddname command *202*

-CMS command *307*

-CMS RUN command *294*

-CRTCLEAR command *202*, *309*

-CRTFORM command *203*, *230*, *247*, *271*, *309*

-DEFAULT command *203*, *247*, *251*, *310*

-EXIT command *203*, *213*, *215*, *311*

-GOTO command *203*, *217–219*, *311*

-HTMLFORM command *203*, *247*

-IF ... GOTO command *219–222*

-IF command *203*, *217*, *219*, *220*, *311*

-IF tests *222*, *276*

  compound *222*

  operators and functions *221*

-INCLUDE command *203*, *226–228*, *312*

  nesting procedures *228*

-label command *203*, *313*

-MVS command *307*

-MVS RUN command *203*, *294*, *313*

-PASS command *203*, *211*, *212*, *314*

-PROMPT command *203*, *247*, *270*, *314*

-QUIT command *203*, *213*, *216*, *315*

-QUIT FOCUS command *213*, *216*

-READ command *203*, *247*, *256*, *257*, *259*, *316*

-READFILE command *203*, *261*, *317*

-REMOTE command *317*

-REPEAT command *203*, *222–224*, *317*

-RUN command *203*, *213*, *214*, *318*

-SET command *203*, *217*, *222*, *225*, *247*, *251*, *252*, *254*, *289*, *293*, *294*, *319*

-SET IN FILE phrase *251*

-TSO command *307*, *319*

-TSO RUN command *203*, *294*, *320*

-TYPE command *204*, *210*, *211*, *320*

-WINDOW command *204*, *230*, *247*, *271*, *321*, *450*

-WRITE command *204*, *256*, *258*, *259*, *322*

? && command *354*

? COMBINE command *330*

? DEFINE command *330*, *331*

? FDT command *333*, *334*

? FILE command *336*, *337*

? FUNCTION command *338*

? HOLD command *339*

? JOIN command *339*

? LET command *341*, *379*, *380*

? LOAD command *341*, *342*, *388*

? n command *342*

? NLS command *340*, *341*

? PFKEY command *343*

? RELEASE command *343*, *344*

? SET command *344*

? SET EUROFILE command *332*, *443*
? SET GRAPH command *346*
? STAT command *348*
? STYLE command *351*
? SU command *353*
? USE command *353*, *354*
?F command *332*
?FF command *335*
.EVAL operator *280*
.EVAL suffix *276*
.EXIST suffix *276*
.LENGTH suffix *276*
.TYPE suffix *276*
&ACCEPTS variable *242*
&BASEIO variable *242*
&CHNGD variable *242*
&CURSOR variable *245*
&DATE variable *234*, *240*
&DATEfmt variable *428*
&DELTD variable *243*
&DMY variable *235*
&DMYY variable *235*, *241*, *428*
&DUPLS variable *243*
&ECHO variable *245*, *300*, *301*
&FOCCODEPAGE variable *236*
&FOCCPU variable *236*
&FOCDISORG variable *243*
&FOCERRNUM variable *243*
&FOCEXTTRM variable *236*
&FOCFIELDNAME variable *236*
&FOCFOCEXEC variable *236*, *240*
&FOCINCLUDE variable *237*
&FOCMODE variable *237*
&FOCNEXTPAGE variable *237*
&FOCPRINT variable *237*
&FOCPUTLVL variable *237*
&FOCQUALCHAR variable *237*
&FOCREL variable *237*
&FOCSBORDER variable *238*
&FOCTRMSD variable *238*
&FOCTRMSW variable *238*
&FOCTRMTYP variable *238*
&FOCUSER variable *238*
&FORMAT variable *243*
&HIPERFOCUS variable *238*
&INPUT variable *243*
&INVALID variable *243*
&IORETURN variable *238*, *300*
&LINES variable *243*
&MDY variable *238*
&MDYY variable *238*, *241*, *428*
&myvar *204*
&NOMATCH variable *243*
&PFKEY variable *245*, *475*
&QUIT variable *217*, *245*
&READS variable *244*
&RECORDS variable *244*
&REJECTS variable *244*
&RETCODE variable *239*, *300*, *306*
&SETFILE variable *239*

&STACK variable *245*, *300*, *305*
&TOD variable *240*
&TRANS variable *244*
&WINDOWNAME variable *245*, *474*
&WINDOWVALUE variable *245*, *474*
&YMD variable *240*
&YYMD variable *240*, *241*, *428*

A

Absolute File Integrity *135*
ACCBLN parameter *29*, *54*
ACCEPT attribute *102*
Access Files *387*
    loading *387*
access to data *211*
accessing APP directories on Windows *160*
accessing data sources *392*
accessing FOCUS data sources *392*
ACROSSLINE parameter *42*, *54*
ACROSSPRT parameter *42*, *54*
ACROSSTITLE parameter *42*, *55*
ACRSVRBTITL parameter *56*
activating currency data source *438*
activating filters *96*
AGGR[RATIO] parameter *28*
aggregation *92*
ALL parameter *38*, *57*
allocating data files *190*
allocating temporary files *196*
ALLOWCVTERR parameter *38*, *57*
ALTBACKPERLINE parameter *43*
amper variables *204*, *230*, *275*, *289–291*
    quote-delimited *285*
    return values *472*
APP ? METALOCATION_SAME command *161*
APP APPENDPATH command *158*
APP application management commands *154*
APP commands *153*
    APP ? METALOCATION_SAME *161*
    APP APPENDPATH *156*, *158*
    APP COPY *164*
    APP COPYF *165*
    APP CREATE *154*
    APP DELETE *154*, *168*
    APP DELETEF *168*
    APP FI *179*
    APP HELP *187*
    APP HOLD *177*
    APP LIST *181*
    APP MAP *159*
    APP MOVE *166*
    APP MOVEF *167*
    APP PATH *156*
    APP PREPENDPATH *156*, *157*
    APP PROPERTY CODEPAGE *169*
    APP QUERY *182*
    APP RENAME *154*, *170*
    APP RENAMEF *170*
    APP SET METALOCATION_SAME *161*
    APP SHOWPATH *161*

