

# FOCUS<sup>®</sup> for S/390<sup>®</sup>

Developing Applications  
Version 7.1

# Contents

- 1 Customizing Your Environment ..... 1-1**
  - The SET Command ..... 1-2
  - SET Parameter Syntax ..... 1-3
  
- 2 Querying Your Environment ..... 2-1**
  - Using Query Commands ..... 2-2
  - Displaying Combined Structures ..... 2-3
  - Displaying Virtual Fields ..... 2-4
  - Displaying the Currency Data Source in Effect ..... 2-5
  - Displaying Available Fields ..... 2-5
  - Displaying the File Directory Table ..... 2-6
  - Displaying Field Information for a Master File ..... 2-8
  - Displaying Data Source Statistics ..... 2-9
    - Determining the Percentage of File Disorganization ..... 2-11
  - Displaying DEFINE Functions ..... 2-11
  - Displaying HOLD Fields ..... 2-12
  - Displaying JOIN Structures ..... 2-13
  - Displaying National Language Support ..... 2-14
  - Displaying LET Substitutions ..... 2-14
  - Displaying Information About Loaded Files ..... 2-15
  - Displaying Explanations of Error Messages ..... 2-15
  - Querying Which PTFs Have Been Applied for a Specific Release ..... 2-16
  - Displaying the Release Number ..... 2-17
  - Displaying Parameter Settings ..... 2-17
  - Displaying Graph Parameters ..... 2-21
  - Displaying Command Statistics ..... 2-22
  - Displaying Information About the SU Machine ..... 2-24
  - Displaying Global Variable Values ..... 2-25
  - Displaying StyleSheet Parameter Settings ..... 2-26
  - Displaying Data Sources Specified With USE ..... 2-26

<b>3 Using Functions and Subroutines .....</b>	<b>3-1</b>
What Is the Difference Between a Function and a Subroutine? .....	3-2
Types of Functions and Subroutines .....	3-3
Bit Functions and Subroutines .....	3-3
Character Functions and Subroutines .....	3-4
Data Source Functions and Subroutines .....	3-6
Date Functions and Subroutines .....	3-6
Decoding Functions and Subroutines .....	3-10
Format Conversion Functions and Subroutines .....	3-11
Numeric Functions and Subroutines .....	3-12
System Functions and Subroutines .....	3-13
Date Function and Subroutine Settings .....	3-14
Using Legacy Versions of Date Subroutines .....	3-14
Setting Business Day Units .....	3-15
Setting Holidays .....	3-16
Enabling Leading Zeros For Date Subroutines in Dialogue Manager .....	3-17
Subroutine Command (Call) Syntax .....	3-18
Types of Arguments in Subroutine Calls .....	3-19
Rules for Arguments in Subroutine Calls .....	3-20
Using Subroutine Calls in FOCUS Functions .....	3-21
Using Subroutine Calls in DEFINE, COMPUTE, and VALIDATE Commands .....	3-22
Using Subroutine Calls in WHERE and IF Tests .....	3-23
Using Subroutine Calls in -SET Control Commands .....	3-23
Using Subroutine Calls in -IF and IF Branching Commands .....	3-24
Operating System -RUN Commands .....	3-26
Using Subroutine Calls in WHEN Criteria .....	3-27
Using Subroutine Calls in RECAP Commands .....	3-28
Storing and Accessing External Subroutines .....	3-30
Storing and Accessing Subroutines on MVS .....	3-30
Dynamic Language Environment Support .....	3-32
Storing and Accessing Subroutines on VM/CMS .....	3-34
Alphabetical List of Functions and Subroutines .....	3-36
ABS: Calculating Absolute Value .....	3-36
ARGLEN: Measuring Argument Length .....	3-37
ASIS: Distinguishing Between a Blank and a Zero .....	3-38
ATODBL: Converting Alphanumeric Strings to a Double-Precision Number .....	3-39
AYM: Adding or Subtracting Months to or From Dates .....	3-43
AYMD: Adding or Subtracting Days to or From Dates .....	3-45
BAR: Producing Bar Charts .....	3-47
BITSON: Determining If Bits Are On or Off .....	3-49
BITVAL: Evaluating Bit Strings as Binary Integers .....	3-50
BYTVAL: Translating a Character to Its ASCII or EBCDIC Code .....	3-52

CHGDAT: Changing Date Formats .....	3-53
CHKFMT: Checking String Format.....	3-56
CHKPCK: Validating Packed Fields.....	3-59
CTRAN: Translating One Character to Another .....	3-62
CTRFLD: Centering a Character String.....	3-68
DA Subroutines: Converting a Date to an Integer .....	3-70
DATEADD: Adding or Subtracting Date Units to or From a Date.....	3-72
DATECVT: Converting Date Formats.....	3-77
DATEDIF: Finding the Difference Between Two Dates .....	3-78
DATEMOV: Moving Dates to a Significant Point.....	3-80
DECODE: Decoding Values .....	3-83
DMY, MDY, YMD: Calculating the Difference Between Two Dates.....	3-86
DOWK and DOWKL: Finding the Day of the Week.....	3-88
DT Subroutines: Converting an Integer to a Date .....	3-89
EDIT: Converting the Format of a Field .....	3-91
EDIT: Extracting or Adding Characters.....	3-93
EXP: Raising “e” to the Nth Power.....	3-95
EXPN: Evaluating Scientific Notation.....	3-96
FEXERR: Retrieving FOCUS Error Messages .....	3-96
FINDMEM: Finding a Member of a Partitioned Data Set .....	3-97
FTOA: Converting a Number to Alphanumeric Format .....	3-100
GETPDS: Determining if a Member of a Partitioned Data Set Exists .....	3-101
GETTOK: Getting a Token From a String.....	3-106
GETUSER: Retrieving the User ID .....	3-108
GREGDT: Converting From Julian to Gregorian Format .....	3-109
HADD: Incrementing a Date-Time Field .....	3-112
HCNVRT: Converting a Date-Time Field to Alphanumeric Format .....	3-113
HDATE: Converting the Date Portion of a Date-Time Field to a Date Format .....	3-114
HDIFF: Finding the Number of Units Between Two Date-Time Values.....	3-115
HDTTM: Converting a Date field to a Date-Time Field.....	3-116
HEXBYT: Converting a Number to a Character.....	3-117
HGETC: Storing the Current Date and Time in a Date-Time Field.....	3-119
HHMMSS: Returning the Current Time .....	3-120
HINPUT: Converting an Alphanumeric String to a Date-Time Value.....	3-121
HMIDNT: Setting the Time Portion of a Date-Time Field to Midnight .....	3-122
HNAME: Extracting a Date-Time Component in Alphanumeric Format.....	3-123
HPART: Returning a Date-Time Component in Numeric Format.....	3-125
HSETPT: Inserting a Component Into a Date-Time Field .....	3-126
HTIME: Converting the Time Portion of a Date-Time Field to a Number .....	3-127
IMOD, FMOD, and DMOD: Calculating the Remainder From a Division .....	3-128
INT: Finding the Greatest Integer .....	3-129
ITONUM: Converting Large Binary Integers to Double-Precision .....	3-130
ITOPACK: Converting Large Binary Integers to Packed-Decimal Format.....	3-132
ITOZ: Converting to Zoned Format .....	3-134
JULDAT: Converting From Gregorian to Julian Format .....	3-135
LAST: Retrieving the Preceding Value.....	3-137

LCWORD: Converting Letters in a Word to Mixed Case.....	3-138
LJUST: Left-justifying a String .....	3-140
LOCASE: Converting Text to Lowercase.....	3-142
LOG: Calculating the Natural Logarithm.....	3-143
MAX and MIN: Finding the Maximum or Minimum Value.....	3-144
MVSDYNAM: Passing a DYNAM Command to the Command Processor.....	3-145
OVRLAY: Overlaying a Substring Within a String.....	3-149
PARAG: Dividing Text Into Smaller Lines .....	3-152
PKCOUT: Writing Packed Numbers of Different Lengths.....	3-154
POSIT: Finding Substring Position .....	3-156
PRDNOR, PRDUNI, RDNORM, and RDUNIF: Generating Random Numbers .....	3-158
RJUST: Right-justifying a String .....	3-161
SQRT: Calculating the Square Root.....	3-163
SUBSTR: Extracting a Substring .....	3-164
TODAY: Returning the Current Date .....	3-166
UFMT: Converting Alphanumeric to Hexadecimal .....	3-168
UPCASE: Converting Text to Uppercase .....	3-170
YM: Calculating Elapsed Months .....	3-174
<b>4 Managing Applications With Dialogue Manager .....</b>	<b>4-1</b>
Overview of Dialogue Manager Capabilities .....	4-2
Overview of Dialogue Manager Variables .....	4-6
Creating and Storing Procedures .....	4-6
Executing Procedures .....	4-7
Controlling Access to Data.....	4-7
Including Comments in a Procedure .....	4-8
Overview of Dialogue Manager Commands .....	4-9
Sending a Message to the User: -TYPE .....	4-11
Controlling Execution: -RUN, -EXIT, and -QUIT .....	4-12
Executing Stacked Commands and Continuing the Procedure: -RUN.....	4-12
Executing Stacked Commands and Exiting the Procedure: -EXIT .....	4-13
Canceling Execution of the Procedure: -QUIT .....	4-15
Branching .....	4-16
-GOTO Processing .....	4-16
Compound -IF Tests.....	4-19
Using Operators and Functions in -IF Tests.....	4-20
Screening Values With -IF Tests.....	4-20
Testing the Status of a Query .....	4-24
Looping .....	4-25
Ending a Loop .....	4-27
Using Expressions: -SET.....	4-27
Computing a New Variable .....	4-28

Using the DECODE Function .....	4-29
Using the EDIT Function .....	4-30
Using the TRUNCATE Function .....	4-31
Controlling a Loop With -SET .....	4-32
Setting a Date .....	4-33
Calling a Subroutine .....	4-33
Additional Facilities .....	4-36
Establishing Startup Conditions .....	4-36
Incorporating Multiple Procedures .....	4-37
Nesting Procedures With -INCLUDE .....	4-40
Using EXEC .....	4-41
Developing an Open-Ended Procedure .....	4-41
Debugging With &ECHO .....	4-42
Testing Dialogue Manager Command Logic With &STACK .....	4-43
Locking Procedure Users Out of FOCUS .....	4-44
Writing to Files: -WRITE .....	4-45
Using Variables in Procedures .....	4-49
Querying the Values of Variables .....	4-51
Local Variables .....	4-52
Global Variables .....	4-53
System Variables .....	4-54
Statistical Variables .....	4-60
Special Variables .....	4-62
Using Variables to Alter Commands .....	4-63
Evaluating a Variable Immediately .....	4-63
Concatenating Variables .....	4-65
Supplying Values for Variables at Run Time .....	4-66
Supplying Values Without Prompting .....	4-68
Supplying Values With -DEFAULTS .....	4-70
Supplying Values With -SET .....	4-71
Supplying Values With -READ .....	4-72
Direct Prompting With -PROMPT .....	4-73
Full-Screen Data Entry With -CRTFORM .....	4-75
Selecting Data From Menus and Windows With -WINDOW .....	4-75
Implied Prompting .....	4-75
Verifying Input Values .....	4-76
Dialogue Manager Quick Reference .....	4-80
System Defaults and Limits .....	4-97
<b>5 Defining a Word Substitution .....</b>	<b>5-1</b>
The LET Command .....	5-2
Variable Substitution .....	5-5
Null Substitution .....	5-7

Multiple-line Substitution.....	5-8
Recursive Substitution.....	5-8
Using LET Substitution in a COMPUTE or DEFINE Command.....	5-9
Checking Current LET Substitutions.....	5-10
Interactive LET Query: LET ECHO.....	5-10
Clearing LET Substitutions.....	5-11
Saving LET Substitutions in a File.....	5-12
Assigning Phrases to Function Keys.....	5-12
<b>6 Enhancing Application Performance.....</b>	<b>6-1</b>
FOCUS Facilities.....	6-2
Loading a File.....	6-2
Loading Master Files, FOCUS Procedures, and Access Files.....	6-4
Loading a Compiled MODIFY Request.....	6-5
Loading a MODIFY Request.....	6-6
Displaying Information About Loaded Files.....	6-6
Compiling a MODIFY Request.....	6-7
Accessing a FOCUS Data Source (MVS Only).....	6-8
Using MINIO.....	6-9
Determining if a Previous Command Used MINIO.....	6-10
<b>7 Working With Cross-Century Dates.....</b>	<b>7-1</b>
When Do You Use the Sliding Window Technique?.....	7-2
The Sliding Window Technique.....	7-2
Defining a Sliding Window.....	7-3
Creating a Dynamic Window Based on the Current Year.....	7-4
Applying the Sliding Window Technique.....	7-5
When to Supply Settings for DEFCENT and YRTHRESH.....	7-5
Date Validation.....	7-6
Defining a Global Window With SET.....	7-7
Defining a Dynamic Global Window With SET.....	7-10
Querying the Current Global Value of DEFCENT and YRTHRESH.....	7-12
Defining a File-Level or Field-Level Window in a Master File.....	7-13
Defining a Window for a Virtual Field.....	7-20
Defining a Window for a Calculated Value.....	7-26
Additional Support for Cross-Century Dates.....	7-31
<b>8 Euro Currency Support.....</b>	<b>8-1</b>

---

Integrating the Euro Currency .....	8-2
Converting Currencies.....	8-2
Preparing FOCUS to Process Currency Conversions.....	8-4
Creating the Currency Data Source.....	8-4
Identifying Fields That Contain Currency Data .....	8-6
Activating the Currency Data Source .....	8-8
Querying the Currency Data Source in Effect.....	8-9
Processing Currency Data .....	8-10
<b>9 Designing Windows With Window Painter.....</b>	<b>9-1</b>
Introduction .....	9-2
How Do Window Applications Work? .....	9-3
Window Files and Windows.....	9-4
Types of Windows You Can Create.....	9-5
Creating Windows.....	9-14
Integrating Windows and the FOCEXEC.....	9-21
Transferring Control in Window Applications.....	9-22
Return Values.....	9-24
Goto Values.....	9-25
Window System Variables .....	9-26
Testing Function Key Values .....	9-26
Executing a Window From the FOCUS Prompt .....	9-28
Tutorial: A Menu-Driven Application.....	9-29
Creating the Application FOCEXEC .....	9-31
Creating the Window File .....	9-33
Executing the Application.....	9-51
Window Painter Screens.....	9-51
Invoking Window Painter .....	9-52
Entry Menu.....	9-53
Main Menu .....	9-54
Window Creation Menu .....	9-57
Window Design Screen.....	9-59
Window Options Menu .....	9-61
Utilities Menu.....	9-72
Transferring Window Files.....	9-75
Creating a Transfer File.....	9-76
Transferring the File to the New Environment.....	9-77
Editing the Transfer File.....	9-77
Compiling the Transfer File .....	9-83
<b>A Master Files and Diagrams.....</b>	<b>A-1</b>
Creating Sample Data Sources .....	A-2



## Contents

---

The EMPLOYEE Data Source .....	A-3
The EMPLOYEE Master File .....	A-4
The EMPLOYEE Structure Diagram .....	A-5
The JOBFIL Data Source .....	A-6
The JOBFIL Master File .....	A-6
The JOBFIL Structure Diagram .....	A-6
The EDUCFIL Data Source .....	A-7
The EDUCFIL Master File .....	A-7
The EDUCFIL Structure Diagram .....	A-7
The SALES Data Source .....	A-8
The SALES Master File .....	A-8
The SALES Structure Diagram .....	A-9
The PROD Data Source .....	A-10
The PROD Master File .....	A-10
The PROD Structure Diagram .....	A-10
The CAR Data Source .....	A-11
The CAR Master File .....	A-11
The CAR Structure Diagram .....	A-12
The LEDGER Data Source .....	A-13
The LEDGER Master File .....	A-13
The LEDGER Structure Diagram .....	A-13
The FINANCE Data Source .....	A-14
The FINANCE Master File .....	A-14
The FINANCE Structure Diagram .....	A-14
The REGION Data Source .....	A-15
The REGION Master File .....	A-15
The REGION Structure Diagram .....	A-15
The COURSES Data Source .....	A-16
The COURSES Master File .....	A-16
The COURSES Structure Diagram .....	A-16
The EMPDATA Data Source .....	A-17
The EMPDATA Master File .....	A-17
The EMPDATA Structure Diagram .....	A-17
The EXPERS Data Source .....	A-18
The EXPERS Master File .....	A-18
The EXPERS Structure Diagram .....	A-18
The TRAINING Data Source .....	A-19
The TRAINING Master File .....	A-19
The TRAINING Structure Diagram .....	A-19
The PAYHIST File .....	A-20

The PAYHIST Master File.....	A-20
The PAYHIST Structure Diagram .....	A-20
The COMASTER File .....	A-21
The COMASTER Master File.....	A-22
The COMASTER Structure Diagram .....	A-23
The VideoTrk and MOVIES Data Sources .....	A-24
VideoTrk Master File .....	A-24
MOVIES Master File .....	A-24
VideoTrk Structure Diagram.....	A-25
MOVIES Structure Diagram .....	A-26
The VIDEOTR2 Data Source.....	A-26
The VIDEOTR2 Master File.....	A-26
The VIDEOTR2 Access File.....	A-27
The VIDEOTR2 Structure Diagram.....	A-28
The Gotham Grinds Data Sources .....	A-29
The GGDEMOG Data Source.....	A-29
The GGORDER Data Source .....	A-30
The GGPRODS Data Source .....	A-31
The GGSALES Data Source .....	A-32
The GGSTORES Data Source.....	A-33
<b>B Error Messages .....</b>	<b>B-1</b>
Accessing Error Files .....	B-2
Displaying Messages Online .....	B-3
<b>C Creating Your Own Subroutines .....</b>	<b>C-1</b>
Process Overview .....	C-2
Considerations for Writing Subroutines .....	C-3
Naming Conventions .....	C-3
Argument Considerations.....	C-3
Programming Considerations .....	C-5
Language Considerations .....	C-5
Programming Technique: Entry Points .....	C-7
Programming Technique: Subroutines With More Than 28 Arguments.....	C-9
Compilation and Storage .....	C-12
CMS: Compilation and Storage.....	C-12
MVS: Compilation and Storage .....	C-13
Testing the Subroutine.....	C-13
Example of a Custom Subroutine: The MTHNAM Subroutine .....	C-14
The MTHNAM Subroutine Written in FORTRAN .....	C-15
The MTHNAM Subroutine Written in COBOL .....	C-16
The MTHNAM Subroutine Written in PL/I.....	C-18

*Contents*

---

The MTHNAM Subroutine Written in BAL Assembler .....	C-19
The MTHNAM Subroutine Written in C .....	C-20
The MTHNAM Subroutine Called by a FOCUS Request .....	C-21
Subroutines Written in REXX .....	C-22
Using REXX Subroutines .....	C-22
Compiling FUSREXX Macros in CMS .....	C-32
<b>Index .....</b>	<b>I-1</b>

Cactus, EDA, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, Information Builders, the Information Builders logo, SmartMode, SNAPpack, TableTalk, and Web390 are registered trademarks and Parlay, SiteAnalyzer, SmartMart, and WebFOCUS are trademarks of Information Builders, Inc.

Acrobat and Adobe are registered trademarks of Adobe Systems Incorporated.

NOMAD is a registered trademark of Aonix.

UniVerse is a registered trademark of Ardent Software, Inc.

IRMA is a trademark of Attachmate Corporation.

Baan is a registered trademark of Baan Company N.V.

SUPRA and TOTAL are registered trademarks of Cincom Systems, Inc.

Impromptu is a registered trademark of Cognos.

Alpha, DEC, DECnet, NonStop, and VAX are registered trademarks and Tru64, OpenVMS, and VMS are trademarks of Compaq Computer Corporation.

CA-ACF2, CA-Datcom, CA-IDMS, CA-Top Secret, and Ingres are registered trademarks of Computer Associates International, Inc.

MODEL 204 and M204 are registered trademarks of Computer Corporation of America.

Paradox is a registered trademark of Corel Corporation.

StorHouse is a registered trademark of FileTek, Inc.

HP MPE/iX is a registered trademark of Hewlett Packard Corporation.

Informix is a registered trademark of Informix Software, Inc.

Intel is a registered trademark of Intel Corporation.

ACF/VTAM, AIX, AS/400, CICS, DB2, DRDA, Distributed Relational Database Architecture, IBM, MQSeries, MVS, OS/2, OS/400, RACF, RS/6000, S/390, VM/ESA, and VTAM are registered trademarks and DB2/2, HiperSpace, IMS, MVS/ESA, QMF, SQL/DS, VM/XA and WebSphere are trademarks of International Business Machines Corporation.

INTERSOLVE and Q+E are registered trademarks of INTERSOLVE.

Orbit is a registered trademark of Iona Technologies Inc.

Approach and DataLens are registered trademarks of Lotus Development Corporation.

ObjectView is a trademark of Matesys Corporation.

ActiveX, FrontPage, Microsoft, MS-DOS, PowerPoint, Visual Basic, Visual C++, Visual FoxPro, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Teradata is a registered trademark of NCR International, Inc.

Netscape, Netscape FastTrack Server, and Netscape Navigator are registered trademarks of Netscape Communications Corporation.

NetWare and Novell are registered trademarks of Novell, Inc.

CORBA is a trademark of Object Management Group, Inc.

Oracle is a registered trademark and Rdb is a trademark of Oracle Corporation.

PeopleSoft is a registered trademark of PeopleSoft, Inc.

INFOAccess is a trademark of Pioneer Systems, Inc.

Progress is a registered trademark of Progress Software Corporation.

Red Brick Warehouse is a trademark of Red Brick Systems.

SAP and SAP R/3 are registered trademarks and SAP Business Information Warehouse and SAP BW are trademarks of SAP AG.

Silverstream is a trademark of Silverstream Software.

ADABAS is a registered trademark of Software A.G.

CONNECT:Direct is a trademark of Sterling Commerce.

Java, JavaScript, NetDynamics, Solaris, and SunOS are trademarks of Sun Microsystems, Inc.

PowerBuilder and Sybase are registered trademarks and SQL Server is a trademark of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company, Ltd.

Due to the nature of this material, this document refers to numerous hardware and software products by their trade names. In most, if not all cases, these designations are claimed as trademarks or registered trademarks by their respective companies. It is not this publisher's intent to use any of these names generically. The reader is therefore cautioned to investigate all claimed trademark rights before using any of these names other than to refer to the product described.

Copyright © 2000, by Information Builders, Inc. All rights reserved. This manual, or parts thereof, may not be reproduced in any form without the written permission of Information Builders, Inc.

Printed in the U.S.A.