APP commands *153*
FI *175*
file types *171*
HOLD *175*
HOLDDATA *175*, *178*
HOLDMETA *175*, *178*
APP COPY command *164*
APP COPYF command *165*, *171*
APP DELETE command *168*
APP DELETEF command *168*, *171*
APP FI command *175*, *179*
APP HELP command *155*, *187*
APP HOLD command *175*, *177*
APP HOLDDATA command *175*, *178*
APP HOLDMETA command *175*, *178*
APP LIST command *155*, *181*
APP MAP command *159*
APP MOVE command *166*
APP MOVEF command *167*, *171*
APP PATH command *156*
configuring *156*
search paths *156*
APP PREPENDPATH command *157*
APP PROPERTY CODEPAGE command *169*
APP QUERY command *155*, *182*
APP RENAME command *170*
APP RENAMEF command *170*, *171*
APP reporting and help commands *155*
APP SET METALOCATION_SAME command *161*
APP SHOWPATH command *161*
appending applications to search path *158*
application (APP) *152*
physical location of *152*
application components *151*, *152*
listing *182*
application directories *159*, *162*
creating *162*
mapping manually *159*
application paths *156*
APP PATH command *156*
configuring *156*
application repositories *151*, *152*
applications *168*, *499*
controlling memory *36*
deleting *168*
optimizing *36*
running *499*
approot parameter *152*
AS phrase *59*
ASNAMES parameter *29*, *38*, *59*
AUTOFIT parameter *43*, *60*
AUTOINDEX parameter *36*, *60*
AUTOPATH parameter *36*, *61*
AUTOSTRATEGY parameter *36*, *61*
AUTOTABLEF parameter *38*, *62*

B

base dates *79*
BASEURL parameter *43*, *62*
BINS parameter *37*, *62*

blank field values *54*
blank lines, suppressing *86*
BLANKINDENT parameter *43*, *63*
BLKCALC parameter *29*
BOTTOMMARGIN parameter *43*, *64*
branching *203*, *217*
    conditional *219–222*
    unconditional *218*, *219*
buffering techniques *392*
BUSDAYS parameter *33*, *38*, *64*
BYDISPLAY parameter *43*, *64*
BYPANEL parameter *43*, *65*
BYSCROLL parameter *43*

c

CACHE parameter *37*, *66*, *350*
calculated values *422*
    MODIFY requests and *424*, *425*
    sliding window *422*, *423*
calculations *28*
    controlling with SET parameters *28*
canceling a procedure *216*
CAR data source *544*, *545*
CARTESIAN parameter *38*, *66*
Cascading Style Sheets (CSS) *72*, *105*
CDN (Continental Decimal Notation) *443–445*
    punctuating large numbers *443*, *444*
CDN parameter *28*, *38*, *67*
CENT-ZERO parameter *38*, *43*, *68*
CENTFIN data source *570*
CENTHR data source *570*
CENTINV data source *570*
CENTORD data source *570*
CENTQA data source *570*
Century Corp data sources *570*
CHECK FILE command *405*
CNOTATION parameter *68*
codepage, application *169*
collapsing PRINT with ACROSS *54*
COLLATION parameter *38*, *69*
column notation *68*
column spacing *138*
columns, reporting on *358*
COLUMNSCROLL parameter *43*
COMASTER Master File *558*
combining fields *141*
comma-delimited files *125*
command statistics *348*
    ? STAT *348*
commands *23*, *401*, *405*
    ? LET *379*
    changing with variables *295*
    CHECK FILE *405*
    COMPUTE *422*
    DEFINE *401*, *416*
    Dialogue Manager *202*, *307*
    EXEC *213*, *214*
    executing *213*
    LET ECHO *380*
    LOAD *385*

commands *23*, *401*, *405*
nesting *228*
ON GRAPH SET *27*
operating system *307*
QUIT *217*
RUN *214*
stacked *205*
testing results *300*, *306*
UNLOAD *385*
WINDOW COMPILE *532*
WINDOW PAINT *500*
comments *209*, *210*
including in procedures *202*, *210*
COMPMISS parameter *70*
compound -IF tests *222*
COMPOUND parameter *43*, *70*
COMPUTE command *422*
sliding window *422*
COMPUTE parameter *28*, *37*, *38*, *71*
concatenating variables *283*
conditional branching *219*, *220*
-IF ... GOTO command *219–221*
compound -IF tests *222*
IF ...GOTO command *276*
screening values *276*
configuration files *152*
configuring application paths *156*
Continental Decimal Notation (CDN) *443–445*
punctuating large numbers *443*, *444*
controlling calculations *28*
controlling memory *36*
conversions *428*
DATE NEW option *428*
dates *428*
converting currencies *432*, *439*, *441*, *442*
currency data source *432*
error messages *441*
converting currency *91*
COUNT fields *72*
COUNTWIDTH parameter *29*, *72*
COURSE data source *552*, *553*
COURSES data source *549*
creating a currency data source *434*
creating data files *190*
creating procedure files *188*
creating procedures *208*
rules *208*
startup profiles *212*
with comments *209*, *210*, *212*
cross-century dates *397*, *401*, *427*
MODIFY requests and *401*
CRTFORMs *204*
CSS (Cascading Style Sheets) *72*
CSSURL parameter *44*, *72*
currencies *431*
converting *432*
CURRENCY attribute *436*, *437*
currency codes *434*
currency conversion function *441*
currency conversions *91*, *432*

currency data source *332*, *432*
activating *438*
creating *433*, *434*, *436*
CURRENCY attribute *436*, *437*
currency codes *433*, *434*, *436*
querying *443*
currency symbols *445*
extended currency symbols *447*
CURRENCY_DISPLAY parameter *73*
CURRENCY_ISO_CODE parameter *73*
CURRENCY_PRINT_ISO parameter *74*
currency-denominated fields *437*
currency-denominated values *436*
CURRSYMB parameter *74*
CURSYM_D parameter *75*
CURSYM_E parameter *75*
CURSYM_F parameter *76*
CURSYM_G parameter *76*
CURSYM_L parameter *76*
CURSYM_Y parameter *77*

D

data *29*
processing *29*
storing *29*
date formats *401*
date functions *79*
DATE_ORDER parameter *77*
DATE_SEPARATOR parameter *78*
date-time parameters *80*
DATEDISPLAY parameter *39*, *79*
DATEFNS parameter *33*, *39*, *79*
DATEFORMAT parameter *29*, *80*
dates *33*, *255*, *398*
converting *428*
default display format *427*
display options *428*
displaying *241*
functions and subroutines *428*
setting *255*
system *428*
time stamp *429*
validating *402*
DATETIME parameter *33*, *39*, *80*
DB_INFILE parameter *39*, *81*
DBA security rules *128*
DBACSENSITIV parameter *81*
DBAJOIN parameter *82*
DBASOURCE parameter *48*, *82*
deactivating filters *96*
debugging procedures *300*
DECODE function *289*, *290*
default century *428*
default file names *95*
default format for HOLD files *102*
default variable values *251*, *310*
-DEFAULT command and *251*, *310*
DEFCENT parameter *33*, *39*, *83*, *398*, *399*, *401*, *402*
COMPUTE command *422*, *424*

DEFCENT parameter *33*, *39*, *83*, *398*, *399*, *401*, *402*
    DEFINE command *416*
    MODIFY requests and *401*
    querying *408*
DEFECHO parameter *39*, *84*, *301*
DEFINE command *401*
    sliding window *401*, *416*
DEFINE FUNCTIONs, reporting on *359*
defined functions *338*
DEFINES parameter *29*, *37*, *84*
dense ranking *132*
designating a network drive on Windows *160*
Dialogue Manager *201*
    calling procedures *226*
    commands *202*, *276*, *307*
    comments *209*
    debugging procedures *300*
    looping procedures *222*
    messages *210*
    operating system commands *307*
    passwords *211*
    processing *205*, *206*
    reading files *256*
    stacked commands *205*, *215*
    uses *201*
    writing files *256*
direct prompting *247*, *270*, *314*
DIRECTHOLD parameter *85*
DISPLAY parameter *48*
displaying base dates *79*
displaying command statistics *348*
displaying currency data source *332*
displaying data sources with USE *353*
displaying dates *33*
displaying defined functions *338*
displaying Dialogue Manager values *354*
displaying fields *330*
    ? COMBINE command *330*
    ? DEFINE command *330*
    ?F command *332*
displaying graph parameters *346*
displaying grid for object placement evaluation *112*
displaying information about loaded files *341*
displaying leading zeros *68*, *112*
displaying LET substitutions *341*
displaying parameter settings *344*
displaying PF key assignments *343*
displaying product release number *343*
DISPLAYROUND parameter *45*
DMH_LOOPLIM parameter *85*
DMH_STACKLIM parameter *85*
DMPRECISION parameter *28*, *86*
DMPRECSION parameter *254*
    rounding *254*
drilling down remotely *98*
DROPBLNKLINE parameter *86*
DTSTRICT parameter *30*, *87*
DUPLICATECOL parameter *45*

DYNAM command *191*
DYNAM SET LONGSNYM *198*
dynamic window *400*, *405*, *406*

E

ECHO variable *84*, *235*
EDAGET *263*
EDAPUT *265*
edaserve.cfg configuration file *152*
EDIT function *290*, *291*
EDUCFILE data source *540*, *541*
EMBEDDABLE parameter *88*
embedding images in .htm file *105*
EMPDATA data source *549*–*551*
EMPLOYEE data source *535*, *537*, *538*
EMPTYREPORT parameter *39*, *88*
encoding HTML data *106*
Entry Menu *500*
EQTEST parameter *30*, *89*
error files *585*
error files, reporting on *360*
error messages *90*, *342*, *585*
    ? n command *342*
    displaying explanations *342*
ERROROUT parameter *39*, *90*
ESTRECORDS parameter *37*, *39*, *90*
euro currency *431*
    converting *432*, *439*, *441*, *442*
EUROFILE parameter *30*, *91*, *438*
    error messages *438*
EUROFILE parameter *30*, *91*, *438*
    restrictions *438*
evaluating variables *279*, *280*
EX command *213*
Excel requests *91*
EXCELSRVURL parameter *39*
EXEC command *213*, *214*, *228*, *229*, *247*, *267*, *268*
    calling procedures *228*
execution windows *458*
EXIST operator *278*
exiting from FOCUS *216*
EXITRC variable *235*
EXL2K output *137*
    SHOWBLANKS *137*
EXL2KLANG parameter *91*
EXL2KTXTDATE parameter *33*, *92*
EXTAGGR parameter *39*, *92*
extended currency symbols *445*, *447*
    formats *447*
EXTENDNUM parameter *93*
external Cascading Style Sheets *72*
external files *256*
    closing *262*
    maximum record length *115*
    reading from *256*, *257*
    writing to *256*
external sorting *94*
external sorts *92*, *93*, *142*
EXTHOLD parameter *40*, *93*