# Preface

This documentation describes FOCUS Application Development tools and environments for FOCUS® Version 7.1. This manual is intended for any FOCUS user. It is part of the FOCUS for S/390 documentation set.

The documentation set consists of the following components:

- The *Overview and Operating Environments* manual contains an introduction to FOCUS and FOCUS tools and describes how to use FOCUS in the VM/CMS and MVS (OS/390) environments.
- The *Creating Reports* manual describes FOCUS Reporting environments and features.
- The *Describing Data* manual explains how to create the metadata for the data sources that your FOCUS procedures will access.
- The *Developing Applications* manual describes FOCUS Application Development tools and environments.
- The *Maintaining Databases* manual describes FOCUS data management facilities and environments.

The users' documentation for FOCUS Version 7.1 is organized to provide you with a useful, comprehensive guide to FOCUS.

Chapters need not be read in the order in which they appear. Though FOCUS facilities and concepts are related, each chapter fully covers its respective topic. To enhance your understanding of a given topic, references to related topics throughout the documentation set are provided. The following pages detail documentation organization and conventions.

References to MVS apply to all supported versions of the OS/390 and MVS operating environments.

## How This Manual Is Organized

This manual includes the following chapters:

<b>Chapter/Appendix</b>		<b>Contents</b>
<b>1</b>	<i>Customizing Your Environment</i>	Lists commands you use to control output, work areas, and many other FOCUS features.
<b>2</b>	<i>Querying Your Environment</i>	Describes how to use query commands to retrieve information about the FOCUS environment.
<b>3</b>	<i>Using Functions and Subroutines</i>	Describes how to use the functions and subroutines available for manipulating numeric, date, and alphanumeric values.
<b>4</b>	<i>Managing Applications With Dialogue Manager</i>	Describes how to make report procedures more dynamic by using Dialogue Manager control statements and variables.
<b>5</b>	<i>Defining a Word Substitution</i>	Describes how to define string substitutions that can be used in FOCUS report requests.
<b>6</b>	<i>Enhancing Application Performance</i>	Describes FOCUS facilities for increasing the speed of your application.
<b>7</b>	<i>Working With Cross-Century Dates</i>	Describes techniques for assigning a century to dates stored with two-digit years.
<b>8</b>	<i>Euro Currency Support</i>	Describes how to perform currency conversions according to the rules established by the European Union.
<b>9</b>	<i>Designing Windows With Window Painter</i>	Describes how to create FOCUS windows and menus that work in conjunction with a FOCEXEC.
<b>A</b>	<i>Master Files and Diagrams</i>	Contains Master Files and diagrams of sample data sources used in the documentation examples.
<b>B</b>	<i>Error Messages</i>	Describes how to obtain additional information about error messages in FOCUS.
<b>C</b>	<i>Creating Your Own Subroutines</i>	Describes how to write subroutines that can be called from FOCUS.

# Summary of New Features

The new FOCUS features and enhancements described in this documentation set are listed in the following table.

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
Aggregating and Sorting Report Columns	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
DEFINE Functions	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Reporting From Independent Paths	<i>Creating Reports</i>	Chapter 5, <i>Selecting Records for Your Report</i>
HOLD FORMAT INTERNAL	<i>Creating Reports</i>	Chapter 11, <i>Saving and Reusing Report Output</i>
Increased Display Fields Support	<i>Creating Reports</i>	Chapter 1, <i>Creating Tabular Reports</i>
Embedding Text Fields in Headings	<i>Creating Reports</i>	Chapter 9, <i>Customizing Tabular Reports</i>
REXX Subroutines	<i>Developing Applications</i>	Appendix C, <i>Creating Your Own Subroutines</i>
Dialogue Manager TRUNCATE Function	<i>Developing Applications</i>	Chapter 4, <i>Managing Applications With Dialogue Manager</i>
CRTFORM HTML Translation	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
Two-Gigabyte and Partitioned FOCUS Database Support	<i>Describing Data</i>	Chapter 7, <i>Describing FOCUS Data Sources</i>
Token Delimited Files	<i>Describing Data</i>	Chapter 5, <i>Describing Sequential Data Sources</i>
DATASET in Master File	<i>Describing Data</i>	Chapter 2, <i>Identifying a Data Source</i>
Date-Time Data Type	<i>Describing Data</i>	Chapter 4, <i>Describing Individual Fields</i>
Comma Suppress Edit Option	<i>Describing Data</i>	Chapter 4, <i>Describing Individual Fields</i>

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
Percent Edit Option	<i>Describing Data</i>	Chapter 4, <i>Describing Individual Fields</i>
Using FILEDEF to Create Extract Files	<i>Overview and Operating Environments</i>	Chapter 4, <i>CMS Guide to Operations</i>

## Documentation Conventions

The following conventions apply throughout this manual:

<b>Convention</b>	<b>Description</b>
<code>THIS TYPEFACE</code>	Denotes a command that you must enter in uppercase, exactly as shown.
<i>this typeface</i>	Denotes a value that you must supply.
{ }	Indicates two choices. You must type one of these choices, not the braces.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices, not the symbol.
[ ]	Indicates optional parameters. None of them is required, but you may select one of them. Type only the information within the brackets, not the brackets.
<u>underscore</u>	Indicates the default value.
...	Indicates that you can enter a parameter multiple times. Type only the information, not the ellipsis points.
. : .	Indicates that there are (or could be) intervening or additional commands.

## Related Publications

See the *Information Builders Publications Catalog* for the most up-to-date listing and prices of technical publications, plus ordering information. To obtain a catalog, contact the Publications Order Department at (800) 969-4636.

You can also visit our World Wide Web site, <http://www.informationbuilders.com>, to view a current listing of our publications and to place an order.



# Customer Support

Do you have questions about FOCUS?

Call Information Builders Customer Support Service (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 a.m. and 8:00 p.m. EST to address all your FOCUS questions. Information Builders consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (*xxxx.xx*) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, <http://www.informationbuilders.com>. It connects you to the tracking system and known-problem repository at the Information Builders support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of [www.informationbuilders.com](http://www.informationbuilders.com) also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your Information Builders representative about InfoResponse, or call (800) 969-INFO.

## Information You Should Have

To help our consultants answer your questions most effectively, be ready to provide the following information when you call:

- Your six-digit site code number (*xxxx.xx*).
- The FOCEXEC procedure (preferably with line numbers).
- Master File with picture (provided by CHECK FILE).
- Run sheet (beginning at login, including call to FOCUS), containing the following information:
  - ? RELEASE
  - ? FDT
  - ? LET
  - ? LOAD
  - ? COMBINE
  - ? JOIN
  - ? DEFINE
  - ? STAT

- ? SET/? SET GRAPH
- ? USE
- ? TSO DDNAME OR CMS FILEDEF
- The exact nature of the problem:
  - Are the results or the format incorrect; are the text or calculations missing or misplaced?
  - The error message and code, if applicable.
  - Is this related to any other problem?
- Has the procedure or query ever worked in its present form? Has it been changed recently? How often does the problem occur?
- What release of the operating system are you using? Has it, FOCUS, your security system, or an interface system changed?
- Is this problem reproducible? If so, how?
- Have you tried to reproduce your problem in the simplest form possible? For example, if you are having problems joining two databases, have you tried executing a query containing just the code to access the database?
- Do you have a trace file?
- How is the problem affecting your business? Is it halting development or production? Do you just have questions about functionality or documentation?

## User Feedback

In an effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual. Please use the Reader Comments form at the end of this manual to relay suggestions for improving the publication or to alert us to corrections. You can also use the Document Enhancement Request Form on our Web site, <http://www.informationbuilders.com>.

Thank you, in advance, for your comments.

## Information Builders Consulting and Training

Interested in training? Information Builders Education Department offers a wide variety of training courses for this and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site (<http://www.informationbuilders.com>) or call (800) 969-INFO to speak to an Education Representative.

---

## CHAPTER 1

# Customizing Your Environment

### Topics:

- The SET Command
- SET Parameter Syntax

The SET command enables you to change parameters that govern your FOCUS environment. These parameters control output, work areas, the Hot Screen facility and other FOCUS features.

# The SET Command

The SET command enables you to customize both the application development and runtime environment. It controls the way that reports and graphs display on the screen or printer; the content of reports and graphs; data retrieval characteristics that affect performance; and system responses to end user requests.

## Syntax

### How to Set Parameters

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

where:

*parameter*

Is the FOCUS setting you wish to change.

*option*

Is one of a number of options available for each parameter.

You can set several parameters in one command by separating each with a comma.

You may include as many parameters as you can fit on one line. Repeat the SET keyword for each new line.

## Syntax

### How to Set Parameters in a Request

Many SET commands that change system defaults can be issued from within TABLE and GRAPH requests. SET used in this manner is temporary, affecting only the current request. The syntax is

```
ON {TABLE|GRAPH} SET parameter value [AND parameter value ...]
```

where:

*parameter*

Is the system default you wish to change.

*value*

Is an acceptable value that will replace the default value.