EXTRACT parameter *94*
EXTSORT parameter *40*, *94*
EXTTERM parameter *48*

F

FDEFCENT attribute *399*, *409*
field name attributes *59*
field names windows *455*
field variables *295*
field-level sliding window *409*, *411*
    DEFCENT attribute *411*
    MODIFY requests and *413*
    YRTHRESH attribute *411*
FIELDNAME parameter *30*, *40*, *95*
fields *330*
    ?FF command *335*
    displaying *330*
file contents windows *456*
file directory table *333*
    ? FDT command *333*
file names windows *454*
    creating *497*
FILE parameter *95*
file utilities *385*
file-level sliding window *409*
    FDEFCENT attribute *409*
    FYRTHRESH attribute *409*
FILEDEF command *190*, *191*, *195*
FILEDEF
    CONCAT *192*
FILENAME parameter *40*, *95*
files *385*
    closing *202*
    displaying information for *388*
    loading *385*, *386*
    reading from *259*
    unloading *385*, *387*
    writing to *258*, *259*
FILTER parameter *40*, *96*
filters *109*
    maintaining across joins *109*
FINANCE data source *547*, *548*
FIXRETRIEVE parameter *37*, *96*
FLOATMAPPING parameter *97*
flow of control *203*, *213*
FMI *355*
FML rows *99*
FOC144 parameter *40*, *97*
FOC441 warnings *147*
FOCALLOC parameter *30*
FOCEXURL parameter *98*
FOCFIRSTPAGE parameter *45*, *98*
FOCSTACK parameter *37*, *98*
FOCUS Database Server *353*
FOCUS facilities *385*
    performance enhancement *385*
FOCUS HOLD files *85*
FOCUS Metadata Interface *355*
FOCUS Report Writer *130*
format specifications *272*

formatting currency *445*
formatting report output *124*
FORMULTIPLE parameter *40*, *99*
full-screen data entry *230*, *271*
function keys *475*
    assigning phrases to *382*, *383*
functions *275*
    calling *293–295*
    running *203*
FYRTHRESH attribute *399*, *409*

G

GETUSER function *238*
GGDEMOG data source *564*
GGORDER data source *564*
GGPRODS data source *564*
GGSALES data source *564*
GGSTORES data source *564*
global sliding window *402*, *403*
global variable values *354*
global variables *204*, *230*, *233*, *234*
    naming *231*, *232*
    querying *246*
Gotham Grinds data sources *564*
goto values *450*, *473*, *509*
graph parameters *346*
graph request *26*

H

HDAY parameter *33*, *100*
HIDENULLACRS parameter *45*, *100*
HIPERFOCUS parameter *30*
HIPERINSTALL parameter *30*
HLDCOM_TRIMANV parameter *100*
HLISUDUMP parameter *37*
HLISUTRACE parameter *37*
HNODATA parameter *40*, *101*
HOLD fields *339*
HOLD files *93*
    comments *104*
    DBA information *104*
    default format *102*
    displaying fields *103*
HOLD Master File *59*, *102*
HOLDATTR parameter *40*, *102*
HOLDFORMAT parameter *30*, *102*
HOLDLIST parameter *30*, *103*
HOLDMISS parameter *30*, *103*
HOLDSTAT parameter *30*, *104*
holiday file *100*
horizontal menus *452*
    creating *461*
HOTMENU parameter *49*
HTML display page *105*
HTML output *137*
    SHOWBLANKS *137*
HTMLARCHIVE parameter *30*, *104*
HTMLCSS parameter *45*, *105*
HTMLEMBEDIMG parameter *105*
HTMLENCODE parameter *106*

I

IBMLE parameter *37*
identifying currency values *436*
images
    embedding in .htm file *105*
IMMEDTYPE parameter *37*
impact analysis, reporting on *363*
implicit prompting *247*
implied prompting *230*, *271*
IN FILE in Dialogue Manager *251*
Include component *226*
INDEX parameter *31*, *106*
indexed variables *284*
    creating *284*
indexes, reporting on *364*
indexing *106*
integrating euro currency *431*
interactive data entry *230*
internal blanks *137*
internal Cascading Style Sheets *105*
invalid date formats *57*
ITEMS data source *562*, *563*

J

JOBFILE data source *538*, *539*
JOBHIST data source;sample data sources
    JOBHIST *553*
JOBLIST data source;sample data sources
    JOBLIST *553*
JOIN CLEAR command *109*
JOIN command *108*, *109*
    maintaining filters *109*
join structures *339*
    displaying *339*
JOIN_LENGTH_MODE parameter *40*, *107*
joining data sources *108*
joining fields *108*
JOINLM parameter *40*, *107*
JOINOPT parameter *40*, *108*
JSURL parameter *40*

K

KEEPDEFINES parameter *31*, *40*, *108*
KEEPFILTERS SET parameter *109*
key fields *61*
keyed retrieval *96*
keys, reporting on *365*

L

lagging values *108*
LANG parameter *41*, *110*
LAYOUTGRID parameter *45*, *112*
leading blanks *137*
leading zeros *68*, *112*
LEADZERO parameter *33*, *41*, *112*
LEDGER data source *546*, *547*
LEFTMARGIN parameter *46*, *112*
legacy dates *401*
    sliding window and *401*

LET command *371*–*374*, *376*, *377*, *379*, *382*, *383*
LET ECHO command *380*
LET substitutions *341*, *371*
    debugging *380*
    displaying *379*
limiting record retrieval *133*
LINES parameter *46*, *113*
LIST requests *66*
LOAD command *385*, *386*
LOAD facility *385*
load procedures *535*
loaded files *341*, *388*
loading files *385*, *386*
    Access Files *387*
    Master Files *387*
loading procedures *387*
local variables *204*, *230*, *232*
    naming *231*, *232*
LOCATOR data source;sample data sources
    LOCATOR *554*
long field names *95*
LONGSYNM *198*
looped procedures *222*
    -REPEAT command *222*–*224*
    -SET command *222*, *225*
looping *203*, *222*–*224*
LRECL *120*