## Example      Setting Parameters in a Request

For example,

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY EMP_ID
ACROSS DEPARTMENT
ON TABLE SET NODATA NONE
END
```

changes the default NODATA character for missing data from a period to the word NONE.

SET commands that cannot be issued from within TABLE include ASNAMES, BINS, and HOLDATTR.

## SET Parameter Syntax

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

<b>Parameter:</b>	ACCBLN
Description:	Accepts blank or zero values for fields with ACCEPT commands in the Master File (see the <i>Describing Data</i> manual).
Syntax:	SET ACCBLN = { <u>ON</u>  OFF}
	where:
	<u>ON</u>
	Accepts blank and zero values for fields with ACCEPT commands unless blank or zero values are explicitly coded in the list of acceptable values. This value is the default.
	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.
<b>Parameter:</b>	AGGR[RATIO]
Description:	Determines the ratio of aggregation based on retrieved records and the final size of the answer set.
Syntax:	SET AGGR[RATIO] = { <i>n</i>   <u>9</u> }
	where:
	<i>n</i>
	Is the ratio of aggregation. The default value is 9.

**Parameter:** [ALL](#)

Description: Handles missing segment instances in a report.

Syntax: `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 (see NODATA).

[OFF](#)  
Omits missing segment instances from a report. This value is the default.

[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).

<b>Parameter:</b>	<code>ALLOWCVTERR</code>
Description:	<p>This 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).</p> <p>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.</p>
Syntax:	<p><code>SET ALLOWCVTERR = {ON OFF}</code></p> <p>where:</p> <p><u>ON</u></p> <p>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.</p> <ul style="list-style-type: none"><li>• If DATEDISPLAY and MISSING are set to OFF, a blank is returned.</li><li>• If DATEDISPLAY is set to OFF, and MISSING is set to ON, the value of the NODATA character (a period, by default) is returned (see NODATA).</li><li>• If DATEDISPLAY and MISSING are set to ON, the value of the NODATA character (a period, by default) is returned.</li><li>• If DATEDISPLAY is set to ON, and MISSING is set to OFF, the base date is returned (either December 31, 1900, for dates with YMD or YYMD format; or January 1901, for dates with YM, YYM, YQ, or YYQ format).</li></ul> <p><u>OFF</u></p> <p>Does not display a row of data that contains an invalid date format and generates an error message. This value is the default.</p>

<b>Parameter:</b>	<a href="#">ASNAMES</a>
Description:	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. Also controls how field names are specified for the values of an ACROSS field when a HOLD file is created.
Syntax:	<code>SET ASNAMES = {ON OFF <a href="#">FOCUS</a>}</code> where: <a href="#">ON</a> Uses the AS phrase for the field name, and controls the way ACROSS fields are named in HOLD files in any format. <a href="#">OFF</a> Does not use the AS phrase for the field name, or affect the way ACROSS fields are named. <a href="#">FOCUS</a> Uses the AS phrase for the field name, and controls the way ACROSS fields are named in HOLD files only. This value is the default.
<b>Parameter:</b>	<a href="#">AUTOINDEX</a>
Description:	Retrieves data faster by automatically taking advantage of indexed fields in most cases where TABLE requests contain equality or range tests on those fields. Applies only to FOCUS data sources.  AUTOINDEX is never performed when the TABLE request contains an alternate file view, for example, TABLE FILE <i>filename.filename</i> . Indexed retrieval is not performed when the TABLE request contains BY HIGHEST or BY LOWEST phrases and AUTOINDEX is ON.
Syntax:	<code>SET AUTOINDEX = {ON <a href="#">OFF</a>}</code> where: <a href="#">ON</a> Uses indexed retrieval when possible. <a href="#">OFF</a> Uses indexed retrieval only when explicitly specified via an indexed view, for example, TABLE FILE <i>filename.fieldname</i> . This value is the default.



<b>Parameter:</b>	AUTOPATH
Description:	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.
Syntax:	SET AUTOPATH = { <a href="#">ON</a>  OFF}
	where:
	<a href="#">ON</a> Dynamically selects an optimal retrieval path. This value is the default.
	<a href="#">OFF</a> Uses sequential data retrieval. The end user controls the retrieval path through <i>filename.segname</i> .
<b>Parameter:</b>	AUTOSTRATEGY
Description:	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.
Syntax:	SET AUTOSTRATEGY = { <a href="#">ON</a>  OFF}
	where:
	<a href="#">ON</a> Stops the search when a match is found. This value is the default.
	<a href="#">OFF</a> Searches the entire data source.
<b>Parameter:</b>	AUTOTABLEF
Description:	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.
Syntax:	SET AUTOTABLEF = { <a href="#">ON</a>  OFF}
	where:
	<a href="#">ON</a> Does not create an internal matrix. This value is the default.
	<a href="#">OFF</a> Creates an internal matrix.

<b>Parameter:</b>	<b>BINS</b>
Description:	Specifies the number of pages of core (blocks of 4,096 bytes) used for data source buffers. You can vary BINS from 13 to 63 pages depending on the size of your core. The default is roughly two-thirds of the core remaining after you start FOCUS.
Syntax:	<code>SET BINS = <i>n</i></code> where: <i>n</i> Is the number of core pages used for data source buffers. Valid values are 13 to 63.
<b>Parameter:</b>	<b>BLKCALC</b>
Description:	This parameter applies only to MVS.  Enables system-determined blocking for HOLD files written to DASD; files written to tape have BLKSIZE 32760, the operating-system maximum.  The SET BLKCALC command must be issued before the TABLE request and cannot be set within a request.
Syntax:	<code>SET BLKCALC = {<u>NEW</u> OLD}</code> where: <u>NEW</u> Calculates optimal blocking factors for both 3380 and 3390 device types. This value is the default.  <u>OLD</u> Uses the method of calculating BLKSIZE that was used prior to FOCUS Release 6.8.
<b>Parameter:</b>	<b>BUSDAYS</b>
Description:	Specifies which days are considered business days and which days are not if your business does not follow the traditional Monday through Friday week.
Syntax:	<code>SET BUSDAYS = <i>week</i></code> where: <i>week</i> Is SMTWTFS, representing the days of the week. Any day that you do not wish to designate as a business day must be replaced with an underscore in that day's designated place.

- Parameter:** `BYPANEL`
- Description:** This parameter applies only to HOTSCREEN.  
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.
- Syntax:** `SET BYPANEL = option`  
where:  
`option`  
Is one of the following:  
`ON` repeats BY field values on panels.  
`OFF` does not repeat field values on panels.  
`0` does not divide column between panels.  
`n` repeats *n* columns on each panel.
- Parameter:** `BYSCROLL`
- Description:** This parameter applies only to HOTSCREEN.  
Scrolls report headings and footers scroll along with the report contents.
- Syntax:** `SET {BYSCROLL|BYPANELSCROL} = {ON|OFF}`  
where:  
`ON`  
Scrolls report headings and footings along with report contents.  
`OFF`  
Does not scroll report headings and footings along with report contents.
- Parameter:** `BOTTOMMARGIN`
- Description:** This command applies to PostScript and PDF report formats.  
Sets the bottom boundary, in inches, of report contents on a page.
- Syntax:** `SET BOTTOMMARGIN = {n/.25}`  
where:  
`n`  
Is the bottom margin, in inches, for report contents on a page.  
The default is .25 inches.

**Parameter:** CACHE

Description:

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 (discussed in Chapter 8, *Euro Currency Support*, 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. The maximum number of cache pages can be set at installation. For more information, see Technical Memo 7838.1, *Setting the Maximum Number of Cache Pages*.

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

Syntax:

SET CACHE = {0|*n*}

where:

0

Allocates no space to cache; cache is inactive. This value is the default.

*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.

<b>Parameter:</b>	<code>CARTESIAN</code>
Description:	<p>Applies to requests containing PRINT or LIST.</p> <p>Generates a report containing all combinations of non-related data instances in the case of a multi-path request. ACROSS cancels this parameter.</p>
Syntax:	<p><code>SET CARTESIAN = {ON <u>OFF</u>}</code></p> <p>where:</p> <p><code>ON</code></p> <p>Generates a report with non-related records.</p> <p><u>OFF</u></p> <p>Disables the Cartesian product. This value is the default.</p>
<b>Parameter:</b>	<code>CDN</code>
Description:	<p>Specifies punctuation used in numerical notation. When set to ON, a comma marks the decimal position, and periods separate groups of three significant digits.</p> <p>Continental Decimal Notation (CDN) is supported for output in TABLE requests. It is not supported in DEFINE or COMPUTE commands.</p>
Syntax:	<p><code>SET {CDN EUROPE} = {ON <u>OFF</u>}</code></p> <p>where:</p> <p><code>ON</code></p> <p>Uses CDN. For example, the number 3,045,000.76 is represented as 3.045.000,76.</p> <p><u>OFF</u></p> <p>Turns CDN off. For example, the number 3,045,000.76 is represented as 3,045,000.76. This value is the default.</p>

<b>Parameter:</b>	<code>COLUMNSCROLL</code>
Description:	Enables you to scroll by column within the panels of a report provided that the report is wider than the screen width.
Syntax:	<code>SET COLUMNSCROLL = {ON OFF}</code> where: <code>ON</code> Enables column scrolling to the right and left by pressing the PF10 key and the PF11 key, respectively. To scroll up and down within the same column, use the PF7 key and the PF8 keys. <code>OFF</code> Disables column scrolling. This value is the default.
<b>Parameter:</b>	<code>COUNTWIDTH</code>
Description:	Expands the default format of COUNT fields from a 5-byte integer to a 9-byte integer.
Syntax:	<code>SET {COUNTWIDTH LISTWIDTH} = {ON OFF}</code> where: <code>ON</code> Expands the default format of COUNT fields from a five-byte integer to a nine-byte integer. <code>OFF</code> Does not expand the default format of COUNT fields from a five-byte integer to a nine-byte integer. This value is the default.
<b>Parameter:</b>	<code>COMPUTE</code>
Description:	Compiles all COMPUTE calculations in DEFINE commands and MODIFY requests into machine code at request time; uses this code to perform calculations at run time.
Syntax:	<code>SET COMPUTE = {NEW OLD}</code> where: <code>NEW</code> Specifies the new, compiled logic. NEW is the default. <code>OLD</code> Forces all calculations into the old logic until the FOCUS session is over, or the SET COMPUTE command is reset.

**Parameter:** `DATEDISPLAY`

**Description:** 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:

<b>Format</b>	<b>Base Date</b>
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.

**Syntax:** `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. This value is the default.

**Parameter:** `DATEFNS`

**Description:** Loads the year 2000-compliant versions of the FUSELIB subroutines.

**Syntax:** `SET DATEFNS = {ON|OFF}`

where:

`ON`

Loads the year 2000-compliant versions of the FUSELIB subroutines. This value is the default.

`OLD`

Uses non-year 2000-compliant subroutines.

**Parameter:** DATEFORMAT  
Description: 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 a value's input format independent of the format of the variable to which it is being assigned.

Syntax: SET DATEFORMAT = *datefmt*  
where:  
*datefmt*  
Can be one of the following: MDY, DMY, YMD, or MYD. The U.S. English default format is MDY.

**Parameter:** DATETIME  
Description: 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, etc. If DATETIME is not set, the behavior of the FOCUS date variables remain the same.

Syntax: SET DATETIME = *option*  
where:  
*option*  
Is one of the following:

<u>STARTUP</u>	Is the time and date when you began your FOCUS session. This setting is the default.
CURRENT NOW	Changes each time it is interrogated. For example, if your batch job starts before midnight at 11:59 P.M., it won't 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	Freezes the date and time of the current run for the rest of the session or until another SET DATETIME command is issued.



---

<b>Parameter:</b>	DEFCENT
Description:	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.)  <b>Note:</b> This same result can be achieved by including the FDEFCENT and FYRTHRESH attributes in the Master File.
Syntax:	SET DEFCENT = { <i>cc</i>   19}  where:  <i>cc</i> Is the default century. If you do not supply a value, <i>cc</i> defaults to 19, for the twentieth century.
<b>Parameter:</b>	DISPLAY
Description:	Is the PC display mode selection.
Syntax:	SET DISPLAY = { <u>OFF</u>   PCCOLOR   PCMONO}  where:  <i>option</i> Is one of the following:  <u>OFF</u> No display mode is selected. This value is the default.  PCCOLOR          The display mode is color.  PCMONO           The display mode is black and white.

<b>Parameter:</b>	<code>DTSTRICT</code>
Description:	Controls how much error checking is done on date-time values when they are input by users, read from an alphanumeric transaction file, displayed, or used in user-written subroutines.
Syntax:	<code>SET DTSTRICT = {<u>ON</u> OFF}</code> where: <u>ON</u> Invokes strict processing. This means that whenever a date-time value is input by a user, read from a transaction file, displayed, or returned by a subroutine it is checked to make sure that the value represents 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. ON is the default value. If you attempt to enter a value that violates this rule, the following message displays: <code>(FOC177) INVALID DATE CONSTANT: dt_constant</code> <u>OFF</u> Does not invoke strict processing. Any date-time component can have any value within the constraint of the number of decimal digits allowed; for example, the month value can be 00 or 13 or 99, but not 115. Furthermore, the values do not have to be consistent; for example, any month in any year can have 30 or 31 days.
<b>Parameter:</b>	<code>EMPTYREPORT</code>
Description:	Controls the output generated when a TABLE request retrieves zero records.  EMPTYREPORT is not supported with TABLEF. When a TABLEF request retrieves zero records, an empty report is always generated.
Syntax:	<code>SET EMPTYREPORT = {<u>ON</u> OFF}</code> where: <u>ON</u> Generates an empty report when zero records are found. <u>OFF</u> Does not generate an empty report when zero records are found. OFF is the default.

<b>Parameter:</b>	ESTRECORDS
<b>Description:</b>	<p>Passes the estimated number of records to be sorted in the request. FOCUS queries using external sorts and including the parameter 'FILSZ=En' can diminish FOC909 errors. This parameter enables the sorting algorithms to estimate SORTWORK space requirements for each sort parameter request.</p> <p>In order to make an accurate estimate for your ESTRECORDS setting, it is suggested that you run the report without an external sort in order to get a record count. If an attempt is made to SET ESTRECORDS from the FOCUS prompt, FOCPARAM, or PROFILE FOCEXEC the following error is generated:</p> <pre>SET ESTRECORDS = n (FOC36210) THE SPECIFIED PARAMETER CAN ONLY BE SET ON TABLE: ESTRECORDS</pre> <p>ESTRECORDS can only be set with the ON TABLE SET command within the TABLE, MATCH, or GRAPH request.</p> <p>For CMS/SyncSort the 'FILSZ=En' parameter is ignored. Therefore, SET ESTRECORDS <i>n</i> has no effect.</p>
<b>Syntax:</b>	<pre>SET ESTRECORDS = n</pre> <p>where:</p> <p><i>n</i></p> <p>Is the estimated number of records to be sorted.</p>

- Parameter:** `EUROFILE`
- Description: 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.
- Syntax: `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.
- Parameter:** `EXTAGGR`
- Description: Uses external sorts to perform aggregation.
- Syntax: `SET EXTAGGR = {ON|OFF|NOFLOAT}`
- where:
- ON
- Uses external sorts to perform aggregation.
- `OFF`
- Does not allow aggregation by an external sort.
- `NOFLOAT`
- Allows aggregation if there are no floating data fields present.
- Parameter:** `EXTHOLD`
- Description: Enables you to create HOLD files using an external sort.
- Syntax: `SET EXTHOLD = {ON|OFF}`
- where:
- `ON`
- Creates HOLD files using an external sort.
- OFF
- Does not create HOLD files using an external sort.

<b>Parameter:</b>	<code>EXTSORT</code>
Description:	<p>Calls an external sort for use with the TABLE, MATCH, and GRAPH commands.</p> <p>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. For additional information, see the <i>Creating Reports</i> manual.</p>
Syntax:	<pre>SET EXTSORT = {ON OFF}</pre> <p>where:</p> <p><u>ON</u> Enables FOCUS to pass records that are retrieved to an external sort. This value is the default.</p> <p><u>OFF</u> Uses the FOCUS internal sorting procedure.</p>
<b>Parameter:</b>	<code>EXTTERM</code>
Description:	Enables the use of extended terminal attributes.
Syntax:	<pre>SET EXTTERM = {ON OFF}</pre> <p>where:</p> <p><u>ON</u> Enables the use of attributes. This value is the default.</p> <p><u>OFF</u> Disables the use of attributes.</p>
<b>Parameter:</b>	<code>FIELDNAME</code>
Description:	Controls the use of long field names (66 characters).
Syntax:	<pre>SET FIELDNAME = {NEW NOTRUNC OLD}</pre> <p>where:</p> <p><u>NEW</u> Supports long field names.</p> <p><u>NOTRUNC</u> Does not support unique truncations.</p> <p><u>OLD</u> Turns off support for long field names.</p>

<b>Parameter:</b>	<code>FILE[NAME]</code>
Description:	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.
Syntax:	<code>SET FILE[NAME] = filename</code> where: <code>filename</code> Is a default file to be used in commands.
<b>Parameter:</b>	<code>FILTER</code>
Description:	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.
Syntax:	<code>SET FILTER = {*_filter [filter]} IN file {ON OFF}</code> where: <u>*</u> Denotes all declared filters. This value is the default. <code>ON</code> Activates the filter. The maximum number of filters set ON for a file is limited by the number of IF/WHERE commands in these filters and should not exceed the standard FOCUS limit of IF/WHERE commands in any single TABLE request. <code>OFF</code> Deactivates the filter. This value is the default. <code>filter</code> Is the name of a filter as declared in the NAME = syntax of the FILTER FILE block.

<b>Parameter:</b>	<code>FIXRETRIEVE</code>
Description:	FOCUS HOLD files support keyed retrieval, which can greatly reduce the IOs incurred in reading extract files. The performance gains are accomplished by using the <code>SEGTYPE=</code> parameter in the Master File as a logical key for sequential files. It 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.
Syntax:	<code>SET {FIXRETRIVE FIXF} = {<u>ON</u> OFF}</code>
	where:
	<u>ON</u> Stops the retrieval process when an equality test on this field holds true.
	<u>OFF</u> Does not stop the retrieval process when an equality test on this field holds true.
<b>Parameter:</b>	<code>FOC144</code>
Description:	Tells FOCUS to suppress warning message FOC144, which reads: "Warning: Testing in Independent sets of Data."
Syntax:	<code>SET FOC144 = {<u>NEW</u> OLD}</code>
	where:
	<u>NEW</u> Displays the FOC144 warning message. This value is the default.
	<u>OLD</u> Suppresses the FOC144 warning message.

<b>Parameter:</b>	<code>FOC2GIGDB</code>
Description:	Enables two-gigabyte FOCUS data sources. Must be set in the FOCPARM profile.
Syntax:	<code>SET FOC2GIGDB = {ON OFF}</code> where: <code>ON</code> Enables support for FOCUS data sources larger than one-gigabyte. Note that an attempt to use FOCUS data sources larger than one-gigabyte in a release prior to FOCUS Version 7.1 can cause database corruption. <code>OFF</code> Disables support for FOCUS data sources larger than one-gigabyte. OFF is the default value.
<b>Parameter:</b>	<code>FOCALLOC</code>
Description:	This parameter applies only to MVS. Automatically allocates of FOCUS files. Allocation is done based on Prefix.Master. FOCUS. The DISP will be SHR.
Syntax:	<code>SET {FOCALLOC FALLOC} = {ON OFF}</code> where: <code>ON</code> Automatically allocates FOCUS files. <code>OFF</code> Does not automatically allocate FOCUS files. This value is the default.
<b>Parameter:</b>	<code>FOCSTACK</code>
Description:	Is the amount of core in thousands of bytes used by FOCSTACK, the stack of FOCUS commands from FOCEXECs awaiting execution. The maximum value of FOCSTACK depends on your current region size. You can also specify the parameter as FOCSTACK SIZE.
Syntax:	<code>SET FOCSTACK = {n 8}</code> where: <code>n</code> Is the amount of core in thousands of bytes used by FOCSTACK. The default value is 8.



<b>Parameter:</b>	<code>HDAY</code>
Description:	Activates the holiday file that is used in conjunction with the data functions DATEDEIF, DATEMOV, DATECVT, and DATEADD.  This setting is by default not set at all.
Syntax:	<code>SET HDAY = <i>string</i></code>  where:  <code><i>string</i></code> Is the part of the name of the holiday file after HDAY. This string must be four characters long.
<b>Parameter:</b>	<code>HLISUTRACE</code>
Description:	Used for debugging, records the last 20 events that the FOCUS Database Server (formerly called the sink machine) performed. The information is written to memory and is intended for use when reading a dump of the SU address space. This setting may only be set in the SU profile, HLIPROF.
Syntax:	<code>SET HLISUTRACE = {<u>ON</u> OFF}</code>  where:  <code><u>ON</u></code> Records the last 20 events that the FOCUS Database Server performed. This value is the default.  <code>OFF</code> Does not record the last 20 events that the FOCUS Database Server performed.
<b>Parameter:</b>	<code>HLISUDUMP</code>
Description:	This setting is only used for debugging FOCUS Database Server problems and may only be set in the SU profile, HLIPROF.
Syntax:	<code>SET HLISUDUMP = <i>n</i></code>  where:  <code><i>n</i></code> When set to 99999, a dump of the FOCUS Database Server address space will occur for any error on the server. The user abend code is set to 275. The user code will also be set to the error number.

<b>Parameter:</b>	<code>HOLDATTR[S]</code>
Description:	Includes the TITLE and ACCEPT attributes from the original Master File in the HOLD Master File. This setting does not affect the way fields are named in the HOLD Master File.
Syntax:	<code>SET HOLDATTR = {ON OFF <a href="#">FOCUS</a>}</code> where: <code>ON</code> Includes the TITLE attribute from the original Master File in HOLD Master Files for HOLD files of any format. The ACCEPT attribute is included in the HOLD Master File when the HOLD file is in FOCUS format. <code>OFF</code> Does not include the TITLE or ACCEPT attributes from the original Master File in the HOLD Master File. <code><a href="#">FOCUS</a></code> Includes the TITLE and ACCEPT attributes in HOLD Master Files when the HOLD file is in FOCUS format. This value is the default.
<b>Parameter:</b>	<code>HOLDLIST</code>
Description:	Determines what fields in a report request are included in the HOLD file.
Syntax:	<code>SET HOLDLIST = {PRINTONLY <a href="#">ALL</a>}</code> where: <code>PRINTONLY</code> Includes only those fields in the HOLD file that are specified in the report request. <code><a href="#">ALL</a></code> Includes all verb object fields referenced in a request in the HOLD file, including both computed fields and fields referenced in a COMPUTE command. This value is the default. (OLD may be used as a synonym for ALL.)

<b>Parameter:</b>	<a href="#">HOLDSTAT</a>
Description:	Includes the comments and DBA information in HOLD Master Files. This information is found in the HOLDSTAT ERRORS file supplied by Information Builders, or in a user-specified file.
Syntax:	<code>SET HOLDSTAT = {ON <a href="#">OFF</a> <i>name</i>}</code> where: <a href="#">ON</a> Derives comments and DBA information from the holdstat.mas or errors.mas file in UNIX and NT. In MVS, this information is derived from the member HOLDSTAT in the PDS allocated to the ddname MASTER or ERRORS in MVS. <a href="#">OFF</a> Does not include information from the HOLDSTAT file in the HOLD Master File. This value is the default. <i>name</i> Specifies a HOLDSTAT file, created by the end user, whose information is included in the HOLD Master File.
<b>Parameter:</b>	<a href="#">HOTMENU</a>
Description:	Automatically displays the Hot Screen PF key legend at the bottom of the Hot Screen report.
Syntax:	<code>SET HOTMENU = {ON <a href="#">OFF</a>}</code> where: <a href="#">ON</a> Displays the PF key legend. <a href="#">OFF</a> Does not display the PF key legend. To see the PF key legend, the user must press PF1. OFF is the default.

<b>Parameter:</b>	<code>IBMLE</code>
Description:	Determines whether LE preinitialization is on or off.
Syntax:	<code>SET IBMLE = {<u>OFF</u> ON}</code>
	where:
	<u>OFF</u> Does not invoke preinitialization. This value is the default and is required for C++ and FORTRAN subroutines. It is the recommended setting for COBOL subroutines that should use the COBOL option RTEREUS (ON).
	<u>ON</u> Invokes preinitialization and is a requirement for PL/I and C subroutines.  Note: Mixed-mode applications calling both LE and non-LE subroutines in the same FOCEXEC or FOCUS session are not supported and may produce unpredictable results.
<b>Parameter:</b>	<code>IMMEDTYPE</code>
Description:	Used with TOE, tells FOCUS where to send line mode output.
Syntax:	<code>SET IMMEDIATE = {ON <u>OFF</u>}</code>
	where:
	<u>ON</u> Sends all line mode output, such as -TYPE, to the Output Window as it is executed, line by line.
	<u>OFF</u> Buffers all line mode output. The output appears in the Output Window as a new full screen. This value is the default.
<b>Parameter:</b>	<code>IMS</code>
Description:	Tells FOCUS whether to use the new version of the IMS interface. The new version is for releases after 6.8 PUT level 9406.
Syntax:	<code>SET IMS = {<u>NEW</u> OLD}</code>
	where:
	<u>NEW</u> Is the new version of the IMS interface. This value is the default.
	<u>OLD</u> Is the version of IMS interface prior to release 6.8 PUT level 9406.

**Parameter:** [INDEX](#)

Description: The indexing scheme used for indexes (fields specified with FIELDTYPE=I keywords in the Master Files).

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

where:

[NEW](#)

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

[OLD](#)

Creates a hash index.

**Parameter:** [JOINOPT](#)

Description: Allows the joining of two files that contain different numeric data types.

Syntax: `SET JOINOPT = {NEW|OLD}`

where:

[NEW](#)

Allows the joining of files that contain different numeric data types.

[OLD](#)

Does not allow the joining of files that contain different numeric data types.

**Parameter:** LANG[UAGE]  
**Description:** Specifies the National Language Support (NLS) environment.  
**Syntax:** SET LANG[UAGE] = value

where:

value

Is a language from the following list. The ID, name, or abbreviation can be used to specify the language.

These TERM values support DBCS. The default value for TERM is IBM3270, which does not support DBCS.

ID	Name	Abbreviation
1	ENGLISH	AME
1	AMENGLISH	AME
20	ARABIC	ARB
359	BULGARIAN	BLG
416	CANFRENCH	CFR
34	CATALAN	CAT
85	S-CHINESE	PRC
86	T-CHINESE	ROC
45	DANISH	DAN
31	DUTCH	DUT
358	FINNISH	FIN
32	FLEMISH	FLM
33	FRENCH	FRE
49	GERMAN	GER
30	GREEK	GRE
972	HEBREW	HEB
91	HINDI	IND
36	HUNGARIAN	HUN
354	ICELANDIC	ICL
39	ITALIAN	ITA
81	JAPANESE	JPN
10081	JAPANESE-E*	JPN
82	KOREAN	KOR

ID	Name	Abbreviation
47	NORWEGIAN	NOR
48	POLISH	POL
351	PORTUGUESE	POR
7	RUSSIAN	RUS
38	SLOVENIAN	SLO
34	SPANISH	SPA
46	SWEDISH	SWE
66	THAI	THA
90	TURKISH	TUR
44	UKENGLISH	UKE

\*To specify JAPANESE-E, you can use the ID or the full name, but not the abbreviation JPN.

In addition, when you select JAPANESE-E, make sure that the TERM parameter is set to a value that supports DBCS. See TERM for values that support DBCS.

**Parameter:**

LEADZERO

**Description:**

Leading zeros are truncated in Dialogue Manager strings. The subroutines 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.

**Syntax:**

SET LEADZERO = {ON|OFF}

where:

ON

Allows the display of leading zeros if they are present.

OFF

Truncates leading zeros if they are present.

<b>Parameter:</b>	<code>LEFTMARGIN</code>
Description:	This parameter applies only to PostScript and PDF formats. Sets the left boundary for report contents on a page.
Syntax:	<code>SET LEFTMARGIN = {value .250}</code> where: <code>value</code> Is the left boundary of report contents on a page. The default is .25 inches.
<b>Parameter:</b>	<code>LINES</code>
Description:	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. 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. The value of LINES can range between 1 and 999999; specify 999999 for continuous forms.  <b>Note:</b> When you use 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.
Syntax:	<code>SET LINES = {n 57}</code> where: <code>n</code> Is the maximum number of lines of printed output that appear on a page. The default value is 57.



<b>Parameter:</b>	MASTER
Description:	This parameter applies only to the FUSION option. New Master Files pass for blank delimited Master Files, which use the new FUSION syntax.
Syntax:	SET MASTER = {NEW OLD} where: NEW Passes a new Master File for a blank delimited Master File, which uses the new FUSION syntax. OLD Does not pass a new Master File for a blank delimited Master File. This value is the default.
<b>Parameter:</b>	MAXLRECL
Description:	Defines the maximum record length for an external file with OCCURS segments. The default is 0. However, FOCUS can read a 16K recl by default. This may be set to a maximum of 32K.
Syntax:	SET MAXLRECL = {n 0} where: n Is the maximum record length for an external file with OCCURS segments. The default value is 0.
<b>Parameter:</b>	MESSAGE
Description:	Controls the display of informational messages.
Syntax:	SET {MESSAGE MSG} = {ON OFF} where: ON Displays informational messages. This value is the default. OFF Suppresses both informational messages and carets that appear when FOCUS executes commands in procedures. FOCUS still displays error messages, and the carets that prompt for input.

<b>Parameter:</b>	<code>MINIO</code>
Description:	<p>This parameter applies only to MVS.</p> <p>Improves performance by reducing I/O operations up to fifty percent when accessing FOCUS data sources under MVS. This is a buffering technique.</p> <p>With FOCUS data sources that are not disorganized, MINIO can greatly reduce the number of I/O operations for TABLE and MODIFY commands. The actual I/O reduction will vary depending on data source structure and average number of children segments per parent segment. By reducing I/O operations, elapsed time for TABLE and MODIFY commands also drop.</p>
Syntax:	<p><code>SET MINIO = {<u>ON</u> OFF}</code></p> <p>where:</p> <p><u>ON</u></p> <p>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.</p> <p><code>OFF</code></p> <p>Disables MINIO.</p>

<b>Parameter:</b>	MULTIPATH
Description:	Controls testing on independent paths.
Syntax:	SET MULTIPATH = {SIMPLE COMPOUND}
	where:
	SIMPLE
	Includes a parent segment in the report output if: <ul style="list-style-type: none"> <li>• It has at least one child that passes its screening conditions.</li> <li>• It lacks any referenced child on a path, but the child is optional (see the <i>Creating Reports</i> manual).</li> </ul> SIMPLE is the default value for FOCUS for S/390. The (FOC144) warning message is generated when a request screens data in a multi-path report. <p>(FOC144) WARNING. TESTING IN INDEPENDENT SETS OF DATA:</p> COMPOUND
	Includes a parent in the report output if it has <i>all</i> of its required children (see the <i>Creating Reports</i> manual). The COMPOUND setting does not generate the (FOC144) warning message. COMPOUND is the default value for EDA and WebFOCUS.
	The segment rule is applied level by level as FOCUS descends the data source/view hierarchy. That is, a parent segment's existence depends on the child segment's existence and the child segment depends on the grandchild's existence, and so on for the full data source tree.
<b>Parameter:</b>	NODATA
Description:	Determines the character string that indicates missing data in a report. The NODATA parameter can be abbreviated to NA.
Syntax:	SET {NODATA NA} = { <i>string</i>  .}
	where:
	<i>string</i>
	Is the character string that indicates missing data in reports. The default is a period.

<b>Parameter:</b>	<code>ONLINE-FMT</code>
Description:	Determines the format of report output. StyleSheet reports are generated in PostScript format. Styled reports can only be printed on a PostScript printer.
Syntax:	<code>SET ONLINE-FMT = {<a href="#">STANDARD</a>   <a href="#">POSTSCRIPT</a>}</code> where: <a href="#">STANDARD</a> Produces the report as un-styled character-based output. This value is the default. <a href="#">POSTSCRIPT</a> Saves the report output to a PostScript file with the name PSOUT. In MVS, the PostScript formatted report output is in a variable length PDS allocated to the ddname PS. In CMS, the output is in a file with the file type PS. The parameters set with the SET STYLESHEET command are in effect. PS can be used as a synonym for POSTSCRIPT.
<b>Parameter:</b>	<code>ORIENTATION</code>
Description:	This parameter applies to PostScript and PDF report formats. Specifies the page orientation for styled reports.
Syntax:	<code>SET ORIENTATION = {<a href="#">PORTRAIT</a>   <a href="#">LANDSCAPE</a>}</code> where: <a href="#">PORTRAIT</a> Displays the page in portrait style. This value is the default. <a href="#">LANDSCAPE</a> Displays the page in landscape style.

**Parameter:** `PAGE[-NUM]`

**Description:** Controls the numbering of output pages.

**Syntax:** `SET PAGE[-NUM] = option`

*option*

Is one of the following:

- `ON` Displays the page number on the upper left-hand corner of the page. This value is the default.
- `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.
- `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.

**Parameter:** `PAGESIZE`

**Description:** Specifies the page size for printed output. For optimal report appearance, the actual paper size must match your setting for `PAGESIZE`. If it does not, your report or your report will be cropped or contain extra blank spaces.

**Syntax:** `SET PAGESIZE = size`

where:

*size*

Specifies the page size. If the actual paper size does not match the `PAGESIZE` setting, your report will either be cropped or contain extra blank space. The 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-11	Sets the page size to 4.5 x 10.375 inches.
ENVELOPE-12	Sets the page size to 4.5 x 11 inches.
ENVELOPE-14	Sets the page size to 5 x 11.5 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.
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.
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-C6	Sets the page size to 4.5 x 6.375 inches.
ENVELOPE-C65	Sets the page size to 4.5 x 9 inches.
A5	Sets the page size to 5.8 x 8.25 inches.
ENVELOPE-C5	Sets the page size to 6.4 x 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.
B5	Sets the page size to 7.2 x 10.1 inches.
A4	Sets the page size to 8.25 x 11.7 inches.
QUARTO	Sets the page size to 8.5 x 10.8 inches.
ENVELOPE-C4	Sets the page size to 9 x 12.75 inches.
ENVELOPE-B4	Sets the page size to 9.8 x 13.9 inches.
B4	Sets the page size to 9.8 x 13.9 inches.
A3	Sets the page size to 11.7 x 16.8 inches.
ENVELOPE-C3	Sets the page size to 12.75 x 18 inches.

**Parameter:** `PANEL`

Description:

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 `BY``PANEL` parameter is `OFF`, a report can be divided into a maximum of four panels. If `SET` `BY``PANEL` has a value other than `OFF`, the report may be divided into 99 panels.

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

Syntax:

`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. This value is the default.



<b>Parameter:</b>	PAPER
Description:	<p>Specifies the physical length of the paper, in lines, for printed output. You derive this value by multiplying the length of the paper, in inches, by the number of lines printed per inch. For example, if your printer prints six lines per inch on standard 11 inch forms, PAPER should be set to 66. If you are placing a footing at the bottom of the page, this value should be less; in this case, 62. Valid values for PAPER are numbers between 1 and 999999. Specify 999999 for continuous forms.</p> <p><b>Note:</b> When the STYLESHEET parameter is in effect, the setting for PAPER is ignored.</p>
Syntax:	<pre>SET PAPER = {n 66}</pre> <p>where:</p> <p><i>n</i></p> <p>Is the length of paper, in lines, for printed output. Valid values are numbers between 1 and 999999. The value 999999 denotes the use of continuous forms. The default value is 66.</p>
<b>Parameter:</b>	PASS
Description:	<p>Enables user access to a data source or stored procedure protected by Information Builders security.</p>
Syntax:	<pre>SET PASS = password [IN filename]</pre> <p>where:</p> <p><i>password</i></p> <p>Is the password that allows access to data sources protected by Information Builders database security.</p> <p><i>filename</i></p> <p>Is the FOCUS data source or stored procedure protected by security.</p>

<b>Parameter:</b>	<code>PAUSE</code>
Description:	<p>Pauses before displaying a FOCUS report on the terminal. When you use a printing terminal, this parameter allows you to adjust the paper before printing the report.</p> <p>When the SCREEN parameter is ON, the PAUSE parameter is set ON (until you set the PAUSE parameter to OFF). If you set the SCREEN parameter to OFF, the PAUSE parameter is set to OFF. Note that you can change the PAUSE parameter without affecting the SCREEN parameter.</p> <p>This setting does not affect offline printing (routing output to a system printer).</p>
Syntax:	<pre>SET PAUSE = {ON OFF}</pre> <p>where:</p> <p><code>ON</code> Pauses before displaying a FOCUS report. This value is the default.</p> <p><code>OFF</code> Does not pause before displaying a FOCUS report.</p>
<b>Parameter:</b>	<code>PFnn</code>
Description:	<p>Assigns a function to the PF key specified by <i>nn</i>, enabling you to change the current PF key setting when using FIDEL (and also, under certain conditions, within the Window facility).</p> <p>The current settings are displayed by the ? PFKEY command.</p>
Syntax:	<pre>SET PFnn = function</pre> <p>where:</p> <p><i>nn</i> Is the PF key you are assigning a function to.</p> <p><i>function</i> Is the function to assign to the PF key specified by PFnn.</p>

<b>Parameter:</b>	<code>PREFIX</code>
Description:	<p>This parameter applies only to MVS.</p> <p>Specifies the prefix of existing data sets automatically allocated by FOCUS.</p>
Syntax:	<p><code>SET PREFIX = <i>prefix</i></code></p> <p>where:</p> <p><i>prefix</i></p> <p>Specifies of the prefix of existing data sets automatically allocated by FOCUS. The default setting in TSO is your user ID; the default setting in batch is FOCUS.</p>
<b>Parameter:</b>	<code>PRINT</code>
Description:	<p>Specifies the report output destination.</p> <p>You can enter ONLINE and OFFLINE as separate commands that have the same effect as specifying ONLINE and OFFLINE as PRINT settings.</p>
Syntax:	<p><code>SET PRINT = {<u>ONLINE</u>   <u>OFFLINE</u>}</code></p> <p>where:</p> <p><u>ONLINE</u></p> <p>Prints report output to the terminal.</p> <p><u>OFFLINE</u></p> <p>Prints report output to the system printer.</p>

<b>Parameter:</b>	<code>PRINTPLUS</code>
Description:	<p>Introduces enhancements to the display alternatives offered by the FOCUS Report Writer. To force a break at a specific spot, you must use NOSPLIT.</p> <p><code>PRINTPLUS</code> is not supported with StyleSheets. Problems may be encountered if <code>HOTSCREEN</code> is set to <code>OFFLINE</code>.</p>
Syntax:	<p><code>SET {PRINTPLUS PRTPLUS} = {ON OFF}</code></p> <p>where:</p> <p><u>ON</u></p> <p>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. ON is the default.</p> <p><u>OFF</u></p> <p>Does not support StyleSheets.</p>

**Parameter:** `QUALCHAR`

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

**Syntax:** `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)

The period is the default character. The use of the other qualifying characters listed above is restricted; they 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.

In VM, if the TERM tabchar is ON or if the CMS INPUT command includes the broken vertical bar (hex 6A), then the broken vertical bar cannot be the qualifying character. To query INPUT, type Q INPUT at the CMS prompt.

**Parameter:** `QUALTITLES`

**Description:** 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.

**Syntax:** `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. This value is the default.

- Parameter:** `REBUILDMSG`
- Description: Allows for direct control over the frequency with which REBUILD issues messages.
- Syntax: `SET {REBUILDMSG|REMSG} = n`
- where:
- n*  
Is any number.
- Parameter:** `RECAP-COUNT`
- Description: Includes lines containing a value created with RECAP when counting the number of lines per page for printed output. The number of lines per page is determined by the LINES parameter.
- Syntax: `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.  
This value is the default.
- Parameter:** `RECORDLIMIT`
- Description: Limits the number of records retrieved.
- Syntax: `SET RECORDLIMIT = {n|RECORDLIMIT}`
- where:
- n*  
Is the maximum number of records to be retrieved.
- RECORDLIMIT  
Respects explicit RECORDLIMIT values only.

<b>Parameter:</b>	<code>RIGHTMARGIN</code>
Description:	This parameter applies to PostScript and PDF report formats. Sets the right boundary for report contents on a page.
Syntax:	<code>SET RIGHTMARGIN = {value .25}</code> where: <code>value</code> Is the right boundary of report contents on a page. The default value is .25 inches.
<b>Parameter:</b>	<code>RPAGESET</code>
Description:	Controls how the number of lines per printed page is determined when output contains text created with SUBFOOT and a field value created with RECAP.
Syntax:	<code>SET RPAGESET = {NEW OLD}</code> where: <code>NEW</code> Sets the number of lines per page equal to the LINES value plus two plus the number of the highest BY field with a SUBFOOT. <code>OLD</code> Works as in release 6.0. OLD is the default.
<b>Parameter:</b>	<code>SAVEMATRIX</code>
Description:	Preserves the internal matrix and keeps it available for subsequent RETYPE, HOLD, SAVE, SAVB, and REPLOT commands when followed by Dialog Manager commands.
Syntax:	<code>SET SAVEMATRIX = {ON OFF}</code> where: <code>ON</code> Saves the last internal matrix generated. This value is the default. <code>OFF</code> Does not guarantee that the internal matrix will be available.

<b>Parameter:</b>	<code>SBORDER</code>
Description:	<p>Generates a solid border on the screen for full-screen mode.</p> <p>If the screen appears to be generated incorrectly, it is possible that the terminal does not support this new feature; change the setting to OFF to correct the situation.</p> <p>The amper variable <code>&amp;FOCSBORDER</code> contains the value of the <code>SBORDER</code> setting. <code>&amp;FOCSBORDER</code> may be included in the Dialogue Manager <code>-TYPE</code> command.</p>
Syntax:	<p><code>SET SBORDER = {<a href="#">ON</a> OFF}</code></p> <p>where:</p> <p><a href="#">ON</a> Enables solid borders. This value is the default.</p> <p><code>OFF</code> Enables dashed (nonsolid) borders.</p>
<b>Parameter:</b>	<code>SCREEN</code>
Description:	<p>Selects the Hot Screen facility.</p> <p>When the <code>SCREEN</code> parameter is ON, the <code>PAUSE</code> parameter is set ON (until you set the <code>PAUSE</code> parameter OFF). If you set the <code>SCREEN</code> parameter OFF, the <code>PAUSE</code> parameter is set OFF. Note that you can change the <code>PAUSE</code> parameter without affecting the <code>SCREEN</code> parameter.</p>
Syntax:	<p><code>SET SCREEN = {<a href="#">ON</a> OFF <a href="#">PAPER</a>}</code></p> <p>where:</p> <p><a href="#">ON</a> Activates the Hot Screen facility. This value is the default.</p> <p><code>OFF</code> Deactivates the Hot Screen facility.</p> <p><a href="#">PAPER</a> Activates the Hot Screen facility and causes <code>FOCUS</code> to use the settings for <code>LINES</code> and <code>PAPER</code> parameters to format screen display.</p>



**Parameter:** SHADOW

Description: Activates the Absolute File Integrity feature.

Syntax: SET SHADOW [PAGE] = {ON|OFF|OLD}

where:

ON

Activates the Absolute File Integrity feature. The maximum number of pages shadowed is 256K.

OFF

Deactivates the Absolute File Integrity feature. OFF is the default.

OLD

Indicates that your FOCUS file was created before Release 7.0. This means that the maximum number of pages shadowed is 63,551.

**Parameter:** SHIFT

Description: Controls the use of “shift” strings.

Syntax: 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.

- Parameter:** `SortLIB`
- Description: This parameter applies only to FOCUS VM/CMS.  
Tells FOCUS which sort package is installed at your site.
- Syntax: `SET SortLIB = option`  
where:  
`option`  
Is one of the following:
- |                          |  |
|--------------------------|--|
| <code>VMSort</code>      | Is the VMSort sort package.  |
| <code>SYNCSort</code>    | Is the SYNCSort sort package.  |
| <code>DFSORT</code>      | Is the DFSORT sort package.  |
| <code>SITeDEFINeD</code> | Use SITeDEFINeD if not VMSort, SYNCSort, or DFSORT. This sort package must be installed in SortLIB TXtLIB in order for FOCUS to find it. |
- Parameter:** `SPACES`
- Description: Sets the number of spaces between columns in a report.
- Syntax: `SET SPACES = {AUTO|n}`  
where:  
AUTO  
Automatically places either one or two spaces between columns.  
This value is the default.
- `n`  
Is the number of spaces to place between columns of a report.  
Valid values are integers between 1 and 8.

<b>Parameter:</b>	<code>SQLTOPTTF</code>
Description:	Enables the SQL Translator to generate TABLEF commands instead of TABLE commands.
Syntax:	<code>SET SQLTOPTTF = {ON OFF}</code> where: <code>ON</code> Generates TABLEF commands when possible. For example, a TABLEREF command is generated if there is no JOIN or GROUP BY command. <code>OFF</code> Always generates TABLE commands. This value is the default.
<b>Parameter:</b>	<code>SQUEEZE</code>
Description:	This parameter applies only to the StyleSheet feature. 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.
Syntax:	<code>SET SQUEEZE = {ON OFF}</code> where: <code>ON</code> Assigns column widths based on the widest data value or widest column title, whichever is longer. This value is the default. <code>OFF</code> 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.

<b>Parameter:</b>	<code>STYLE[SHEET]</code>
Description:	Controls the format of report output by accepting or rejecting StyleSheet parameters. These parameters specify formatting options such as page size, orientation, and margins.
Syntax:	<code>SET STYLE[SHEET] = {stylesheet ON OFF}</code>
	where:
	<i>stylesheet</i>
	Is the name of the StyleSheet file. For UNIX and NT, this is the name of the StyleSheet file without the file extension .sty. For MVS, this is the member name in the PDS allocated to ddname FOCSTYLE by a DYNAM command.
	For a PDF or PostScript report, FOCUS 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.
	<u>ON</u>
	Creates an HTML table using the default proportional font defined in the end user's browser. This value is the default.
	For a PDF or PostScript report, FOCUS 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.
	<u>OFF</u>
	Creates a pre-formatted report using the default fixed font defined in the end user's browser.
	For a PDF or Postscript report, FOCUS uses the settings for LINES, PAPER, PANEL, and WIDTH; the settings for UNITS, TOPMARGIN, BOTTOMMARGIN, LEFTMARGIN, RIGHTMARGIN, PAGESIZE, ORIENTATION, and SQUEEZE are ignored.

---

<b>Parameter:</b>	<code>SUMPREFIX</code>
Description:	When an external sort product performs aggregation of alphanumeric or smart date formats, the order of the answer set returned differs from the order of the FOCUS sorted answer sets.
Syntax:	<code>SET SUMPREFIX = {<u>FST</u> <u>LST</u>}</code> where: <code>FST</code> Displays the first value in cases of data aggregation of alphanumeric or smart date data types. <code>LST</code> Displays the last value in cases of data aggregation of alphanumeric or smart date data types.
<b>Parameter:</b>	<code>SUSI</code>
Description:	See the <i>Simultaneous Usage Reference Manual for TSO</i> .
<b>Parameter:</b>	<code>SUTABSIZE</code>
Description:	See the <i>Simultaneous Usage Reference Manual for TSO</i> .
<b>Parameter:</b>	<code>TEMP[DISK]</code>
Description:	This parameter applied only to CMS. Determines the disk FOCUS uses for temporary work space, and to store extract files (HOLD and SAVE).
Syntax:	<code>SET TEMP[DISK] = <i>disk</i></code> where: <code><i>disk</i></code> Is the disk FOCUS uses for temporary workspace, and to store extract files.

**Parameter:** `TERM`

Description: Selects the terminal type.

Syntax: `SET TERM[INAL] = {type | IBM3270}`

where:

*type*

Is the terminal type. The options are:

[IBM3270](#) Is the default value. It does not support DBCS.

[IBM5550](#) Specifies an IBM 5550 or a PS/55 terminal. Supports DBCS.

[F6650](#) Specifies a Facom F-6650 terminal. Supports DBCS.

[H56020](#) Specifies a Hitachi H-560/20 terminal. Supports DBCS.

**Parameter:** `TESTDATE`

Description: Temporarily alters the system date in order to test a dynamic window. That is, it allows you to simulate clock settings beyond the year 1999 to determine the behavior of your program.

Only use TESTDATE for testing purposes with test data. The value of TESTDATE affects all reserved variables that retrieve the current date from the system. Setting TESTDATE also affects anywhere in FOCUS that a date is used (such as CREATE, MODIFY, MAINTAIN) but does not affect the date referenced directly from the system.

TESTDATE can either be equal to TODAY or a date in the format YYYYMMDD. If anything else is entered the following message is displayed:

```
TESTDATE MUST BE YYYYMMDD OR TODAY
```

Syntax: `SET TESTDATE = {yyyymmdd | TODAY}`

where:

*yyyymmdd*

Is an 8-digit date in the format YYYYMMDD.

[TODAY](#)

Is the current date. This value is the default.

<b>Parameter:</b>	TEXTFIELD
Description:	Preserves downward compatibility with prior FOCUS releases. FOCUS text fields have been enhanced significantly with Release 7.0.  <b>Note:</b> FOCUS Release 7.0 preserves text fields exactly as they are entered into a data source with the ON MATCH/NOMATCH TED command. See the <i>Overview and Operating Environments</i> manual for additional information.
Syntax:	SET {TEXTFIELD TXTFIELD} = { <a href="#">OLD</a>  NEW} where: <a href="#">OLD</a> Enables you to use text field data in prior releases of FOCUS when that data has been created or modified in Release 7.0. This value is the default.  <a href="#">NEW</a> Disables the ability to use text field data in prior FOCUS releases when that data has been created or modified in Release 7.0.
<b>Parameter:</b>	TITLE
Description:	Uses pre-defined column titles in the Master File as column titles in report output.
Syntax:	SET TITLE[S] = { <a href="#">ON</a>  OFF} where: <a href="#">ON</a> Uses pre-defined column titles in the Master File as column titles in report output. This value is the default.  <a href="#">OFF</a> Uses the field names in the Master File as column titles in report output.

- Parameter:** `TOPMARGIN`
- Description: This parameter applies to PostScript and PDF report formats. Sets the top boundary on a page for report output.
- Syntax: `SET TOPMARGIN = {value|.25}`
- where:
- `value`
- Is the top boundary on a page for report output. The default value is .25 inches.
- Parameter:** `TRACKIO`
- Description: MVS FOCUS gathers more pages to fill a track before reading or writing the pages to disk. This results in significant reductions in I/O requirements and in elapsed time for FOCUS files.
- Syntax: `SET TRACKIO = {ON|OFF}`
- where:
- `ON`
- Enables FOCUS to fill a track before reading or writing to disk. This value is the default.
- `OFF`
- Does not fill a track before reading and writing to a disk.
- Parameter:** `TRMOUT`
- Description: Suppresses all output messages to the terminal.
- Syntax: `SET TRMOUT = {ON|OFF}`
- where:
- `ON`
- Displays output messages to the terminal. This value is the default.
- `OFF`
- Suppresses messages to the terminal.



**Parameter:** UNITS

Description: This parameter applies to PostScript and PDF report formats. Specifies the unit of measure for page margins, column positions, and column widths.

Syntax: `SET UNITS = {INCHES | CM | PTS}`

where:

INCHES

Uses inches as the unit of measure. This value is the default.

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)

**Parameter:** USER

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

Syntax: `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.

<b>Parameter:</b>	<a href="#">WEBTAB</a>
Description:	<p>Instructs FOCUS to enclose CRTFORM display fields in @ signs.</p> <p>When the HTML/TP feature of Web390 generates replacement HTML forms for a 3270 screen, it can dynamically account for fields that may or may not be populated with data during execution. HTML/TP can use this technique with turnaround (T.) fields on CRTFORMs because they are enclosed in @ signs. These @-sign markers enable HTML/TP to recognize them and handle them dynamically on a customized HTML form. In contrast, CRTFORM display (D.) fields are not normally enclosed in @ signs.</p> <p><b>Note:</b> This setting is only for those MODIFY CRTFORM or Dialogue Manager -CRTFORM applications that will be used in conjunction with the HTML/TP feature of Web390. For information about Web390 and the HTML/TP feature, see the <i>Web390 for OS/390 and MVS Developer's Guide and Installation Manual</i>.</p>
Syntax:	<p><code>SET WEBTAB = {<a href="#">ON</a> <a href="#">OFF</a>}</code></p> <p>where:</p> <p><a href="#">ON</a></p> <p>Adds @ signs around CRTFORM display fields. These markers may cause the fields displayed on the CRTFORM to shift slightly to the right. Use this setting only for MODIFY CRTFORM or Dialogue Manager -CRTFORM applications that will be used in conjunction with the HTML/TP feature of Web390.</p> <p><a href="#">OFF</a></p> <p>Does not place @ signs around CRTFORM display fields. This value is the default.</p>

<b>Parameter:</b>	WEEKFIRST
Description:	This parameter is used in week computations by the HDIFF, HNAME, HPART, and HSETPT functions described in Chapter 3, <i>Using Functions and Subroutines</i> . The values from 1 to 7 represent Sunday through Saturday.
Syntax:	<code>SET WEEKFIRST = number</code> where: <i>number</i> Is a number from one to seven, where one represents Sunday and seven represents Saturday. The U.S. English default value is seven (Saturday) meaning that Saturday is the first day of each week, so every Friday-Saturday transition is the start of a new week.  The WEEKFIRST setting does not change the number that corresponds to each day of the week, it just specifies which one is considered the start of the week. The default of Saturday (7) as the first day of the week is consistent with the Microsoft SQL Server convention.
<b>Parameter:</b>	WIDTH
Description:	Specifies the logical record length of your output data set when the STYLESHEET parameter is OFF. When the STYLESHEET parameter is in effect, FOCUS ignores the setting for WIDTH.
Syntax:	<code>SET WIDTH = {n 130}</code> where: <i>n</i> Is the logical record length of your output data set. The default value is 130.

- Parameter:** XRETRIEVAL
- Description: 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.
- Syntax: SET XRETRIEVAL = {ON|OFF}
- where:
- ON  
Performs retrieval when previewing a report. This value is the default.
- OFF  
Specifies that no retrieval is to be performed.
- Parameter:** YRTHRESH
- Description: Defines the start of a 100-year window globally or on a field-level. Used with DEFCEM, 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 DEFCEM.)
- Note:** This same result can be achieved by including the FDEFCEM and FYRTHRESH attributes in the Master File.
- Syntax: 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, that number is the start of the 100-year window. Any two-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 date is automatically calculated. The start date is automatically incremented by one at the beginning of each successive year.

---

## CHAPTER 2

# Querying Your Environment

### Topics:

- Using Query Commands

You can query your environment to display information such as status of files, release information, server information, and joins.

# 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 stored procedure or at the command prompt.

## Reference

### Query Command Summary

The following is a list of query commands. A detailed description of each is in this topic.

? COMBINE	Lists FOCUS files comprising the current COMBINE structures.
? DEFINE	Displays currently active virtual fields created by the DEFINE command or attribute.
? EUROFILE	Displays the currency data source in effect.
? F	Lists fields currently available to you.
? 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 were changed.
? FUNCTION	Displays the defined functions and their parameters.
? 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.
? n	Displays an explanation of an error message ( <i>n</i> represents the number of the error message).
? PTF	Displays the PTFs that have been applied to your version of FOCUS.

? RELEASE	Displays the release number of your product.
? SET	Displays parameter settings that control your development and run-time environment.
? SET GRAPH	Displays parameter settings that control graphs produced with the GRAPH command.
? 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.
? &&	Displays values of global variables.

## Displaying Combined Structures

The ? COMBINE command displays files that are in the current COMBINE structures.

### Syntax

### How to Display Combined Structures

```
? COMBINE [filename]
```

where:

*filename*

Is the data source containing the virtual fields. If *filename* is omitted, the command displays all virtual fields.

### 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, then an ampersand (&) in the 32nd position indicates a longer field name.

## Syntax

### How to Display Virtual Fields

```
? DEFINE [filename]
```

where:

*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 virtual fields 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:

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: <pre>DEFINE FILE EMPLOYEE.JOBCODE</pre>
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 ? EUROFILE command displays the currency data source in effect.

## Syntax How to Display the Currency Data Source in Effect

? EUROFILE

## Example Displaying the Currency Data Source in Effect

Issuing the command

? EUROFILE

produces information similar to the following:

EUROFILE                      GBP

# Displaying Available Fields

The ?F command displays the fields that are currently available.

## Syntax How to Display Available Fields

?F [*filename*]

where:

*filename*

Is a data source. If *filename* is omitted, the command displays all virtual fields.

## Example Displaying Available Fields

Issuing the command

?F

produces information similar to the following:

```
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_ATTENDATTENDSEG.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.

A FOCUS data source is composed of fixed-length, 4096-byte records called pages. Each segment and each 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          3          3          1          4
5  SECSEG           4              4          3          3          1          2
6  SKILLSEG         11             4          3          3          1          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          6          6          1          3
11 COURSEG          11             10         6          6          1          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 determine the number of instances that can fit into 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 that are 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
```

produces information similar to the following:

```
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_INFOJOB   JOBCODEJBC   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 file's last update 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 HLI PRINT 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. Each page is 4096 bytes.
LAST CHANGE	Is the date and time the data source was last changed.

## Determining the Percentage of File Disorganization

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 you can 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 DEFINE Functions

The ? FUNCTION command displays all defined functions and their parameters.

### Syntax

### How to Display DEFINE Functions

```
? 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, their aliases, and their 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**

? 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
-----      -
JOBCODE      EMPLOYEE  JOBCODE  JOBFILE    EMPJOB

```

>

## 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.

## Displaying National Language Support

The ? LANG command displays information about National Language Support.

### Syntax

#### How 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 your documentation on defining LET substitutions.

### Syntax

#### How to Display LET Substitutions

```
? 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 will be displayed. 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

CAR	MASTER	4200	BYTES
EXPERSON	MASTER	4200	BYTES
CARTEST	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
```

and want a fuller explanation, issue:

```
? 125
```

The following message is displayed:

```
(FOC125) RECAP CALCULATIONS MISSING
```

```
The word RECAP is not followed by a calculation. Either the RECAP should be removed, or a calculation provided.
```

## Querying Which PTFs Have Been Applied for a Specific Release

The ? PTF command displays a list of PTFs that have been applied to the version of FOCUS you are currently using.