M

Main Menu *502*
manipulating dates *33*
mapping application directories
    manually *159*
mask option of EDIT function *290*, *291*
Master Files *195*, *409*, *410*, *535*
    comments *104*
    DBA information *104*
    defining sliding windows in *409*–*413*
    loading *387*
    parsing *389*–*391*
    running multiple requests *389*, *390*
    SAVEDMASTERS parameter *134*
    saving *391*
    saving in memory *389*, *390*
    TITLE parameter *143*
    virtual fields *420*, *421*
MASTER parameter *31*
MATCHCOLUMNORDER parameter *113*
maximum record length *115*
maximum report panel line width *124*
MAXLRECL parameter *31*, *115*
MDI builds *116*
MDI files *116*
MDICARDWARN parameter *31*, *115*
MDIENCODING parameter *31*, *116*
MDIPROGRESS parameter *31*, *116*
memory *36*
    controlling *36*

menus *451*
creating *449*
horizontal *452*, *461*
pulldown *464*
vertical *452*
MESSAGE parameter *41*, *42*, *117*
messages *210*
displaying *203*, *204*, *210*
metadata *29*, *187*
accessing *187*
processing *29*
storing *29*
MINIO facility *385*
MINIO parameter *31*, *392–394*
MISS_ON parameter *28*
missing data characters *101*
missing field values *70*
missing report data *119*
missing segment instances *57*
MISSING=ON attribute *101*
MISSINGTEST parameter *29*, *118*
MODIFY requests *397*
DEFCENT parameter *401*
YRTHRESH parameter *401*
MOVIES data source *562*
multi-input windows *460*
creating *466*
MULTIPATH parameter *41*, *119*

N

National Language Support (NLS) *340*
? NLS command *340*
natural date literals *255*
navigating procedures *217*
branching *218*, *219*
calling another procedure *228*
incorporating another procedure *226*
looping *222*
nesting procedures *228*
NLS (National Language Support) *340*
? NLS command *340*
NODATA parameter *41*, *119*
non-data files *188*
non-FOCUS files *203*, *204*
reading *203*
NULL parameter *31*, *120*
numbering output pages *122*
numeric amper variables in functions *296*
numeric data types *108*
numeric precision *86*
numerical notation *67*

O

OLDSTYRECLEN parameter *120*
ON GRAPH SET command *27*
ON TABLE SET command *90*
ONFIELD parameter *41*, *120*
open-ended procedures *229*, *230*
operating system commands *307*

optimizing retrieval paths *61*
ORIENTATION parameter *46*, *121*
output files *177*, *178*
    creating *177*, *178*
output page numbering *98*, *122*
OVERFLOWCHAR parameter *46*, *121*

P

page numbering *98*
page orientation *121*
PAGE parameter *46*
PAGE-NUM parameter *46*, *122*
PAGE-SCALE parameter *46*
PAGESIZE parameter *46*, *122*
PANEL parameter *46*, *124*
PAPER parameter *46*
parameter settings *344*
parameter values *245*
    querying settings *245*–*247*
parameters
    querying value settings *247*
PARTITION_ON parameter *29*, *41*
PASS parameter *48*, *125*
passwords *211*
    permanent *128*
    setting *203*, *211*
    unchangeable *128*
PAUSE parameter *41*, *42*
PCHOLD file *103*
    displaying fields *103*
PCOMMA parameter *32*, *125*
PCTFORMAT parameter *46*, *127*
PDFLINETERM parameter *127*
PERMPASS parameter *48*, *128*
PERSINFO data source;sample data sources
    PERSINFO *555*
PF key assignments *343*
PF keys *129*, *475*
PFnn command *475*
PFnn parameter *41*, *42*
PHONETIC_ALGORITHM parameter *129*
physical file location *179*
    mapping to *159*
positional variables *268*, *269*
PostScript reports *144*
PREFIX parameter *32*
prepending applications to search path *157*
PRFTITLE parameter *46*, *129*
PRINT parameter *46*, *129*
PRINT requests *66*
PRINTDST parameter *41*, *130*
printed output *113*
PRINTPLUS parameter *46*, *130*
procedures *48*, *188*, *201*, *387*
    branching *218*, *219*
    calling *226*–*229*
    calling;EXEC command *214*
    canceling *216*
    comments in *209*, *210*
    creating *208*, *229*, *230*

procedures *48*, *188*, *201*, *387*
debugging *300*
exiting *203*, *215*
including *203*, *226*
including comments *210*
loading *387*
looping *222*
navigating *217–219*
nesting *228*
prompting *230*
rules for creating *208*
running *205*, *206*, *213*, *214*, *216*
SAMPLE *479*
sample in Window Painter *499*
screening values in *276*
security *211*, *212*, *217*
startup *212*
variables and *230*, *276*
processing data *29*
processing Dialogue Manager procedures *205*, *206*
PROD data source *543*, *544*
product release number *343*
profiles *156*
applications and *156*
prompting for variable values *230*, *247*, *270*, *271*
supplying text *274*
PSPAGESETUP parameter *47*, *131*
pull-down menus *464*
punctuating large numbers *443–445*