**Syntax**            **How to Query a List of PTFs**

```
? PTF
```

**Example**            **Querying a List of PTFs**

Issuing the command

```
? PTF
```

produces results similar to the following:

```
>
```

```
? ptf
```

```
PTFS APPLIED TO RELEASE 70XFOC  
FROM PTFTABLE LOCATED IN IBITEST LOADLIB C1
```

COUNT	PTF NUM	CREATED	APPLIED	SUPERSEDED BY	PUT LEVEL
-----	-----	-----	-----	-----	-----
1)	95828	.....	.....	112600	.....
2)	107164	.....	.....	112600	.....
3)	110763	.....	.....	112600	.....
4)	112600	19990427	19990513	.....	200295

```
>
```

**Note:** Dots are used to denote the lack of data if no information exists for a column entry in the resulting report. If there are no PTFs for the version of FOCUS that you are currently running, the following is displayed:

```
NO PTFS HAVE BEEN APPLIED
```

## 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

```
? RELEASE
```

### *Example*      Displaying the Release Number

Issuing the command

```
? RELEASE
```

produces information similar to the following:

```
FOCUS 7.0.9                      created 9/16/1999 QA-30.01
```

## Displaying Parameter Settings

The ? SET command lists the parameter settings that control your development and run-time environments. Your application sets default values for these parameters, but you can change them with the SET command.

Two options give you additional information. The FOR option lists the current state of the command queried, and describes where you can set it. The NOT option produces a list of SET commands not settable in five specific areas.

SET parameters are described in Chapter 1, *Customizing Your Environment*.

### *Syntax*      How to Display Parameter Settings

```
? SET [ALL|parameter]
```

where:

*ALL*

Optionally displays all possible parameter settings.

*parameter*

Is a SET parameter.

## Example      Displaying Parameter Settings

Issuing the command

```
? SET
```

produces information similar to the following:

```

                                PARAMETER SETTINGS
ALL.                            OFF  HIPERFOCUS                OFF  QUALCHAR                .
ASNAMES                         FOCUS HOLDATTRS                FOCUS QUALTITLES            OFF
AUTOINDEX                       ON   HOLDLIST                  ALL  RECAP-COUNT            OFF
AUTOPATH                         ON   HOLDSTAT                 OFF  SAVEMATRIX              ON
BINS                             64  HOTMENU                  OFF  SCREEN                  ON
BLKCALC                          NEW  INDEX TYPE                NEW  SHADOW PAGE            OFF
BYPANELING                       OFF  LANGUAGE                  AMENGLISH SPACES                AUTO
CACHE                             0   LINES/PAGE                66  SQLENGINE               .
CARTESIAN                       OFF  LINES/PRINT              57  TCPIPINT                OFF
CDN                              OFF  MESSAGE                  ON   TEMP DISK               A
COLUMNSCROLL                     OFF  MODE                      CMS  TERMINAL                IBM3270
DATETIME  STARTUP/RESET          NODATA .                      .   TITLES                  ON
DEFCENT                          19  PAGE-NUM                 ON   WIDTH                   130
EMPTYREPORT                      OFF  PANEL                    0   WINPFKEY                OLD
EXTSORT                          ON   PAUSE                   ON   XRETRIEVAL              ON
FIELDNAME                       NEW  PRINT                    ONLINE YRTHRESH              0
FOCSTACK SIZE                    8   PRINTPLUS                ON
>

```

Some parameters are listed differently from the way you specify them in the SET command. These include:

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 ONLINE-FMT
```

produces information similar to the following:

```
ONLINE-FMT                    STANDARD
```

**Syntax**

**How to Query a Command**

? SET FOR *parameter*

where:

*parameter*

Is any SET parameter.

**Example**

**Querying Where the EXTSORT Parameter Is Valid**

Entering

? SET FOR EXTSORT

yields

```
EXTSORT                                ON

-----
SETTABLE FROM COMMAND LINE             : YES
SETTABLE ON TABLE                     : YES
SETTABLE FROM SYSTEM-WIDE PROFILE      : YES
SETTABLE FROM HLI PROFILE              : YES
POOL TABLE BOUNDARY                   : YES
>
```

The preceding screen shows that EXTSORT is currently set ON and that it is settable from all five features.

**Syntax**

**How to Determine Where a Command Is Valid**

? SET NOT *functional\_area*

where:

*functional\_area*

Is one of the following:

PROMPT is in a PROMPT command.

ONTABLE is in a report request.

FOCPARM is in the FOCPARM profile.

HLIPROF is in the HLI profile.

PT is in Pooled Tables.

**Example**

**Determining Which Commands Are Not Valid Using ON TABLE**

Entering

```
? SET NOT ONTABLE
```

yields:

NON-SETTABLE ON TABLE PARAMETER SETTINGS

BINS	64	LANGUAGE	AMENGLISH	REBUILDMSG	1000
BLKCALC	NEW	MAXPOOLMEM	32768	SAVEMATRIX	ON
BYPANELING	OFF	MDIBINS	8000	TCPIPINT	OFF
CACHE	0	MDIPROGRESS	100000	TEMP DISK	C
COLUMNSCROLL	OFF	MODE	CMS	TRMSD	24
DATEDISPLAY	OFF	MPRINT	NEW	TRMSW	80
DATEFNS	ON	POOL	OFF	TRMTYP	1 (3270)
DEFCENT	19	POOLBATCH	OFF	WEBHOME	OFF
EUROFILE		POOLFEATURE	OFF	WIDTH	130
FIELDNAME	NEW	POOLMEMORY	16384	WINPFKEY	OLD
FOCSTACK SIZE	8	POOLRESERVE	1024	YRTHRESH	0
HTMLMODE	OFF	PRINTPLUS	OFF		

>  
The preceding screen shows a list of parameters that are not settable using ON TABLE.



# 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 Chapter 1, *Customizing Your Environment*.

## Syntax

### How to Display Graph Parameters

```
? 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 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. For more information, see Chapter 4, *Managing Applications With Dialogue Manager*.

## Syntax

### How to Display Command Statistics

```
? STAT
```

## Example

### Displaying Command Statistics

Depending on the commands executed

```
? 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       =          0      FORMAT ERRORS =          0
SORT PAGES   =          0      INVALID CONDT=          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 is listed when the ? STAT query is issued:

RECORDS	Is for TABLE, TABLEF, MATCH, and GRAPH commands. Indicates the number of records used in the report. Note that the meaning of a record depends on the type of data source used.
LINES	Is for TABLE and TABLEF commands. Indicates the number of lines printed in a report.
BASEIO	Is for TABLE, TABLEF, GRAPH, MODIFY, and FSCAN commands. Indicates the number of I/O operations performed on the data source.
SORTIO	Is for TABLE, TABLEF, MATCH, and GRAPH commands. Indicates the number of I/O operations performed on the FOCSORT file, a work file invisible to the user.
SORTPAGES	Is for TABLE and TABLEF commands. Indicates the number of physical records in the FOCSORT file.
READS	Is for the MODIFY and FSCAN commands. Indicates the number of fixed format records read in external files by the FIXFORM command.
TRANSACTIONS	Is for the MODIFY and FSCAN commands. Indicates the number of transactions processed—inputs, updates, deletions, and rejections.
ACCEPTED	Is for the MODIFY and FSCAN commands. Indicates the number of transactions accepted.
SEGS INPUT	Is for MODIFY and FSCAN commands. Indicates the number of segment instances accepted into the data source.
SEGS CHNGD	Is for MODIFY and FSCAN commands. Indicates the number of segment instances updated in the data source.
SEGS DELTD	Is for MODIFY and FSCAN commands. Indicates the number of segment instances deleted from the data source.
NOMATCH	Is for the MODIFY command. 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 command. Indicates the number of transactions rejected because their matching field values already exist in the data source. This occurs on an ON MATCH REJECT condition.
FORMAT ERRORS	Is for the MODIFY command. Indicates the number of transactions rejected because data field values for data fields do not conform to the field formats defined in the Master File.

<code>INVALID CONDTS</code>	Is for the MODIFY command. Indicates the number of transactions rejected because their values failed validation tests.
<code>OTHER REJECTS</code>	Is for the MODIFY command. Indicates the number of transactions rejected for reasons other than those listed above.
<code>CACHE READS</code>	Shows the number of CACHE READS performed (see Chapter 1, <i>Customizing Your Environment</i> ).
<code>MERGES</code>	Is the number of times that FOCUS merge routines have been invoked.
<code>SORT STRINGS</code>	Is the number of times that the FOCUS SORT capacity has been exceeded.
<code>INTERNAL MATRIX CREATED</code>	Can have a value of YES/NO.
<code>SORT USED</code>	Is the type of sort facility used. It can have a value of FOCUS, EXTERNAL, SQL, or NONE.

## Displaying Information About the SU Machine

The ? SU command displays the communication available to the SU machine.

### Syntax

#### How to Display Information About the SU Machine

```
? SU [userid|ddname]
```

where:

`userid`

Is a valid user ID.

`ddname`

Is a valid ddname.

### Example

#### Displaying Information About the SU Machine

Issuing the command

```
? SU SYNCA
```

produces the following information:

```
USERID FILEID QUEUE  
WIBMLH QUERY  
WIBJBP CAR
```

## Displaying Global Variable Values

The ? && command lists Dialogue Manager global variables and their current values. Global variables maintain their values for the duration of the session.

See your documentation about Dialogue Manager for details on global and other variables.

### *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:

```
? %%
```

You can query all Dialogue Manager variables (local, global, and system) from a stored procedure by issuing:

```
-? &
```

### *Example*

#### Displaying Global Variable Values

Depending on the variables in effect, issuing the command

```
? &&
```

produces information similar to the following:

```
&&STORECODE    001
&&STORENAME    MACYS
>
```

```
>
```

## 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
```

## 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

? 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
```

---

## CHAPTER 3

# Using Functions and Subroutines

### Topics:

- What Is the Difference Between a Function and a Subroutine?
- Types of Functions and Subroutines
- Date Function and Subroutine Settings
- Subroutine Command (Call) Syntax
- Storing and Accessing External Subroutines
- Alphabetical List of Functions and Subroutines

FOCUS offers a rich set of functions and subroutines that operate on one or more arguments and return a single value as a result. Functions and subroutines provide a convenient way to perform certain calculations and manipulations.

The next few topics describe the following:

- What is the difference between a function and a subroutine?
- Types of functions and subroutines.
- How to use subroutines.
- An alphabetical description of the functions and subroutines.

You can also create your own subroutines.

## What Is the Difference Between a Function and a Subroutine?

There are two differences between a function and a subroutine:

- How they are invoked.
- How they are accessed.

A function call has the following syntax

```
function (arg1, arg2, ...)
```

where:

```
function
```

Is the name of the function.

```
arg1, arg2, ...
```

Are the arguments.

A subroutine call has the following syntax

```
subroutine (arg1, arg2, ... {outputfield|'format'})
```

where:

```
subroutine
```

Is the name of the subroutine.

```
arg1, arg2, ...
```

Are the arguments.

```
{outputfield|'format'}
```

Is the name of the output field or its format.

In addition, on some platforms, the functions are available immediately; whereas, the subroutines are available in a special subroutine library that you must access.



# Types of Functions and Subroutines

You can access any of the following kinds of functions and subroutines:

## Bit

Enable you to manipulate bits.

## Character

Enable you to manipulate alphanumeric fields or character strings.

## Data Source

Enable you to search for or retrieve data source records or values.

## Date and Time

Enable you to manipulate dates and times.

## Decoding

Enable you to assign values.

## Format Conversion

Enable you to convert fields from one format to another.

## Numeric

Enable you to perform numeric calculations on numeric constants and fields.

## System

Enable you to make calls to the operating system to obtain information about the operating environment or to use a system service.

## Bit Functions and Subroutines

Bit functions and subroutines enable you to manipulate bits. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-36.

### BITSON subroutine

Evaluates an individual bit within a character string to determine whether it is on or off.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### BITVAL subroutine

Evaluates a string of bits within character strings and returns its binary value.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### BYTVAL subroutine

Translates a character to its corresponding ASCII code.

**HEXBYT subroutine**

Translates an integer between 0 and 255 (base 10) into the corresponding ASCII or EBCDIC character (depending on your platform).

**UFMT subroutine**

Converts characters in alphanumeric field values to hexadecimal (HEX) representation.

**Available on:** MVS, VM/CMS, OpenVMS, and WebFOCUS.

## Character Functions and Subroutines

The following functions and subroutines enable you to manipulate alphanumeric fields or character strings. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-36.

**ARGLEN subroutine**

Measures the length of a character string within a field, excluding trailing blanks.

**ASIS function**

In Dialogue Manager, distinguishes between a blank and a zero.

**Available on:** MVS, VM/CMS, and WebFOCUS.

**BITSON subroutine**

Evaluates an individual bit within a character string to determine whether it is on or off.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

**BITVAL subroutine**

Evaluates a string of bits within character strings and returns its binary value.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

**BYTVAL subroutine**

Translates a character to its corresponding ASCII code.

**CHKFMT subroutine**

Checks for incorrect character types by comparing each character in the input string to the corresponding character in a mask.

**CTRAN subroutine**

Converts one character in a string to another character.

**CTRFLD subroutine**

Centers a character string within a field, excluding trailing blanks.

**EDIT function**

Extracts characters or adds characters to an alphanumeric string (with mask).

**GETTOK subroutine**

Divides a character string at a character called the delimiter and returns a substring called the token.

**LCWORD subroutine**

Converts the letters in the given string to mixed case. The subroutine converts to uppercase the first letter of each new word and the first letter after a single or double quotation mark.

**Available on:** MVS, VM/CMS, WebFOCUS, and Windows.

**LJUST subroutine**

Left-justifies a character string within a field. All leading blanks become trailing blanks.

**LOCASE subroutine**

Converts uppercase characters to lowercase.

**OVLAY subroutine**

Overlays a substring on another character string.

**PARAG subroutine**

Divides lines of text into smaller lines with delimiters.

**POSIT subroutine**

Finds the starting position of a substring within a larger string.

**RJUST subroutine**

Right-justifies a character string within a field. All trailing blanks become leading blanks.

**SOUNDEX subroutine**

Searches for a character string phonetically rather than by the way it is spelled.

**SUBSTR subroutine**

Extracts substrings, based on where they start and end in the parent string.

**UPCASE subroutine**

Converts lowercase characters to uppercase.

## Data Source Functions and Subroutines

Data source functions and subroutines enable you to search for or retrieve data source records or values. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-36.

### FIND function

Verifies if a value exists in an indexed field in another file.

**Available on:** MVS, VM/CMS, and UNIX.

### LAST function

Retrieves the preceding value selected for a field.

### LOOKUP function

Retrieves a value from a cross-referenced file.

**Available on:** MVS, VM/CMS, and UNIX.

## Date Functions and Subroutines

The following functions and subroutines enable you to manipulate dates. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-2.

### AYM subroutine

Adds or subtracts months from dates that are in year-month format.

### AYMD subroutine

Adds or subtracts days from dates that are in year-month-day format.

### CHGDAT subroutine

Rearranges the year, month, and day portions of dates, and converts dates between long and short date formats.

### DA subroutines

Convert dates to the corresponding number of days elapsed since December 31, 1899.

### DATEADD subroutine

Adds or subtracts date units to or from a date.

**Available on:** MVS and VM/CMS.

### DATECVT function

Converts dates from one date format to another.

**Available on:** MVS and VM/CMS.

**DATEDIF function**

Calculates the difference between two dates.

**Available on:** MVS and VM/CMS.

**DATEMOV function**

Moves a date to a significant point on the calendar.

**Available on:** MVS and VM/CMS.

**DMY, MDY, and YMD functions**

Calculate the difference between two dates.

**DOWK[L] subroutines**

Determine the day of the week for dates.

**DT subroutines**

Convert smart dates (the number of days elapsed since December 31, 1899) to corresponding dates.

**GREGDT subroutine**

Converts dates in Julian format to year-month-day format.

**HADD**

Increments date-time values by a specified number of units.

**HCNVRT**

Converts date-time values to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.

**HDATE**

Extracts the date components from a date-time field and converts them to a date field.

**HDIFF**

Finds the number of boundaries of a given type crossed going from date 2 to date 1.

**HDTTM**

Converts a date field to a date-time field. The time portion is set to midnight.

**HGETC**

Stores the current date and time in a date-time field.

**HHMMSS subroutine**

Retrieves the current time from the system.

**HINPUT**

Converts an alphanumeric string to a date-time value.

**HMIDNT**

Changes the time portion of a date-time field to midnight.

**HNAME**

Extracts specified components of a date-time value and converts them to alphanumeric format.

**HPART**

Extracts a component of a date-time value in numeric format.

**HSETPT**

Inserts the numeric value of a specified component in a date-time field.

**HTIME**

Extracts all of the time components from a date-time field and converts them to a number of milliseconds or microseconds in numeric format.

**JULDAT subroutine**

Converts dates from year-month-day format to Julian (year-day format).

**TODAY subroutine**

Retrieves the current date from the system.

**YM subroutine**

Calculates the number of months that elapse between two dates. The dates must be in year-month format.

*Reference*

**Component Names and Values for Use With Date-Time Functions**

The following component names and values are supported as arguments to those date-time functions that require you to specify a component name as an argument:

<b>Component Name</b>	<b>Valid Values</b>
<code>year</code>	0001-9999
<code>quarter</code>	1-4
<code>month</code>	1-12
<code>day-of-year</code>	1-366
<code>day</code> or <code>day-of-month</code>	1-31 (The two names for the component are equivalent.)
<code>week</code>	1-53
<code>weekday</code>	1-7 (Sunday-Saturday)
<code>hour</code>	0-23
<code>minute</code>	0-59
<code>second</code>	0-59
<code>millisecond</code>	0-999
<code>microsecond</code>	0-999999

**Note:**

- In those arguments that give you a choice of 8 or 10, use 8 for processing values without microseconds, 10 when the field value includes microseconds.
- The last argument is always a USAGE format that indicates the data type returned by the function. The type may be A (alpha), I (integer), D (double precision), DATE (smart date), or H (date-time).

**Valid Date Input**

Date subroutines accept the following types of dates:

- Years that have four digits and display the century, such as 2000 and 1900, if their formats are specified as I8YYMD, P8YYMD, D8YYMD, F8YYMD, or A8YYMD.

The following example uses the DECODE function to assign dates with four-digit years. It then converts these dates to Julian and Gregorian formats.

```
DEFINE FILE CAR
DATE/I8YYMD=DECODE COUNTRY (ENGLAND 19960101 FRANCE 19991231 ELSE
20010101);
JDATE/I7=JULDAT(Date, 'I7');
GDATE/I8=GREGDT(JDATE, 'I8');
END
TABLE FILE CAR
PRINT DATE JDATE GDATE
END
```

The request produces the following report:

```
PAGE      1

      DATE      JDATE      GDATE
      ----      -
1996/01/01  1996001  19960101
2001/01/01  2001001  20010101
2001/01/01  2001001  20010101
2001/01/01  2001001  20010101
1999/12/31  1999365   19991231
```

- Two-digit years with a field length of 6 (such as I6YMD). In this case, you can use the SET DEFCENT and SET YRTHRESH commands to assign the century values.

The following example shows how to return an eight-digit date from the AYMD subroutine when the input argument has a six-digit date format. Since DEFCENT is 19 and YRTHRESH is 50, year values from 50 through 99 are interpreted as 1950 through 1999, and year values from 00 through 49 are interpreted as 2000 through 2049:

```
SET DEFCENT=19, YRTHRESH=50
TABLE FILE DATE
PRINT D2_I6YMD AND COMPUTE
NEWDATE/I8YYMD=AYMD(D2_I6YMD,1, 'I8');
END
```

The DEFCENT and YRTHRESH values create a 100-year window as follows:

```
0 _____ < YRTHRESH=50 ≥ _____ 99
      ↑                               ↑
      Century=DEFCENT+1 (20)         Century=DEFCENT (19)
```

The request produces the following report:

```
PAGE      1

D2_I6YMD   NEWDATE
-----
97/09/16   1997/09/17
00/02/29   2000/03/01
01/02/28   2001/03/01
00/02/28   2000/02/29
```

**Note:** If you do not require dates for the year 2000 and beyond, you can deactivate this feature by issuing the following command:

```
SET DATEFNS=OFF
```

## Decoding Functions and Subroutines

Decoding functions and subroutines enable you to assign values. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-36.

### DECODE function

Assigns values based on the value of an input field.



## Format Conversion Functions and Subroutines

The following functions and subroutines convert fields from one format to another. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-36.

### ASIS function

In Dialogue Manager, distinguishes between a blank and a zero.

**Available on:** MVS, VM/CMS, and WebFOCUS.

### ATODBL subroutine

Converts a number in alphanumeric format to double-precision format.

### CHKPCK subroutine

Verifies that the value of a packed field is indeed in packed format.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### EDIT function

Converts an alphanumeric field to numeric or a numeric field to alphanumeric.

### FTOA subroutine

Converts a number in a numeric format to alphanumeric format.

### ITONUM subroutine

Converts large binary integers in non-FOCUS files to double-precision format.

**Available on:** MVS, VM/CMS, and WebFOCUS.

### ITOPACK subroutine

Converts large binary integers in non-FOCUS files to packed-decimal format.

**Available on:** MVS, VM/CMS, and WebFOCUS.

### ITOZ subroutine

Converts numbers from numeric format to zoned format for extract files.

**Available on:** MVS, VM/CMS, OpenVMS, and WebFOCUS.

### PCKOUT subroutine

Writes packed numbers of varying lengths (between one and 16 bytes) to extract files.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### UFMT subroutine

Converts characters in alphanumeric field values to hexadecimal (HEX) representation.

**Available on:** MVS, VM/CMS, OpenVMS, and WebFOCUS.

## Numeric Functions and Subroutines

The following functions and subroutines enable you to perform numeric calculations on numeric constants or fields. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-16.

### ABS function

Returns the absolute value of its argument.

### ASIS function

In Dialogue Manager, distinguishes between a blank and a zero.

**Available on:** MVS, VM/CMS, and WebFOCUS.

### BAR subroutine

Produces horizontal bar charts in reports.

**Available on:** MVS and VM/CMS.

### EXP subroutine

Raises the number “e” to a power you specify.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### EXPN function

Evaluates an argument expressed in scientific notation.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### IMOD, DMOD, and FMOD

Calculate the remainder from a division and returns it in integer format.

### INT function

Returns the integer part of its argument.

### LOG function

Returns the logarithm of its argument.

### MAX and MIN functions

Returns the maximum or minimum value from its list of arguments.

### PRDNOR, PRDUNI, RDNORM, and RDUNIF subroutines

Generate random numbers.

### SQRT function

Returns the square root of its argument.

## System Functions and Subroutines

The following functions and subroutines enable you to make calls to the operating system to obtain information about the operating environment or to use a system service. Unless otherwise noted, functions and subroutines are supported on all platforms. For more information on these functions and subroutines, see *Alphabetical List of Functions and Subroutines* on page 3-36.

### **CALLDOS** subroutine

Calls a DOS program, a DOS batch program, or a Windows application.

**Available on:** Windows.

### **FEXERR** subroutine

Retrieves error messages.

**Available on:** MVS and VM/CMS.

### **FINDMEM** subroutine

Determines if a specific member of a partitioned data set exists.

**Available on:** MVS.

### **GETPDS** subroutine

Determines if a specific member of a partitioned data set exists, and if so, returns the data set name.

**Available on:** MVS.

### **GETUSER** subroutine

Retrieves the user ID from the system.

**Available on:** MVS, VM/CMS, UNIX, OpenVMS, and WebFOCUS.

### **HHMMSS** subroutine

Retrieves the current time from the system.

### **MVSDYNAM** subroutine

Passes a DYNAM command to the command processor.

**Available on:** MVS.

### **TODAY** subroutine

Retrieves the current date from the system.

## Date Function and Subroutine Settings

The following settings affect the behavior of the date functions and subroutines:

- DATEFNS determines whether you use the new or old version of a date subroutine.
- DEFCENT and YRTHRESH determine what century is used for dates that do not have a century specified. For more information on these two settings, see Chapter 7, *Working With Cross-Century Dates*.
- BUSDAYS enables you to define which days of the week are considered business days and which are not. Then, when you use DATEADD or DATEDIF with the business day unit, or use DATEMOV, these functions ignore dates that are not business days.
- HDAY determines a file with a list of holidays. Then, when you use DATEADD or DATEDIF with the business day unit, or use DATEMOV, these functions ignore dates that are currently defined as holidays.
- LEADZERO enables you to display leading zeros when a date subroutine in Dialogue Manager returns a date with leading zeros.

## Using Legacy Versions of Date Subroutines

All of the date subroutines have been rewritten to support Year 2000 dates. In some cases, however, you may want to use the old version of the subroutine. You can “turn off” the new versions with the DATEFNS setting.

See the description of each subroutine affected by this setting for the result of turning off DATEFNS.

**Note:** The old versions of these subroutines may not work correctly with dates after December 31, 1999.

### Syntax

### How to Activate Legacy Date Subroutines

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

where:

ON

Activates the subroutines that support dates for the year 2000 and beyond. This value is the default.

OFF

Deactivates the subroutines that support dates for the year 2000 and beyond; this is useful if you require the older version that does not have Year 2000 capabilities.

## Setting Business Day Units

You can direct which days are considered business days and which days are not. Business days are traditionally Monday through Friday, but not every business works the same schedule. For example, if your company does business on Sunday, Tuesday, Wednesday, Friday and Saturday, you can tailor business day units to reflect that situation.

Then, when you use DATEADD or DATEDIF with the business day unit, or use DATEMOV, these functions ignore dates that are not business days.

### Syntax

#### How to Set Business Days

```
SET BUSDAYS = smtwtfs
```

where:

*smtwtfs*

Is the seven-character list of days that represents your business week. The list has a position for each day from Sunday to Saturday.

- If you want a day of the week to be a business day, enter the first letter of that day in that day's position.
- If you want a day of the week not to be a business day, enter an underscore (\_) in that day's position.

If any position within SMTWTFS is either not in its correct position or is not an underscore, you will see an error message.

### Example

#### Setting Business Days to Sunday, Tuesday, Wednesday, Friday, and Saturday

Using the example of a company that does business on Sunday, Tuesday, Wednesday, Friday, and Saturday, business days are represented as:

```
SET BUSDAYS = S_TW_FS
```

### Syntax

#### How to View the Current Setting of Business Days

```
? SET [ALL]
```

## Setting Holidays

You can individually tailor holiday schedules that affect the calculation of business days by skipping those days when calculating offsets. Then, when you use DATEADD or DATEDIF with the business day unit, or use DATEMOV, these functions ignore dates that are currently defined as holidays.

For example, in a given week, if Friday is designated as a holiday, the next business day (BD) after Thursday is the following Monday.

The list of holidays is defined by a file called HDAYxxxx.

- In MVS this file should be a member in ERRORS called HDAYxxxx.
- In CMS the list should be HDAYxxxx ERRORS.

Each year for which data exists must be represented in the holiday file. Calling a date function with a date value outside the range of the holidays file returns a zero on BD requests.

### Procedure

#### How to Define Holidays Using a Holiday File

1. Open the procedure HDAYMAKE and follow the directions to create the holiday file. (Information Builders supplies a sample Master File named HDAYDB.)
2. Execute HDAYMAKE.
3. Execute the following SET command

```
SET HDAY = xxxx
```

where:

```
xxxx
```

Is the part of the name of the holiday file after HDAY.

A sample Master File (HDAYDB) and procedure (HDAYMAKE) that creates an errors member from a data source used to maintain a list of holidays is available on the FOCUS disk. Create a flat file of holidays as described in the procedure and execute the procedure to create the holiday file. The SET HDAY command controls the value of xxxx so that a single installation can support different holiday schedules.

### Example

#### Using the HDAYSTKM Holiday File

For example,

```
SET HDAY = STKM
```

reads in the holidays from member HDAYSTKM.

**Syntax**      **How to View the Current Setting of HDAY**

```
? SET [ALL]
```

**Enabling Leading Zeros For Date Subroutines in Dialogue Manager**

If you use a date subroutine in Dialogue Manager that returns a numeric integer format, Dialogue Manager will truncate any leading zeros. This means, for example, that if your subroutine returns the value 000101, indicating January 1, 2000, Dialogue Manager will truncate the leading zeros, and use 101.

To avoid this problem, you can use the LEADZERO setting.

**Syntax**      **How to Set the Display of Leading Zeros**

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

where:

ON

Allows the display of leading zeros if they are present.

OFF

Truncates leading zeros if they are present. This is the default.

**Example**      **Displaying Leading Zeros**

The following request calls the AYM subroutine (which adds months to dates in YM format) to add one month to the input date of December 1999.

```
-SET &IN = '9912';  
-SET &OUT = AYM(&IN, 1, 'I4');  
-TYPE &OUT
```

This yields:

```
1
```

Adding

```
SET LEADZERO = ON
```

before the above example yields

```
0001
```

correctly indicating January 2000.

**Note:** LEADZERO only supports expressions that make a direct call to a subroutine. Expressions that have nesting or other mathematical functions truncate leading zeros. For example,

```
-SET &OUT = AYM(&IN, 1, 'I4'/100);
```

## Subroutine Command (Call) Syntax

This topic describes the general syntax for subroutine calls, types of arguments, and rules for using arguments.

Subroutines return a single value or character string which can be stored in a field, assigned to a Dialogue Manager variable, used in calculations and other processing, or used in selection or validation tests. Subroutine calls in FOCUS commands have the general syntax

```
subroutine (input1, input2, input3, ... {outfield | 'format' }
```

where:

```
subroutine
```

Is the name of the subroutine, up to eight characters long.

```
input1...
```

Are the input subroutine arguments (data values and fields that the subroutine needs to do its processing).

```
{outfield | 'format' }
```

Is the output argument. It is the name of the field that contains the output or the format of the output value, enclosed in single quotation marks, depending on the application. For Maintain, specify the field name. Maintain does not support the output format as an argument. Dialogue Manager requires the output format.

The basic syntax to store the output in a field looks like:

```
field = subroutine (input1, input2, ... outfield);
```

Subroutine syntax varies for different FOCUS commands and phrases. These variations are discussed in subsequent sections.



## Types of Arguments in Subroutine Calls

This topic lists the acceptable arguments for each subroutine distributed in the subroutine library. Arguments are the values that you specify within the parentheses; this is also referred to as a “call list.” Arguments can take many different forms. They can be:

- Numeric constants, such as 6 or 15.
- Alphanumeric literals, such as 'STEVENS' or 'NEW YORK NY'. Literals are enclosed in single quotation marks.
- Numbers stored in alphanumeric format.
- Field names, such as FIRST\_NAME or HIRE\_DATE. Fields can be data source fields or temporary fields. The field names can be 66-character or qualified field names, unique truncations, or aliases.
- Expressions, such as numeric, date, and alphanumeric. Expressions can use the arithmetic operators and the concatenation sign (). For example, an expression may consist of `CURR_SAL * 1.03` or `FN || LN`.

For example, this calculation uses subroutine output:

```
field= sub1(input1, input2,...'format') + sub2(input1,  
input2,...'format');
```

The values returned by two subroutines are added and the result is stored in a field. The '*format*' argument in single quotation marks is the format of the value returned by each subroutine. The format argument is not supported in Maintain.

- Dialogue Manager variables, such as &CODE or &DDNAME.
- Date constants, such as '022894'.
- Formats of the output values, enclosed in single quotation marks.
- As input arguments for RECAP commands only, row and column references (R notation, E notation, or labels), or names of other RECAP calculations.

## Rules for Arguments in Subroutine Calls

The following rules apply for arguments:

- Depending on the subroutine, arguments can be either alphanumeric or numeric:
  - Alphanumeric arguments (such as literals and alphanumeric fields) are stored internally as one character per byte. Numbers can also be stored in alphanumeric format. Literals are enclosed in single quotation marks, except when specified in operating system -RUN commands (-MVS RUN, for example).  
**Note:** If an argument is listed as having a specific alphanumeric format such as A8, it is a required format and you must specify it.
  - Numeric arguments (such as numeric constants and numeric fields) are stored internally as binary or packed numbers. This includes arguments in integer (I), floating-point (F), double-precision (D), and packed (P) formats.  
**Note:** If an argument is listed as having a numeric format, you may specify any of the four numeric formats (I, F, D, and P). If an argument is listed as having a specific numeric format such as double-precision, it is a required format and you must specify it.

If you supply the wrong type of data for an argument, you will either cause an error or the subroutine will not return correct data.

- Arguments are passed to subroutines by reference, meaning that the memory location of the argument is passed. No indication of length of the argument is implied. The length, when needed (usually for alphanumeric strings) must be passed as another argument.

When lengths of arguments are required, you must be careful to ensure that all lengths are correct. Some subroutines require a length for the input arguments and output arguments (for example, SUBSTR); others use one length for both input and output arguments (for example, UPCASE). In general, when one length is specified, it is used for both input and output fields.

Providing an incorrect length can cause incorrect results:

- If the specified length is shorter than the actual length, an initial subset of a string is used. For example, passing an argument of 'ABCDEF' and specifying a length of 3, is treated as a string of 'ABC'.
- If the specified length is too long, whatever is in memory beyond the length is included. For example, passing an argument of 'ABC' and specifying a length of 6, is treated as a string beginning with 'ABC' plus whatever three characters are in the next 3 positions of memory. Depending on memory utilization, the extra three characters can be anything.

- Arguments must be specified in the exact order as shown for each subroutine; the order varies, according to each subroutine.
- The number of arguments varies, according to each subroutine. Subroutines may require up to six arguments.

Customized subroutines may require any number of arguments. The maximum number of arguments per subroutine call is 28, including the output argument. If the subroutine requires more than 28 arguments, you must use two or more call statements to pass the arguments to the subroutine.

- Subroutine calls can serve as arguments in other subroutine calls or in FOCUS functions.
- You cannot specify a Dialogue Manager amper variable for the output argument without coding &VAR.EVAL. You may specify an amper variable as an input argument.
- Dialogue Manager converts arguments to double precision when it deems appropriate. The determination is made solely based on the value of the argument; not on what the subroutine expects for its pre-determined formats.

If the argument is numeric (&arg.TYPE is 'N'), the value is converted to double precision. If the subroutine expects an alphanumeric string and the input is a numeric string, incorrect results will occur because of the conversion to double precision. To resolve this problem, append a non-numeric character to the end of the string, but do not count this extra character in the length of the argument.

For example, to prevent the conversion of a delimiter blank character (' ') to a double precision zero in the GETTOK subroutine, include a non-numeric character after the blank (for example, ' @'). The GETTOK uses only the first character (the blank) as a delimiter and the extra character (@) prevents conversion to double precision.

## Using Subroutine Calls in FOCUS Functions

Subroutines can serve as arguments in the FOCUS functions described in this chapter. For example, the MAX function returns the largest argument in a list. The statement

```
field = MAX (5000, subroutine (arguments, 'format'));
```

stores either the value 5000 or the value returned by the subroutine, whichever is larger, in a field.

## Using Subroutine Calls in DEFINE, COMPUTE, and VALIDATE Commands

Subroutines may be called from the DEFINE command or Master File attribute, COMPUTE command, and VALIDATE command. The syntax is:

```
DEFINE [FILE filename]
tempfield[/format] = subroutine (input1, input2, input3, ...
{outfield|'format2'} );
```

```
COMPUTE
tempfield[/format] = subroutine (input1, input2, input3, ...
{outfield|'format2'} );
```

```
VALIDATE
tempfield[/format] = subroutine (input1, input2, input3, ...
{outfield|'format2'} );
```

The resulting temporary field is the same field that is specified for the outfield argument.

The temporary field's format is required if it is the first time the field is defined; afterwards, it is optional.

If the subroutine returns output as the format of the output value (format2), the format of the temporary field must match the 'format2' argument. For example:

```
CENTER_NAME/A15=CTRFLD (LAST_NAME, 15, 'A15');
```

For a calculation or a compound IF statement, you must specify the format for the output value. There are two methods to do this:

- Pre-define the format of the output field with a separate statement. For example:

```
COMPUTE
AMOUNT/D8.2 =;
AMOUNT_FLAG/A5 = IF subroutine(input1,input2,AMOUNT) GE 500
THEN 'LARGE' ELSE 'SMALL';
```

The AMOUNT field is pre-defined with the format D8.2. The subroutine returns a value to the output field AMOUNT (last argument). The IF statement tests if AMOUNT is greater or less than 500 and stores the result in the temporary flag AMOUNT\_FLAG.

- Specify the last argument in the argument list as the format. For example:

```
AMOUNT_FLAG/A5 = IF subroutine(input1,input2,'D8.2') GE 500
THEN 'LARGE' ELSE 'SMALL';
```

The statement tests the returned value directly. This is possible because the subroutine call defines the format of the return value (D8.2).

## Using Subroutine Calls in WHERE and IF Tests

Subroutines may be specified in WHERE and IF selection tests. The output value of the subroutine is compared against the test value.

For example, this request prints employee names and current salaries for last names that begin with the letters MC. The SUBSTR subroutine extracts the first two characters as a substring.

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME CURR_SAL
WHERE SUBSTR(15, LAST_NAME, 1, 2, 2, 'A15') IS 'MC';
END
```

The report returns as:

FIRST_NAME	LAST_NAME	CURR_SAL
-----	-----	-----
JOHN	MCCOY	\$18,480.00
ROGER	MCKNIGHT	\$16,100.00

## Using Subroutine Calls in -SET Control Commands

In Dialogue Manager, -SET commands are used to create amper variables. To assign the returned value of a subroutine to an amper variable, use the syntax:

```
-SET &variable = subroutine (input1, &variable2[.LENGTH], ..., 'format');
```

The *'format'* argument is the format of the output value, enclosed in single quotation marks. You cannot specify a Dialogue Manager amper variable for the output argument (*'format'*); however, you may specify an amper variable as an input argument.

If a subroutine requires the length of a character string as an input argument, you may prompt for the character string, then use the suffix .LENGTH to test the length.

For example, this Dialogue Manager procedure prompts for a sentence (&SN), then uses the GETTOK subroutine to extract the third word (token) from the sentence. (See *GETTOK: Getting a Token From a String* on page 3-61.) The suffix .LENGTH passes the number of characters in the sentence to the subroutine. The extra character (%) is included to prevent the conversion of a delimiter blank character to a double precision zero (see *Rules for Arguments in Subroutine Calls* on page 3-20):

```
-PROMPT &SN. ENTER A SENTENCE.
-SET &WORD3 = GETTOK (&SN, &SN.LENGTH, 3, '%', 30, 'A30');
```

Dialogue Manager variables only contain alphanumeric data. If a subroutine returns a numeric value and you set a Dialogue Manager variable to this value, FOCUS truncates and converts it to a character string before storing it in the variable based on the format.

Another example, the AYMD subroutine, adds 14 days to dates:

```
-SET &OUTDATE = AYMD (&INDATE, 14, 'I6');
```

For more information, see *AYMD: Adding or Subtracting Days to or From Dates* on page 3-45.

The &INDATE variable for the input date is previously set in the procedure. The date is in the six-digit year-month-day format.

The format of the output date is a six-digit integer. Although the format ('I6') indicates that the output is an integer, it is stored in the &OUTDATE variable as a character string. For this reason, if you display the value of &OUTDATE, you will not see slashes separating the year, month, and day.

## Using Subroutine Calls in -IF and IF Branching Commands

You can use subroutines in Dialogue Manager -IF and IF branching commands. The syntax is:

```
[-]IF subroutine (args) relation expression GOTO label1  
[-]ELSE GOTO label2;
```

Specify input arguments and the format of the output ('*format*').

You may specify any valid relation and logical expression. Depending on whether the condition is true or false, the procedure branches to the specified Dialogue Manager label or MODIFY case.

For -IF statements:

- You cannot specify a Dialogue Manager amper variable for the output argument ('*format*') unless you use the &VAR.EVAL syntax; however, you may specify an amper variable as an input argument.
- If a subroutine requires the length of a character string as an input argument, you can prompt for the character string, then use the suffix .LENGTH to test for the length (see *Using Subroutine Calls in -SET Control Commands* on page 3-23).

This annotated example illustrates an -IF statement that executes one of two requests depending on when a planned project is expected to be completed.

```
-LOOP
1. -PROMPT &INDATE. ENTER START DATE IN YEAR-MONTH-DAY FORMAT OR ZERO TO EXIT: .
2. -IF &INDATE EQ 0 GOTO EXIT;
3. -SET &WEEKDAY = DOWK(&INDATE, 'A4' );
4. -TYPE START DATE IS &WEEKDAY &INDATE
5. -PROMPT &DAYS. ENTER ESTIMATED PROJECT LENGTH IN DAYS: .
6. -IF AYMD(&INDATE, &DAYS, 'I6YMD') LT 960101 GOTO EARLY;
   -TYPE LONG PROJECT
   -*EX LONGPROJ
7. -RUN
   -GOTO LOOP
   -EARLY
   -TYPE SHORT PROJECT
   -*EX SHRTPROJ
8. -RUN
   -GOTO LOOP
   -EXIT
```

This procedure processes as follows:

1. The procedure prompts you for a start date of a project in YYMMDD format.
2. If you enter a 0, the procedure terminates execution.
3. The DOWK subroutine obtains the day of week for the start date.
4. The -TYPE statement displays the day of week and date for the start of the project.
5. The procedure prompts you for the estimated length of the project in days.
6. The AYMD subroutine calculates the date that the project will finish. If this date is before January 1, 1996, the -IF statement branches to the label EARLY.
7. If the project will finish on or after January 1, 1996, the procedure types the words “LONG PROJECT” and returns to the top of the procedure.
8. If the procedure branches to the label -EARLY, it types the words “SHORT PROJECT” and returns to the top of the procedure.

## Operating System -RUN Commands

You can call subroutines with all alphanumeric arguments from Dialogue Manager -CMS RUN, -TSO RUN, and -MVS RUN commands. These subroutines perform specific tasks but typically do not return any values (for instance, a private subroutine that clears the screen of a non-3270 terminal).

The syntax of the RUN command is:

```
-CMS RUN subroutine, input1, input2, ... [,&output]
-TSO RUN subroutine, input1, input2, ... [,&output]
-MVS RUN subroutine, input1, input2, ... [,&output]
```

Separate the subroutine name and each argument with a comma. For alphanumeric literals used as arguments, do not enclose them in single quotation marks.

Specify an output argument as a Dialogue Manager variable if the subroutine returns a value; otherwise, omit it. If you specify an output variable, you must pre-define its length using a -SET statement. For example, if the subroutine requires an output argument that is eight bytes long, you need to define the variable with eight characters enclosed in single quotation marks before the subroutine call:

```
-SET &output = '12345678';
```

For subroutines that require arguments in numeric format, you must first convert the arguments (whether they are numeric constants or stored in variables) into double-precision numbers using the ATODBL subroutine. (See *ATODBL: Converting Alphanumeric Strings to a Double-Precision Number* on page 3-39 for more details). All numeric arguments in Dialogue Manager are stored in alphanumeric format and require conversion before being passed to subroutines. Unlike the -SET statement, operating system -RUN commands do not automatically convert them. You need to use the ATODBL subroutine, because the EDIT function cannot store double-precision numbers in Dialogue Manager variables.

If a subroutine requires the length of a character string as an input argument, you may prompt for the character string, then use the suffix .LENGTH to test the length. (See *Using Subroutine Calls in -SET Control Commands* on page 3-23 for an example.)

The following is an example of a subroutine that does not return any values. Suppose you write a subroutine called BLANKOUT that clears part of the screen on a Tektronix terminal (a non-3270 terminal). The subroutine reads one argument that indicates which part of the screen to blank out. To clear the top half of the screen, you include this statement in a procedure

```
-CMS RUN BLANKOUT, H1
```

or:

```
-TSO RUN BLANKOUT, H1
```



## Using Subroutine Calls in WHEN Criteria

Subroutines may be called from the WHEN criteria as part of a Boolean expression. The syntax is

```
WHEN (value relation value) [{AND|OR} (value relation value)];
```

or:

```
WHEN NOT (logical expression)
```

### Example

#### Using a Subroutine Call in a WHEN Phrase

For example, this report request checks the values in the LAST\_NAME field against a mask. It prints a subfoot message when the condition, a match, occurs.

**Note:** In this example, in order to produce a true condition, specify WHEN NOT, because the CHKFMT subroutine returns a 0 value when a match occurs.