Q

QUALCHAR parameter *32*, *41*, *131*
qualified column titles *132*
qualified field names *95*, *131*
QUALTITLES parameter *47*, *132*
query commands *328*, *347*, *408*
? && *354*
? COMBINE *330*
? COMBINE command *330*
? DEFINE *330*, *331*
? FDT *333*, *334*
? FILE *336*, *337*
? FUNCTION *338*
? HOLD *339*
? JOIN *339*
? LET *341*
? LOAD *341*, *342*, *388*
? n *342*
? NLS *340*, *341*
? PFKEY *343*
? RELEASE *343*, *344*
? SET *344*
? SET EUROFILE *332*, *443*
? SET GRAPH *346*
? STAT *348*
? STYLE *351*
? SU *353*
? USE *353*, *354*
?F *332*
?FF *335*

query commands *328*, *347*, *408*
SITECODE *347*
querying procedure status *300*, *306*
QUIT command *217*
quote-delimited *285*
quote-delimited string *285*
QUOTEDSTRING *285*

R

RANK parameter *132*
REBUILDMSG parameter *47*
RECAP-COUNT parameter *47*, *133*
RECORDLIMIT parameter *133*
records *203*, *256*
accessing *203*
reading from a file *256*
writing *204*
writing to a file *256*
REGION data source *548*
release number *343*
remote drill-downs *98*
reports *38*
displaying *42*
requests *374*
translating *374*
resizing graphs *60*
resources *391*
retrieval paths *61*
retrieving comma-delimited files *125*
retrieving records *133*
return value display windows *456*
return values *472*
RIGHTMARGIN parameter *47*, *134*
rounding using DMPRECSION *254*
RPAGESET parameter *134*
rules for supplying values *248*

S

SALES data source *541–543*
SALHIST data source;sample data sources
SALHIST *556*
sample data sources *535*
CAR *544*, *545*
Century Corp *570*
COMASTER Master File *558*
COURSE *552*, *553*
COURSES *549*
EDUCFILE *540*, *541*
EMPLOYEE *535*, *537*, *538*
FINANCE *547*, *548*
Gotham Grinds *564*
ITEMS *562*, *563*
JOBFILE *538*, *539*
LEDGER *546*, *547*
MOVIES *562*
PROD *543*, *544*
REGION *548*
SALES *541–543*
TRAINING *551*, *552*, *556*, *557*
VIDEOTR2 *563*, *564*

sample data sources *535*
VideoTrk *559–561*
SAVEDMASTERS parameter *32*, *134*, *389*, *390*
limitations *391*
querying *390*
saving in memory; resources *391*
SAVEMATRIX parameter *42*, *135*
saving Master Files in memory *389–391*
SBORDER parameter *49*
SCREEN parameter *49*
screening data sources *96*
screening values with -IF tests *276–278*
screens *202*, *499*
clearing *202*
Entry Menu *500*
Main Menu *502*
Utilities Menu *521*
Window Creation Menu *504*
Window Design Screen *506*
Window Options Menu *509*
search paths *156*
adding names to *157*, *158*
appending applications manually *158*
commands *153*
prepending applications manually *157*
searching for key fields *61*
security *48*
SET command *23–27*
parameters *27*
SET DROPBLNKLINE *86*
SET HIPERFOCUS facility *385*
SET parameter types *27*
SET parameters *23–27*, *50*
ACCBLN *54*
ACROSSLINE *54*
ACROSSPRT *54*
ACROSSTITLE *55*
ACRSVRBTITL *56*
ALL *57*
ALLOWCVTERR *57*, *428*
ASNAMES *59*
AUTOFIT *60*
AUTOINDEX *60*
AUTOPATH *61*
AUTOSTRATEGY *61*
AUTOTABLEF *62*
BASEURL *62*
BINS *62*
BLANKINDENT *63*
BOTTOMMARGIN *64*
BUSDAYS *64*
BYDISPLAY *64*
BYPANEL *65*
CACHE *66*
CARTESIAN *66*
CDN *67*
CENT-ZERO *68*
CNOTATION *68*
COLLATION *69*
COMPMISS *70*

SET parameters *23–27*, *50*
COMPOUND *70*
COMPUTE *71*
COUNTWIDTH *72*
CSSURL *72*
CURRENCY_DISPLAY *73*
CURRENCY_ISO_CODE *73*
CURRENCY_PRINT_ISO *74*
CURRSYMB *74*
CURSYM_D *75*
CURSYM_E *75*
CURSYM_F *76*
CURSYM_G *76*
CURSYM_L *76*
CURSYM_Y *77*
DATE_ORDER *77*
DATE_SEPARATOR *78*
DATEDISPLAY *79*, *428*
DATEFNS *79*
DATEFORMAT *80*
DATETIME *80*
DB_INFILE *81*
DBACSENSITIV *81*
DBAJOIN *82*
DBASOURCE *82*
DEFCENT *83*, *402*
DEFECHO *84*, *301*
DEFINES *84*
DIRECTHOLD *85*
displaying settings *344*

SET parameters *23–27*, *50*
DMH_LOOPLIM *85*
DMH_STACKLIM *85*
DMPRECISION *86*
DTSTRICT *87*
EMBEDDABLE *88*
EMPTYREPORT *88*
EQTEST *89*
ERROROUT *90*
ESTRECORDS *90*
EUROFILE *91*, *438*
EXL2KLANG *91*
EXL2KTXTDATE *92*
EXTAGGR *92*
EXTENDNUM *93*
EXTHOLD *93*
EXTRACT *94*
EXTSORT *94*
FIELDNAME *95*
FILE *95*
FILENAME *95*
FILTER *96*
FIXRETRIEVE *96*
FLOATMAPPING *97*
FOC144 *97*
FOCEXURL *98*
FOCFIRSTPAGE *98*
FOCSTACK *98*
FORMULTIPLE *40*, *99*
HDAY *100*