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT BY LAST_NAME
ON LAST_NAME SUBFOOT
  *** LAST NAME <LAST_NAME DOES MATCH MASK"
WHEN NOT CHKFMT (15, LAST_NAME, 'SMITH          ', 'I6');
END
```

The report returns as:

```
LAST_NAME      DEPARTMENT
-----      -
BANNING        PRODUCTION
BLACKWOOD      MIS
CROSS          MIS
GREENSPAN      MIS
IRVING         PRODUCTION
JONES          MIS
MCCOY          MIS
MCKNIGHT       PRODUCTION
ROMANS         PRODUCTION
SMITH          MIS
              PRODUCTION
*** LAST NAME SMITH DOES MATCH MASK
STEVENS        PRODUCTION
```

## Using Subroutine Calls in RECAP Commands

Subroutines may be called from Financial Modeling Language (FML) RECAP commands. The syntax is:

```
RECAP name[(n)][/format] = subroutine (input1,...,'format2');
```

Instead of a temporary field, specify the name of the calculation. You also may specify the number (*n*) of the column where you want the value displayed. If you omit the column number, the value appears in all columns.

The format of the calculation is optional; the default is D12.2. If the calculation consists of only the subroutine, make sure that the format of the subroutine output value (*format2*) agrees with the calculation's format. If the calculation format is larger than the column width, the value displays in that column as asterisks.

The input arguments for a RECAP command may include numeric constants, alphanumeric literals, row and column references (R notation, E notation, or labels) or names of other RECAP calculations.

### *Example*

#### Using a Subroutine in a RECAP Command

Suppose you have a subroutine named INVEST in your private collection of subroutines (INVEST is not available in the supplied library) and it calculates the amount on the basis of cash on hand, total assets, and the current date. In order to create a report that prints an account of company assets and calculates how much money the company has available to invest, you must create a report request that invokes the INVEST subroutine.

The current date is obtained from the &YMD system variable. The NOPRINT option beside it prevents the date from appearing in the report; the date is solely used as input for the next RECAP statement.

The request is:

```
TABLE FILE LEDGER
HEADING CENTER
"ASSETS AND MONEY AVAILABLE FOR INVESTMENT </2"
SUM AMOUNT ACROSS HIGHEST YEAR
IF YEAR EQ 1985 OR 1986
FOR ACCOUNT
1010 AS 'CASH'                                LABEL CASH      OVER
1020 AS 'ACCOUNTS RECEIVABLE'                LABEL ACR        OVER
1030 AS 'INTEREST RECEIVABLE'                LABEL ACI        OVER
1100 AS 'FUEL INVENTORY'                      LABEL FUEL       OVER
1200 AS 'MATERIALS AND SUPPLIES'             LABEL MAT        OVER
BAR
RECAP TOTCAS = CASH+ACR+ACI+FUEL+MAT; AS 'TOTAL ASSETS'
BAR
RECAP THISDATE/A8 = &YMD; NOPRINT
RECAP INVAIL = INVEST(CASH,TOTCAS,THISDATE,'D12.2'); AS
'AVAIL. FOR INVESTMENT'                      OVER
BAR AS '='
END
```

The request produces the following report:

```
PAGE      1

ASSETS AND MONEY AVAILABLE FOR INVESTMENT

              YEAR
              1986   1985
-----
CASH                2,100   1,684
ACCOUNTS RECEIVABLE    875     619
INTEREST RECEIVABLE   4,026   3,360
FUEL INVENTORY        6,250   5,295
MATERIALS AND SUPPLIES 9,076   7,754
-----
TOTAL ASSETS         22,327  18,712
-----
AVAIL. FOR INVESTMENT  3,481   2,994
=====
```

## Storing and Accessing External Subroutines

Accessing external subroutines varies by platform. The following topics describe how to access subroutines on specific platforms.

**Note:** If you have a private collection of subroutines (you created your own or use customized subroutines), do not store them in the subroutine library. Store them separately to avoid overwriting them whenever your site installs a new release.

## Storing and Accessing Subroutines on MVS

In MVS, Information Builders-supplied subroutines are stored as part of FUSELIB.LOAD. In addition to this load library, your site may have private collections of subroutines stored in separate load libraries. Load libraries are partitioned data sets containing link-edited modules.

### MVS Batch Allocation

To use subroutines stored as load libraries, allocate the load libraries to the ddname USERLIB in the FOCUS JCL or CLIST. For example, to allocate subroutines stored in BIGLIB.LOAD in JCL:

```
//USERLIB DD DISP=SHR,DSN=BIGLIB.LOAD
```

The FOCUS search order is USERLIB, STEPLIB, JOBLIB, link pack area, and linklist.

### MVS/TSO Allocation

To use these subroutines in MVS/TSO, allocate the load libraries to ddname USERLIB. Issue the ALLOCATE command: 1) in MVS/TSO before going into your FOCUS session; or 2) from FOCUS before executing your request. You may also include the command in your PROFILE FOCEXEC.

The syntax is

```
{MVS|TSO} ALLOCATE FILE(USERLIB) DSN(lib1 lib2 lib3 ...) SHR
```

where:

*MVS* or *TSO*

Specify the prefix if you issue the ALLOCATE command from FOCUS or include it in your PROFILE FOCEXEC.

*lib1...*

Are the names of the load libraries. (This concatenates the data sets to ddname USERLIB.)

**Note:**

- If you have private collections of subroutines, you need to allocate those load libraries in addition to the FUSELIB load library.
- If you are in a FOCUS session, you may also use the DYNAM ALLOCATE command to specify the allocation.

For example, to allocate the FUSELIB.LOAD load library in a FOCUS session, use either the TSO ALLOCATE or DYNAM ALLOCATE command

```
TSO ALLOC F(USERLIB) DA('prefix.FUSELIB.LOAD') SHR
```

or

```
DYNAM ALLOC FILE USERLIB DA prefix.FUSELIB.LOAD SHR
```

where *prefix* is your high-level qualifier.

As another example, suppose a report request calls two subroutines: BENEFIT stored in library SUBLIB.LOAD, and EXCHANGE stored in library BIGLIB.LOAD. FOCUS can locate user-written subroutines in ddname USERLIB; therefore, the BIGLIB library is concatenated to USERLIB. Before executing the report request, enter:

```
DYNAM ALLOC FILE USERLIB DA SUBLIB.LOAD SHR
DYNAM ALLOC FILE BIGLIB DA BIGLIB.LOAD SHR
DYNAM CONCAT FILE USERLIB BIGLIB
```

FOCUS searches the load libraries in the order that you specified them in the ALLOCATE command.

Or, for batch mode, concatenate the load library to the ddname STEPLIB or USERLIB in your JCL:

```
//FOCUS EXEC PGM=FOCUS
//STEPLIB DD DSN=FOCUS.FOCLIB.LOAD,DISP=SHR
// DD DSN=FOCUS.FUSELIB.LOAD,DISP=SHR
.
.
.
```

The FOCUS search order is: USERLIB, STEPLIB, JOBLIB, and link pack area and linklist.

## Dynamic Language Environment Support

IBM's Dynamic Language Environment (LE) enables you to use a common run-time environment for all LE-supported high-level languages (HLLs).

From a non-LE-conforming driver (such as FOCUS), you can use LE preinitialization facilities to create and initialize a common run-time environment, execute applications written in an LE-conforming HLL multiple times within the preinitialized environment, and terminate the preinitialized environment. FOCUS utilizes the CEEPIPI preinitialized interface to perform these tasks.

In the preinitialized environment, FOCUS provides support for executing subroutines multiple times.

Language Environment preinitialization is commonly used to enhance performance for repeated invocations of an application or for a complex application with many repetitive requests where fast response is required. For example, if FOCUS invokes an HLL subroutine a number of times, the creation and termination of that HLL environment multiple times is needlessly inefficient. A more efficient method is to create the HLL environment only once for use by all invocations of the routine.

The IBMLE parameter controls preinitialization for calls to subroutines from FOCUS. The following table summarizes FOCUS preinitialization support for user-written subroutines:

<b>HLL</b>	<b>Preinitialization Supported?</b>	<b>Preinitialization Interface</b>	<b>IBMLE Setting</b>
C	Yes	C assembler interface. Calls to the subroutine use a special extended parameter list.	ON
C++	No		OFF
COBOL	Yes	<p>The COBOL run-time option RTEREUS(ON) is the recommended preinitialization interface for COBOL subroutines. This interface requires IBMLE=OFF.</p> <p>FOCUS can also accommodate LE compliant COBOL subroutines with IBMLE=ON if required by site characteristics or restrictions.</p> <p>RTEREUS preinitialization and CEEPIPI preinitialization are mutually exclusive for FOCUS subroutines and cannot be used simultaneously. If used simultaneously, unpredictable results will occur.</p>	OFF  ON
FORTRAN	No		OFF
PL/I	Yes	<p>PL/I for MVS &amp; VM-defined preinitialization interfaces with calls to CEESTART or PLISTART with a special extended parameter list.</p> <p>Preinitialization services do not support PL/I multitasking applications.</p>	ON

For more information regarding the IBM Language Environment see IBM's OS/390 V2R10.0 *Language Environment for OS/390 & VM Programming Guide*, Document Number: SC28-1939-09.

## Storing and Accessing Subroutines on VM/CMS

In CMS, Information Builders-supplied subroutines are stored as:

- The load library FUSELIB LOADLIB.
- The text library FUSELIB TXTLIB. A text library is a file that is composed of multiple text files called “members.” Subroutines can be stored as members of one or more text libraries. The file type for text libraries is TXTLIB.
- Text files. The file name of a text file must match the subroutine name. The file type is TEXT. For example, the EXCHANGE subroutine stored as a text file has the file identifier (ID):

EXCHANGE TEXT

### Note:

- In addition to the FUSELIB load library, your site may have private collections of subroutines stored in separate libraries or text files.
- If you create your own subroutines in text files or text libraries, the subroutine must be 31-bit addressable and created as part of a LOADLIB.

If your request calls subroutines stored as text files, FOCUS can find the subroutines automatically. Remember, though, that you must have access to the disks where the subroutines reside.

FOCUS searches for subroutines in the standard CMS search sequence:

1. Load libraries, in the order that you specified them in the GLOBAL LOADLIB command.
2. Text files, searching attached disks in alphabetical order.
3. Text libraries, in the order that you specified them in the GLOBAL TXTLIB command.

For subroutines stored as text files in CMS, the access method is automatic. When your request calls the subroutine, FOCUS searches attached disks in alphabetical order, provided that you have proper authorization.

For subroutines stored as load or text libraries in CMS, you need to issue the CMS GLOBAL command. The GLOBAL command enables FOCUS to search specified libraries for the subroutines.

Issue the command in CMS before starting the server. You may also include the command in your server’s global profile.

**Note:** Subroutines written in languages such as COBOL and PL/I, or subroutines that call system subroutines, require that the GLOBAL command also specify a system library. FUSELIB subroutines do not require any other system libraries.



## Syntax

### How to Enable FOCUS to Search Specified Libraries for Subroutines

```
[CMS] GLOBAL {LOADLIB|TXTLIB} library1 library2 library3 ...
```

where:

CMS

Specify this prefix if you issue the GLOBAL command from a profile or stored procedure, or include it in your server's global profile.

library1...

Are the file names of the load and text libraries containing the subroutines. The maximum number of libraries is 63.

**Note:** If you have private collections of subroutines, you need to specify those libraries in the GLOBAL command in addition to the FUSELIB load library.

## Syntax

### How to List Libraries Specified by the GLOBAL Command

To list load or text libraries specified by the GLOBAL command, issue:

```
CMS QUERY {LOADLIB|TXTLIB}
```

## Example

### Using the GLOBAL COMMAND to Access Subroutines

For example, your server's global profile may contain the GLOBAL command:

```
CMS GLOBAL LOADLIB FUSELIB
```

For another example, suppose a report request calls two subroutines: BENEFIT, stored in text library SUBLIB, and EXCHANGE, stored in text library BIGLIB. Before executing the request, issue the GLOBAL command in a stored procedure or at the command line:

```
CMS GLOBAL TXTLIB SUBLIB BIGLIB
```

If you issue two GLOBAL commands, the second command replaces the first. Once a library is opened (as a result of referencing one of its members), the library cannot be changed until you exit FOCUS.

# Alphabetical List of Functions and Subroutines

The following sections describe the functions and subroutines in alphabetical order.

## ABS: Calculating Absolute Value

The ABS function returns the absolute value of its argument.

**Available on:** All platforms.

### Syntax

### How to Calculate Absolute Value

*ABS(argument)*

where:

*argument*

Numeric

Is the value on which ABS operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

### Example

### Report Request Calculating Absolute Value of Difference Between UNIT\_SOLD and DELIVER\_AMT

The following request calculates the absolute value of the difference between the number of units sold and the number delivered:

```
TABLE FILE SALES
PRINT UNIT_SOLD AND DELIVER_AMT AND
COMPUTE DIFF/I5 = DELIVER_AMT - UNIT_SOLD; AND
COMPUTE ABS_DIFF/I5 = ABS(DIFF);
BY PROD_CODE
WHERE DATE LE '1017';
END
```

The request produces the following output:

PAGE 1

PROD_CODE	UNIT_SOLD	DELIVER_AMT	DIFF	ABS_DIFF
-----	-----	-----	----	-----
B10	30	30	0	0
B17	20	40	20	20
B20	15	30	15	15
C17	12	10	-2	2
D12	20	30	10	10
E1	30	25	-5	5
E3	35	25	-10	10

## ARGLEN: Measuring Argument Length

The ARGLEN subroutine measures the argument length of a character string within a field, excluding trailing spaces. (The field format specifies the length of the field, including trailing spaces.)

Note that in Dialogue Manager, you can measure the length of prompted character strings using the .LENGTH suffix.

**Available on:** All platforms.

### Syntax

### How to Measure the Length of a Character String

`ARGLEN(inlength, infield, outfield)`

where:

*inlength*

Integer

Is the total length of the field containing the character string.

*infield*

Alphanumeric

Is the name of the field for which the argument length is to be determined.

*outfield*

Integer

Is the field to which the integer result is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Measuring Length of Employee Last Names**

The following request displays the lengths of employee last names:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
NAME_LEN/I3 = ARGLEN(15, LAST_NAME, NAME_LEN);
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

```
PAGE      1
```

LAST_NAME	NAME_LEN
-----	-----
SMITH	5
JONES	5
MCCOY	5
BLACKWOOD	9
GREENSPAN	9
CROSS	5

**ASIS: Distinguishing Between a Blank and a Zero**

The ASIS function is used in Dialogue Manager to distinguish between a blank and a zero. By using the ASIS function in Dialogue Manager, numeric string constants and variables defined as numeric strings (numerics within single quotation marks) can be differentiated from fields defined simply as numeric. The ASIS function forces FOCUS to evaluate a variable as it is entered rather than converting it to a number. It is used in Dialogue Manager equality expressions only.

**Available on:** All platforms.

**Syntax**

**How to Distinguish Between a Blank and a Zero**

```
ASIS(argument)
```

where:

```
argument
```

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. The expression may call a function or a subroutine.

If you specify an alphanumeric literal, enclose it in single quotation marks ('). If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

**Example****Distinguishing Between a Blank and a Zero**

The following requests show how the ASIS function affects the way FOCUS recognizes values. In the first example, the ASIS function is not used. FOCUS does not distinguish between variables defined as '' and 0:

```
-SET &VAR1 = ' ';
-SET &VAR2 = 0;
-IF &VAR2 EQ &VAR1 GOTO ONE;
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 NOT TRUE
-QUIT
-ONE
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 TRUE
```

The output is:

```
VAR1 EQ VAR2 0 TRUE
```

The second request shows the use of the ASIS function to distinguish between the two variables:

```
-SET &VAR1 = ' ';
-SET &VAR2 = 0;
-IF &VAR2 EQ ASIS(&VAR1) GOTO ONE;
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 NOT TRUE
-QUIT
-ONE
-TYPE VAR1 &VAR1 EQ VAR2 &VAR2 TRUE
```

The output is:

```
VAR1 EQ VAR2 0 NOT TRUE
```

## ATODBL: Converting Alphanumeric Strings to a Double-Precision Number

The ATODBL subroutine converts numbers from alphanumeric to double-precision format. You use this subroutine primarily to prepare arguments for other subroutines that you call from Dialogue Manager -CMS RUN, -TSO RUN, and -MVS RUN statements. For other applications, the EDIT function performs the same operation.

All numeric arguments in Dialogue Manager are in alphanumeric format. These arguments must be converted to double-precision format before being passed to subroutines. The -SET statements automatically convert these arguments, but -CMS RUN, -TSO RUN, and -MVS RUN statements do not.

In order to call a subroutine from an operating system -RUN statement, you must convert each numeric argument into double-precision format and store it in a Dialogue Manager variable. The variable is used in the subroutine argument list. Since the EDIT function cannot store double-precision numbers in Dialogue Manager variables, you must call the ATODBL subroutine to convert the arguments.

**Available on:** All platforms.

**Related functions and subroutines:**

- EDIT
- FTOA

Two syntaxes for the ATODBL subroutine exist.

*Procedure*

## How to Convert Alphanumeric Strings to a Double-Precision Number From an Operating System -RUN Statement

To use the ATODBL subroutine in Dialogue Manager, perform these steps:

1. Define the output variable as 8 bytes long. The syntax is

```
-SET &outfield = '12345678';
```

where:

*&outfield*

Is the output variable. The value must be eight characters, enclosed in single quotation marks.

2. Call the ATODBL subroutine from an operating system -RUN statement, not from a -SET statement. The syntax is

```
-{operating_system} RUN ATODBL, number, inlength, &outfield
```

where:

*operating\_system*

Is CMS, TSO, or MVS.

*number*

Alphanumeric

Is the number you want to convert or the variable containing the number. The number can be up to 15 bytes long. It can contain a sign and a decimal point but no other character; otherwise, the subroutine returns a 0.

*inlength*

Alphanumeric

Is the number of bytes in the *number* argument; maximum value is 15.

**Note:** This must be a character string.

*outfield*

A8

Is the predefined output variable.

**Syntax****How to Convert Alphanumeric Strings to a Double-Precision Number From a Non-Dialogue Manager Statement**

The syntax for specifying the ATODBL subroutine in other applications (except from -SET statements) is

```
ATODBL(number, inlength, outfield)
```

where:

*number*

Alphanumeric

Is the number to be converted, the field that contains the number, or a variable.

*inlength*

Alphanumeric

Is the number of bytes in the *number* argument; maximum value is 15. If you are specifying this field as a numeric constant, enclose it in single quotation marks.

*outfield*

Double-Precision

Is the name of the field that contains the double-precision number. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

**Example****MODIFY Request Converting a Prompted Alphanumeric Value Into a Double-Precision Number**

For the following example, the MISSING attribute is specified for the CURR\_SAL field in the EMPLOYEE Master File. This means that, if you do not enter a current salary value for this double-precision field, the null is interpreted as a default value, a period.

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO, SEGTYPE=S1
FIELDNAME=EMP_ID, ALIAS=EID, FORMAT=A9, $
.
.
.
FIELDNAME=CURR_SAL, ALIAS=CSAL, FORMAT=D12.2M, MISSING=ON, $
.
.
.
```

In this MODIFY procedure, the ATODBL subroutine converts the alphanumeric value from the TCSAL field to double-precision format. After you enter an employee ID and the employee's last and first names display, you are prompted to supply a current salary or the characters N/A, if one is not available. The current salary value is stored in a temporary alphanumeric field, TCSAL. The ATODBL converts the alphanumeric value and the TYPE statement displays it.

```
MODIFY FILE EMPLOYEE
COMPUTE TCSAL/A12=;
PROMPT EID
MATCH EID
ON NOMATCH REJECT
ON MATCH TYPE "EMPLOYEE <D.LAST_NAME <D.FIRST_NAME"
ON MATCH TYPE "ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE"
ON MATCH PROMPT TCSAL
ON MATCH COMPUTE
CSAL MISSING ON=IF TCSAL EQ 'N/A' THEN MISSING
                ELSE ATODBL(TCSAL,'12','D12.2');
ON MATCH TYPE "SALARY NOW <CSAL"
DATA
```

A sample execution is as follows:

```
EMPLOYEEFOCUS  A ON 11/14/96 AT 13.42.55
DATA FOR TRANSACTION  1

EMP_ID      =
071382660
EMPLOYEE STEVENS ALFRED
ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE
TCSAL      =
n/a
SALARY NOW
DATA FOR TRANSACTION  2

EMP_ID      =
112847612
EMPLOYEE SMITH MARY
ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE
TCSAL      =
45000
SALARY NOW      $45,000.00
DATA FOR TRANSACTION  3

EMP_ID      =
end
TRANSACTIONS:          TOTAL =      2  ACCEPTED=      2  REJECTED=      0
SEGMENTS:              INPUT =      0  UPDATED =      0  DELETED =      0
```



The procedure processes as:

1. For the first transaction, the procedure prompts you for an employee ID. You enter: 071382660.
2. The procedure displays the last and first name of the employee, STEVENS ALFRED.
3. Then it prompts you for a current salary. You enter: N/A.
4. It displays a period (.).
5. For the second transaction, the procedure prompts you for an employee ID. You enter: 112847612.
6. The procedure displays the last and first name of the employee, SMITH MARY.
7. Then it prompts you for a current salary. You enter: 45000.
8. It displays \$45,000.00.

## **AYM: Adding or Subtracting Months to or From Dates**

The AYM subroutine adds and subtracts months from dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT subroutine or the EDIT function.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

**Available on:** All platforms.

### **Related functions and subroutines:**

- CHGDAT
- EDIT
- AYMD

## Syntax

### How to Add or Subtract Months to or From Dates

`AYM(indate, months, outfield)`

where:

*indate*

Numeric

Is the input date in year-month format. If the date is not valid, AYM returns a 0.

*months*

Integer

Is the number of months you are adding or subtracting from the date. To subtract months, make the number negative.

*outfield*

Integer

Is the name of the field to which the resulting date in year-month format is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

#### Tip:

If the input date is in integer year-month-day format (I6YMD or I8YYMD), simply divide the date by 100 to convert to year-month format and set the result to be an integer. This causes the day portion of the date, which is now after the decimal point, to be dropped.

## Example

### Report Request Adding Six Months to Hire Date

The following request adds six months to the hire dates of employees. Note that the Compute expression converts the dates from year-month-day to year-month formats by dividing the dates by 100.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100 ;
AFTER6MONTHS/I4YM = AYM(HIRE_MONTH, 6, AFTER6MONTHS);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	HIRE_DATE	HIRE_MONTH	AFTER6MONTHS
BLACKWOOD	ROSEMARIE	82/04/01	82/04	82/10
CROSS	BARBARA	81/11/02	81/11	82/05
GREENSPAN	MARY	82/04/01	82/04	82/10
JONES	DIANE	82/05/01	82/05	82/11
MCCOY	JOHN	81/07/01	81/07	82/01
SMITH	MARY	81/07/01	81/07	82/01

## AYMD: Adding or Subtracting Days to or From Dates

The AYMD subroutine takes a valid date in the form [YY]YYMMDD and adds or subtracts a given number of days from the submitted date.

AYMD only operates on dates in year-month-day format. You can convert a date to this format using the CHGDAT subroutine or the EDIT function.

If the addition (or subtraction) of days crosses forward or back into a century, the century digits of the output year are adjusted.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

**Available on:** All platforms.

### **Related functions and subroutines:**

- CHGDAT
- EDIT
- AYM

## Syntax

## How to Add or Subtract Days to or From Dates

`AYMD(indate, days, outfield)`

where:

*indate*

Integer

Is the input date in [YY]YYMMDD format. If the date is not valid, the subroutine returns a 0. If *indate* is a field name, it must refer to a field with I6, I6YMD, I8, I8YYMD, P6, P6YMD, F6, F6YMD, D6, or D6YMD format.