SET parameters *23–27*, *50*
HIDENULLACRS *100*
HLDCOM_TRIMANV *100*
HNODATA *101*
HOLDATTR *102*
HOLDFORMAT *102*
HOLDLIST *103*
HOLDMISS *103*
HOLDSTAT *104*
HTMLARCHIVE *104*
HTMLCSS *105*
HTMLEMBEDIMG *105*
HTMLENCODE *106*
INDEX *106*
JOIN_LENGTH_MODE *40*, *107*
JOINLM *40*, *107*
JOINOPT *108*
KEEPDEFINES *108*
KEEPFILTERS *109*
LANG *110*
LAYOUTGRID *112*
LEADZERO *112*
LEFTMARGIN *112*
LINES *113*
MATCHCOLUMNORDER *113*
MAXLRECL *115*
MDICARDWARN *115*
MDIENCODING *116*
MDIPROGRESS *116*
MESSAGE *117*
SET parameters *23–27*, *50*
MINIO *392–394*
MISSINGTEST *118*
MULTIPATH *119*
NODATA *119*
NULL *120*
OLDSTYRECLEN *120*
ONFIELD *120*
ORIENTATION *121*
OVERFLOWCHAR *121*
PAGE-NUM *122*
PAGESIZE *122*
PANEL *124*
PASS *125*
PCOMMA *125*
PCTFORMAT *127*
PDFLINETERM *127*
PERMPASS *128*
PHONETIC_ALGORITHM *129*
PRFTITLE *129*
PRINT *129*
PRINTDST *130*
PRINTPLUS *130*
PSPAGESETUP *131*
QUALCHAR *131*
QUALTITLES *132*
RANK *132*
RECAPCOUNT *133*
RECORDLIMIT *133*
RIGHTMARGIN *134*

SET parameters *23–27*, *50*
RPAGESET *134*
SAVEDMASTERS *134*, *389–391*
SAVEMATRIX *135*
SHADOW *135*
SHIFT *136*
SHOWBLANKS *137*
SPACES *138*
SQLTOPTIF *139*
SQUEEZE *139*
STYLESHEET *140*
SUBTOTAL *141*
SUMMARYLINES *141*
SUMPREFIX *142*
TESTDATE *405*
TIME_SEPARATOR *143*
TITLE *143*
TITLELINE *143*
TOPMARGIN *144*
UNITS *144*
USER *145*
USERFCHK *145*
USERFNS *146*
WARNING *147*
WEEKFIRST *147*
XLSXPAGEBRKIGNORE *149*
XRETRIEVAL *149*
YRTHRESH *149*
SET SAVEDMASTERS facility *385*
SET SQUEEZE parameter *139*
SET WPMINWIDTH *148*
setting parameters *23–27*
setting multiple parameters *25*
SHADOW parameter *32*, *135*
SHIFT parameter *32*, *136*
SHORTPATH parameter *42*, *136*
SHOWBLANKS parameter *137*
sink machine *195*
SITECODE command *347*
sliding window *397*, *398*
calculated value *422–425*
date format *401*
date options *427*
DEFCENT parameter *401*
defining in a Master File *409–412*
defining with SET *402*, *403*, *405*, *406*
dynamic window *400*, *405*, *406*
field-level *409*, *411–414*
file-level *409*, *410*, *414*
global *402*, *403*
legacy dates *401*
subroutines and *419*, *426*
YRTHRESH parameter *401*
sort groups *64*
sorting externally *94*
sorting fields *64*
sorting records *90*
SORTLIB parameter *42*
SORTMATRIX parameter *42*

SORTMEMORY parameter *42*
SPACES parameter *47*, *138*
sparse ranking *132*
special variables *245*
    &CURSOR *245*
    &CURSORAT *245*
    &ECHO *245*
    &PFKEY *245*
    &QUIT *245*
    &STACK *245*
    &WINDOWNAME *245*
    &WINDOWVALUE *245*
specifying business days *64*
specifying currency symbols *74–77*
specifying punctuation *67*
specifying qualifying characters *131*
specifying value ranges *273*
SQL Translator *139*
SQLTOPTTF parameter *37*, *139*
SQUEEZE parameter *47*
stacked commands *205*, *206*, *221*
    executing *203*, *214*, *215*
startup procedures *212*
statistical variables *204*, *230*, *242*, *244*
    querying *246*
    testing *221*
stored procedures, reporting on *366*
storing data *29*
storing parsed Master Files in memory *389*, *390*
strict processing *87*
strings *285*
structure diagrams *535*
structured HOLD files *94*
StyleSheet bottom boundary *64*
StyleSheet margins *112*
STYLESHEET parameter *47*, *140*
StyleSheet parameters *134*
StyleSheets *351*
    ? STYLE *351*
    formatting *140*
    page orientation *121*
    SQUEEZE parameter *139*
styling HTML reports *72*
styling reports *121*
SU machine *353*
subroutines *203*
    calling *293*
    running *203*
substitutions *371–377*
    assigning to function keys *382*, *383*
    debugging *380*
SUBTOTAL parameter *47*
SUBTOTAL SET parameter *141*
SUMMARYLINES SET parameter *141*
SUMPREFIX parameter *42*, *142*
supplying default variable values *251*
supplying variable values *25*, *247*
suppressing blank lines *86*
SUSI parameter *32*
SUTABSIZE parameter *32*

sync machines *353*
synonym locations *161*
synonyms *198*
    long names on z/OS *198*
sysapps *357*
syscolum *358*
sysdeffn *359*
syserr *360*
sysfiles *361*
sysimp *363*
sysindex *364*
syskeys *365*
sysrpdir *366*
sysset *367*
syssqlop *367*
systable *368*
system date *428*
system defaults *323*
system tables *355*
    sysapps *357*
    syscolum *358*
    sysdeffn *359*
    syserr *360*
    sysfiles *361*
    sysimp *363*
    sysindex *364*
    syskeys *365*
    sysrpdir *366*
    sysset *367*
    syssqlop *367*
system tables *355*
    systable *368*
    sysvdtp *369*
system variables *204*, *230*, *234*, *241*, *300*
    &DATE *240*
    &DMY *235*
    &DMYY *235*
    &FOCBORDERS *238*
    &FOCCODEPAGE *235*
    &FOCCPU *236*
    &FOCEXTTRM *236*
    &FOCFIELDNAME *236*
    &FOCFOCEXEC *240*
    &FOCINCLUDE *237*
    &FOCMODE *237*
    &FOCNEXTPAGE *237*
    &FOCPRINT *237*
    &FOCPUTLVL *237*
    &FOCQUALCHAR *237*
    &FOCSUER *238*
    &FOCTRMSD *238*
    &FOCTRMSW *238*
    &FOCTRMTYP *238*
    &HIPERFOCUS *238*
    &MDYY *238*
    &RETCODE *239*
    &SETFILE *239*
    &TOD *240*
    &YYMD *240*, *241*
    ECHO *235*

system variables *204*, *230*, *234*, *241*, *300*
    EXITRC *235*
    testing *221*
sysvdtp *369*

T

TABLE requests *88*
TABLEF commands *139*
tables, reporting on *368*
TED (text editor) *208*
temporary files *196*
TERMINAL parameter *47*
TESTDATE parameter *34*, *405*
text display windows *454*
    creating *483*, *487*
text input windows *453*
TIME_SEPARATOR parameter *143*
TITLE attribute *102*
TITLE parameter *42*, *143*
TITLELINE parameter *47*, *143*
TOPMARGIN parameter *47*, *144*
TRACKIO parameter *32*
trailing blanks *291*
    deleting *292*
TRAINING data source *551*, *552*, *556*, *557*
TRANTERM parameter *48*
TRMOUT parameter *49*
TRUNCATE function *291*, *292*
truncating leading zeros *112*

U

unconditional branching *203*, *218*
    -GOTO command *218*, *219*
UNITS parameter *48*, *144*
Universal Naming Convention (UNC) *160*
UNLOAD command *385*, *387*
unloading files *385*, *387*
URL location *62*
USE command *190*, *195*
USER parameter *48*, *145*
USERFCHK parameter *145*
USERFNS parameter *146*
Utilities Menu *521*

V

valid values *273*, *274*
values *289*, *450*
    specifying ranges *273*
    default *323*
    defining *289*, *290*
    goto *473*, *509*
    prompting for *271*, *274*
    querying *245*, *246*
    screening *276–278*
    setting *294*
    supplying *230*, *248*, *254*, *256*, *267*, *268*, *270–272*, *274*
    valid *273*, *274*
variable name length *248*

variable types *204*
amper *230*
global *204*, *230*, *233*
local *204*, *230*, *232*
positional *268*
special *245*
statistical *204*, *230*, *242*
system *204*, *230*, *234*
variable values *267*, *268*
-DEFAULT command and *251*, *310*
? && *354*
dates *255*
length *248*
querying *354*
supplying *25*, *26*, *247–249*, *251*, *252*, *254*, *256*, *267*, *268*, *270*, *272–274*, *319*
supplying default *251*
verifying *247*
variables *230*, *275*, *285*
appending values *284*
assigning values to *203*, *294*
changing commands *295*
computing *289*, *290*
concatenating *283*
evaluating *280*
procedures and *230*
querying *245*, *246*
quote-delimited *285*
setting *203*
substitution *230*
variables *230*, *275*, *285*
supplying values *25*, *26*, *248*, *254*, *256*, *267*, *268*, *270*, *272*, *274*
vertical menus *452*
creating *489*, *495*
VIDEOTR2 data source *563*, *564*
VideoTrk data source *559–561*
viewing source code *117*
virtual fields *84*, *108*, *330*, *416*
? DEFINE command *330*
defining windows for *416*, *418*, *420*, *421*
displaying *330*
visual overflow *93*

W

warning message *115*
MDICARDWARN *115*
WARNING parameter *147*
web archive documents *104*
Web Console *156*
application paths *156*
WEBARCHIVE parameter *32*
WEBTAB parameter *48*
WEEKFIRST parameter *32*, *147*
WIDTH parameter *32*
window applications *470*
WINDOW COMPILE command *532*
Window Creation Menu *504*
Window Design Screen *506*
Window facility *470*

window files *450*, *451*
creating *481*
Window Options Menu *509*
Display list *513*
Heading *512*
Help window *515*, *520*
Hide list *514*
Line break *518*
Multi-select *519*
Popup *514*
Quit *520*
Show a window *512*
Unshow a window *512*
WINDOW PAINT command *500*
Window Painter *204*, *449*
goto values *450*
invoking *500*
main menu *483*
screens *499*
tutorial *478*
window files *450*
windows *451*, *458*
&WINDOWNAME variable *474*
&WINDOWVALUE variable *474*
accessing a network drive *160*
creating *449*, *461*
field names *455*
file contents *456*
file names *454*
horizontal *452*, *461*
windows *451*, *458*
multi-input *460*, *466*
procedures *469*
return value display *456*
returning to caller *474*
running *476*, *477*
text display *454*
text input *453*
types *452*
vertical menu *452*
word substitutions *341*, *371–377*
assigning to function keys *382*, *383*
debugging *380*
WPMINWIDTH parameter *32*, *148*

X

XFOCUSBINS parameter *33*
XLSXPAGEBRKIGNORE parameter *48*, *149*
XRETRIEVAL parameter *33*, *149*

Y

Y2K compliance *79*
year-2000 compliance *79*
YRTHRESH parameter *34*, *149*, *398*, *399*, *401*, *402*
COMPUTE command *422*
DEFINE command *416*
MODIFY requests and *401*
querying *408*
with COMPUTE *424*

Z

zero field values *54*

ZIIP parameter *38*