*days*

Integer

Is the number of days you are adding to *indate*. To subtract days, make the number negative.

*outfield*

I6, I6YMD, I8, or I8YYMD

Is the name of the field to which the resulting date is returned. This argument can also be the format of the output value, enclosed in single quotation marks. If *indate* is a field, both fields must have the same format.

## Example

## Report Request Adding 35 Days to Hire Date

The following request adds 35 days to the hire date of employees:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
AFTER35DAYS/I6YMD = AYMD(HIRE_DATE, 35, AFTER35DAYS);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	HIRE_DATE	AFTER35DAYS
-----	-----	-----	-----
BANNING	JOHN	82/08/01	82/09/05
IRVING	JOAN	82/01/04	82/02/08
MCKNIGHT	ROGER	82/02/02	82/03/09
ROMANS	ANTHONY	82/07/01	82/08/05
SMITH	RICHARD	82/01/04	82/02/08
STEVENS	ALFRED	80/06/02	80/07/07

## BAR: Producing Bar Charts

The BAR subroutine enables you to produce horizontal bar charts. The bars, which consist of repeating characters, constitute a field with each bar as a field value. When a report request prints the field, the bars appear in the report.

**Available on:** All platforms.

### Syntax

### How to Produce Bar Charts

`BAR(barlength, infield, maxvalue, 'char', outfield)`

where:

*barlength*

Numeric

Is the maximum length of the bar in repeating characters. If this value is less than or equal to 0, the subroutine does not return a bar.

*infield*

Numeric

Is the field you wish to illustrate as a bar chart.

*maxvalue*

Numeric

Is the maximum length of a bar. This value should be greater than the maximum value stored in the input field (*infield*). If an *infield* value is larger than the *maxvalue* argument, the subroutine uses *maxvalue* and returns a bar at maximum length.

'char'

Alphanumeric

Is the repeating character that creates the bars. If the argument specifies more than one character, only the first character is used to create the bars.

*outfield*

Alphanumeric

Is the name of the field that contains the bars. This output field must be large enough to contain a bar at maximum length, as defined by the *barlength* argument. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

**Example**

**Report Request Creating a Bar Chart of CURR\_SAL**

The following request prints the salaries of employees and graphs them with a bar chart. The maximum length of a bar is 30 characters long. The 30-character bar represents a maximum salary of \$30,000 (which is \$29,700, rounded up). Each equal sign represents approximately \$1,000. The request is:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
      SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	CURR_SAL	SAL_BAR
BANNING	JOHN	\$29,700.00	=====
IRVING	JOAN	\$26,862.00	=====
MCKNIGHT	ROGER	\$16,100.00	=====
ROMANS	ANTHONY	\$21,120.00	=====
SMITH	RICHARD	\$9,500.00	=====
STEVENS	ALFRED	\$11,000.00	=====

**Example**

**Report Request Creating a Bar Chart of CURR\_SAL With Scale**

You may find it useful to print a scale over the bar chart. Consider the following request, which replaces the name of the computed field with a scale.

**Note:** If you are running this request on a platform where the default font is proportional (for example WebFOCUS), either use a non-proportional font, or issue SET STYLE=OFF before running the request.

```
SET STYLE=OFF

TABLE FILE EMPLOYEE
HEADING
"CURRENT SALARIES OF EMPLOYEES IN PRODUCTION DEPARTMENT"
"GRAPHED IN THOUSANDS OF DOLLARS"
" "
PRINT CURR_SAL AS 'CURRENT_SALARY'
AND COMPUTE
      SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
      AS
' 5 10 15 20 25 30,-----+-----+-----+-----+-----+'
BY LAST_NAME AS 'LAST_NAME'
BY FIRST_NAME AS 'FIRST_NAME'
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

```

PAGE          1

CURRENT SALARIES OF EMPLOYEES IN PRODUCTION DEPARTMENT
GRAPHED IN THOUSANDS OF DOLLARS

                                                    5   10   15   20   25
30
LAST_NAME          FIRST_NAME    CURRENT_SALARY
-----+-----+-----+-----+
-----
BANNING            JOHN              $29,700.00
=====
IRVING             JOAN              $26,862.00 =====
MCKNIGHT          ROGER             $16,100.00 =====
ROMANS            ANTHONY           $21,120.00 =====
SMITH             RICHARD           $9,500.00  =====
STEVENS           ALFRED            $11,000.00 =====
    
```

## BITSON: Determining If Bits Are On or Off

The BITSON subroutine evaluates individual bits within character strings to determine if a bit is on or off. If the bit is on, the subroutine returns a value of 1; otherwise, it returns a value of 0. This subroutine is useful in interpreting multi-punch data, where each punch conveys an item of information.

**Note:** BITSON returns different values, depending on your operating system.

**Available on:** All platforms.

### Syntax

#### How to Determine If Bits Are On or Off

`BITSON(bitnumber, infield, outfield)`

where:

*bitnumber*

Integer

Is the number of the bit to be evaluated, counting from the left-most bit in the character string (counting from 1).

*infield*

Alphanumeric

Is the character string, enclosed in single quotation marks, or the field that contains the character string. The character string is in multiple 8-bit blocks.

*outfield*

Integer

Is the name of the field that contains the value of the bit: 1 or 0. This argument can also be the format of the output value, enclosed in single quotation marks.

### Example

### Report Request Evaluating the Twenty-Fourth Bit of LAST\_NAME

This request evaluates the twenty-fourth bit of the names stored in the LAST\_NAME field.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
BIT_24/I1 = BITSON(24, LAST_NAME, BIT_24);
WHERE DEPARTMENT EQ 'MIS';
END
```

On the OS/390 platform, the request produces the following output:

```
PAGE      1
```

```
LAST_NAME      BIT_24
-----      -
SMITH          1
JONES          1
MCCOY          1
BLACKWOOD     1
GREENSPAN     1
CROSS          0
```

## BITVAL: Evaluating Bit Strings as Binary Integers

The BITVAL subroutine evaluates strings of bits within character strings. The bit strings can be any group of bits within the character string and can cross byte and word boundaries. The subroutine evaluates the bit strings as binary integers and returns the corresponding values.

**Note:** BITVAL returns different values, depending on your operating system.

**Available on:** All platforms.



## Syntax

## How to Evaluate Bit Strings

`BITVAL(infield, startbit, number, outfield)`

where:

*infield*

Alphanumeric

Is the character string or field that contains the bit string.

*startbit*

Integer

Is the number of the first bit in the bit string, counting from the left-most bit in the character string. If this argument is less than or equal to 0, the subroutine returns a value of zero (0).

*number*

Integer

Is the number of bits in the bit string. If this argument is less than or equal to 0, the subroutine returns a value of zero (0).

*outfield*

Integer

Is the name of the field that contains the integer equivalent. This argument can also be the format of the output value, enclosed in single quotation marks.

## Example

## Report Request Evaluating Bits 12 Through 20 of LAST\_NAME

This report request evaluates bits 12 through 20 of the last names stored in the field LAST\_NAME:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
STRING_VAL/I5 = BITVAL(LAST_NAME, 12, 9, 'I5');
WHERE DEPARTMENT EQ 'MIS';
END
```

On the OS/390 platform, the resulting output is:

PAGE 1

LAST_NAME	STRING_VAL
-----	-----
SMITH	332
JONES	365
MCCOY	60
BLACKWOOD	316
GREENSPAN	412
CROSS	413

## BYTVAL: Translating a Character to Its ASCII or EBCDIC Code

The BYTVAL subroutine translates characters to the ASCII or EBCDIC decimal values that represent them.

**Available on:** All platforms.

**Related functions and subroutines:**

HEXBYT

### Syntax

#### How to Translate a Character

`BYTVAL(character, outfield)`

where:

*character*

Alphanumeric

Is the input character. If you supply more than one character in this argument, the subroutine evaluates the first character. You can specify a field or amper variable that contains the character, or specify the character itself.

*outfield*

Integer

Is the name of the field to which the corresponding decimal value—an integer between 0 and 255—is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example****Dialogue Manager Procedure Returning EBCDIC Value for Prompted Character**

This Dialogue Manager request prompts for a character, then returns the corresponding number:

```
-PROMPT &CHAR. ENTER THE CHARACTER TO BE DECODED.
-SET &CODE = BYTVAl (&CHAR, 'I3');
-TYPE
-TYPE THE EQUIVALENT VALUE IS &CODE
```

Suppose you want to know the equivalent value of the exclamation point (!). A sample execution is:

```
ENTER THE CHARACTER TO BE DECODED
!

THE EQUIVALENT VALUE IS 90
>
```

The request processes as:

1. When you execute the request, it prompts:

```
ENTER THE CHARACTER TO BE DECODED
```

2. You enter an exclamation point: !.
3. The request responds:

```
THE EQUIVALENT VALUE IS 90
```

**CHGDAT: Changing Date Formats**

The CHGDAT subroutine rearranges the year, month, and day portions of dates and converts dates between long and short date formats. Long formats contain the year, month, and day; short formats contain one or two of these elements, such as year and month or just day. A format can be longer if four digits are used for the year (for example, 1987), or shorter if only the last two digits are used (for example, 87).

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

**Available on:** All platforms.

**Related functions and subroutines:**

- DA subroutines
- DATECVT
- DT subroutines

## Syntax

## How to Change Date Formats

```
CHGDAT('oldformat', 'newformat', indate, outfield)
```

where:

*oldformat*

A5

Is the format of the input date.

*newformat*

A5

Is the format of the converted date.

*indate*

Alphanumeric

Is the input date. If the date is in numeric format, change it to alphanumeric format using the EDIT function. If the input date is invalid, the subroutine returns spaces.

*outfield*

Alphanumeric or A17

Is the name of the field to which the converted date is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

### Tip:

Since CHGDAT returns the date in alphanumeric format with 17 characters, you can use the EDIT function to truncate this field to a shorter field or to convert the date to numeric format.

The date formats specified by the arguments *oldformat* and *newformat* contain the following characters in any combination:

- |      |   |
|------|---|
| D    | Days in the month (01 through 31).  |
| M    | Months in the year (01 through 12).   |
| Y[Y] | Year. One Y indicates a two-digit date (such as 94); two Y's indicate a four-digit date (such as 1994). |

If you want to spell out the month rather than use a number for the month, you can append one of the following to the format of the new date:

- |   |  |
|---|--|
| T | Displays the month as a three-letter English abbreviation. |
| X | Displays the full English name of month.                   |

Any other character in the format is ignored.

If you are converting a date from short to long format (for example, from year-month to year-month-day), the subroutine supplies the portion of the date missing in the short format, as shown in the following table:

Portion of Date Missing	Portion Supplied by the Subroutine
Day (that is, from YM to YMD)	Last day of the month.
Month (that is, from Y to YM)	The month 12 (December).
Year (that is, from MD to YMD)	The year 99.
Converting year from short to long form (that is, from YMD to YYMD)	<p>If DATEFNS=ON, the century will be determined by the 100-year window defined by DEFCENT and YRTHRESH. See Chapter 7, <i>Working With Cross-Century Dates</i> for details on DEFCENT and YRTHRESH.</p> <p>If DATEFNS=OFF, the year 19xx, where xx is the last two digits in the year.</p>

*Example*

**Report Request Converting Numeric Date to Full Name**

The following request displays the names and hire dates of employees, both in year-month-day format and in month-day-year format. The second format displays the full name of the month and the full year.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYXX', ALPHA_HIRE, 'A17');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

```
PAGE      1

LAST_NAME      FIRST_NAME    HIRE_DATE    HIRE_MDY
-----
BANNING        JOHN          82/08/01     AUGUST 01 1982
IRVING         JOAN          82/01/04     JANUARY 04 1982
MCKNIGHT       ROGER         82/02/02     FEBRUARY 02 1982
ROMANS         ANTHONY       82/07/01     JULY 01 1982
SMITH          RICHARD       82/01/04     JANUARY 04 1982
STEVENS        ALFRED        80/06/02     JUNE 02 1980
```

## CHKFMT: Checking String Format

The CHKFMT subroutine checks character strings for incorrect character types. It compares each string to a second string called a “mask,” comparing each character in the input string to the corresponding character in the mask. If all characters in the string match the characters or character types of those in the mask, CHKFMT returns the value 0. Otherwise, CHKFMT returns a value equal to the position of the first character in the string not matching the mask.

**Available on:** All platforms.

### Syntax

### How to Check String Format

```
CHKFMT(numchar, infield, 'mask', outfield)
```

where:

*numchar*

Integer

Is the number of characters you want to compare against the mask.

*infield*

Alphanumeric

Is the character string you are inspecting (enclosed in single quotation marks) or the field containing the string.

*'mask'*

Alphanumeric

Is the mask as described with the character symbols (enclosed in single quotation marks).

*outfield*

Integer

Is the name of the temporary field to which the result is returned, or the format of the output value, enclosed in single quotation marks.

Some characters in the mask are generic: they represent character types. If a character in the string is compared to one of these characters and is the same type, it matches. These generic characters are:

A	Represents any of the letters A-Z (uppercase or lowercase).
9	Represents any of the digits 0-9.
x	Represents any of the letters A-Z or digits 0-9.
\$	Represents any character.

Any other character in the mask represents only that character. For example, if the third character in the mask is the letter B, the third character in the string must be the letter B to match.

If the mask is shorter than the character string, the subroutine checks only the portion of the character string corresponding to the mask. For example, if you are using a four-character mask to test a nine-character string, only the first four characters in the string are checked; the rest are returned as a nomatch with `CHKFMT` giving the position as a result.

### Example

### Report Request Checking the Format of `EMP_ID`

The following request checks the format of `EMP_ID` against a mask for nine numeric characters, beginning with the numerals 11.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND
COMPUTE CHK_ID/I3 = CHKFMT(9, EMP_ID, '119999999', CHK_ID);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

PAGE 1

EMP_ID	LAST_NAME	CHK_ID
-----	-----	-----
071382660	STEVENS	1
119265415	SMITH	0
119329144	BANNING	0
123764317	IRVING	2
126724188	ROMANS	2
451123478	MCKNIGHT	1

**Example**

**MODIFY Request Checking the Format of EMP\_ID**

The following MODIFY procedure adds records of new employees to the EMPLOYEE data source. Each transaction begins as an employee ID that is alphanumeric with the first five characters as digits. The procedure rejects records with other characters in the employee ID. The procedure is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT
MATCH EMP_ID
  ON MATCH REJECT
  ON NOMATCH COMPUTE
    BAD_CHAR/I3 = CHKfmt (5, EMP_ID, '99999', BAD_CHAR);
  ON NOMATCH VALIDATE
    ID_TEST = IF BAD_CHAR EQ 0 THEN 1 ELSE 0;
    ON INVALID TYPE
      "BAD EMPLOYEE ID: <EMP_ID"
      "INVALID CHARACTER IN POSITION <BAD_CHAR"
  ON NOMATCH INCLUDE
  LOG INVALID MSG OFF
DATA
```

A sample execution is:

```
>
EMPLOYEEFOCUS  A ON 12/05/96 AT 15.42.03
DATA FOR TRANSACTION  1

EMP_ID      =
111w2
LAST_NAME   =
johnson
FIRST_NAME  =
greg
DEPARTMENT  =
production
BAD EMPLOYEE ID: 111W2
INVALID CHARACTER IN POSITION  4
DATA FOR TRANSACTION  2

EMP_ID      =
end
TRANSACTIONS:          TOTAL =      1  ACCEPTED=      0  REJECTED=      1
SEGMENTS:              INPUT  =      0  UPDATED =      0  DELETED =      0
>
```



The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department assignment. You enter the following data:

```
EMP_ID:          111w2
LAST_NAME:       johnson
FIRST_NAME:      greg
DEPARTMENT:      production
```

2. The procedure searches the data source for the ID 111W2. If it does not find this ID, it continues processing the transaction.
3. The CHKFMT subroutine checks the ID against the mask 99999, which represents five digits.
4. The fourth character in the ID, the letter “W”, is not a digit. The subroutine returns the value 4 to the BAD\_CHAR field.
5. The VALIDATE statement tests the BAD\_CHAR field. Since BAD\_CHAR is not equal to 0, the procedure rejects the transaction and displays a message indicating the position of the invalid character in the ID.

## CHKPCK: Validating Packed Fields

The CHKPCK subroutine validates packed fields (if they are available on your platform), checking that their values are in packed format. The subroutine prevents data exceptions that occur when requests read packed fields from files containing values that are not valid packed numbers.

**Available on:** All platforms.

## Syntax

### How to Validate Packed Fields

`CHKPCK(inlength, infield, error, outfield)`

where:

*inlength*

Numeric

Is the field length to be validated, from 1 to 16 bytes.

*infield*

Alphanumeric

Is the input field to be validated. The field is described as alphanumeric, not packed.

*error*

Numeric

Is the error code that the subroutine returns if a value is not packed. The error code is first truncated to an integer, then converted to packed. (The error code may appear on a report with a decimal point because of the format of the output field.) Choose an error code outside the range of data.

*outfield*

Packed

Is the name of the field that contains the input value if the value is packed or the error code. This argument can also be the format of the output value, enclosed in single quotation marks.

To use the CHKPCK subroutine, use these steps:

1. Make sure that the Master File (FORMAT, USAGE, and ACTUAL attributes), or the MODIFY FIXFORM statement describing the file, defines the field as alphanumeric, not packed. This does *not* change the field data, which remains packed. It enables the request to read the data without causing data exceptions.
2. Call the CHKPCK subroutine to examine the field. The subroutine returns its output to a field defined as packed. If the value it examines is a valid packed number, the subroutine returns the value; otherwise, it returns an error code.

## Example Validating Packed Data

In order to reproduce this example, you need to prepare a data source with invalid packed data. Issue this request:

```
DEFINE FILE EMPLOYEE
PACK_SAL/A8 = IF EMP_ID CONTAINS '123'
    THEN 'AAA' ELSE PCKOUT(CURR_SAL, 8, 'A8');
END

TABLE FILE EMPLOYEE
PRINT DEPARTMENT PACK_SAL BY EMP_ID
ON TABLE SAVE AS TESTPACK
END
```

The request creates the TESTPACK file that is used in this example. The file contains employee IDs, department assignments, and salaries. The salary field named PACK\_SAL is defined as alphanumeric but contains packed data. The invalid packed data is a string of three letters (AAA).

```
>
NUMBER OF RECORDS IN TABLE=      12  LINES=      12

{EBCDIC|ALPHANUMERIC} RECORD NAMED TESTPACK
FIELDNAME                ALIAS          FORMAT          LENGTH

EMP_ID                   EID           A9              9
DEPARTMENT               DPT           A10             10
PACK_SAL                  A8            A8              8

TOTAL                                27
[DCB USED WITH FILE TESTPACK IS DCB=(RECFM=FB,LRECL=00027,BLKSIZE=00540)]
>
```

Next, you must create a Master File for the TESTPACK file. If the description defines the salary field as packed, the bad values will cause data exceptions when a request reads the file. Instead, define the field as alphanumeric both in the USAGE and ACTUAL attributes:

```
FILE = TESTPACK, SUFFIX = FIX
FIELD = EMP_ID ,ALIAS = EID,FORMAT = A9 ,ACTUAL = A9 , $
FIELD = DEPARTMENT,ALIAS = DPT,FORMAT = A10,ACTUAL = A10,$
FIELD = PACK_SAL ,ALIAS = PS ,FORMAT = A8 ,ACTUAL = A8 , $
```

After you create the Master File, prepare the request to produce the report. In the DEFINE command, the CHKPCK subroutine validates the salaries and moves them from the alphanumeric field, PACK\_SAL, to the packed field, GOOD\_PACK. The GOOD\_PACK field contains either employees salaries or the error code -999. The request is:

```
DEFINE FILE TESTPACK
GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
END

TABLE FILE TESTPACK
PRINT DEPARTMENT GOOD_PACK BY EMP_ID
END
```

The request produces the following output:

```
PAGE      1

EMP_ID      DEPARTMENT      GOOD_PACK
-----      -
071382660   PRODUCTION      $11,000
112847612   MIS              $13,200
117593129   MIS              $18,480
119265415   PRODUCTION      $9,500
119329144   PRODUCTION      $29,700
123764317   PRODUCTION      -$999
126724188   PRODUCTION      $21,120
219984371   MIS              $18,480
326179357   MIS              $21,780
451123478   PRODUCTION      -$999
543729165   MIS              $9,000
818692173   MIS              $27,062
```

## CTRAN: Translating One Character to Another

The CTRAN subroutine translates one character to another. This subroutine is especially useful for changing replacement characters to unavailable characters, or to characters that are difficult to input or unavailable on your keyboard.

**Note:** This subroutine is especially useful for inputting characters that are difficult to enter in PROMPT, such as “,” and “'”. It eliminates the need to enclose entries in single quotation marks. To use this subroutine, you need to know the decimal equivalent of the characters in internal machine representation. Printable EBCDIC or ASCII characters and their decimal equivalents are listed in character charts.

**Available on:** All platforms.

### Related subroutines:

HEXBYT

## Syntax

### How to Translate One Character to Another

`CTRAN(inlen, infield, decfrm, decto, output)`

where:

*inlen*

Integer

Is the length in characters of the input string.

*infield*

Alphanumeric

Is the input string.

*decfrm*

Integer

Is the decimal value of the character to be translated.

*decto*

Integer

Is the decimal ASCII or EBCDIC value of the character to be used as a substitute for *decfrm*.

*output*

Alphanumeric

Is the name of the field that contains the resulting output string or the format of the output value enclosed in single quotation marks.

## Example

### Report Request Converting Spaces to Underscores

The following request converts blank spaces in a field containing addresses to underscores:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
ALT_ADDR/A20 = CTRAN(20, ADDRESS_LN3, 32, 95, ALT_ADDR);
BY EMP_ID
WHERE TYPE EQ 'HSM'
END
```

The request produces the following output:

PAGE 1

EMP_ID	ADDRESS_LN3	ALT_ADDR
-----	-----	-----
117593129	RUTHERFORD NJ 07073	RUTHERFORD_NJ_07073_
119265415	NEW YORK NY 10039	NEW_YORK_NY_10039__
119329144	FREEPORT NY 11520	FREEPORT_NY_11520__
123764317	NEW YORK NY 10001	NEW_YORK_NY_10001__
126724188	FREEPORT NY 11520	FREEPORT_NY_11520__
451123478	ROSELAND NJ 07068	ROSELAND_NJ_07068__
543729165	JERSEY CITY NJ 07300	JERSEY_CITY_NJ_07300
818692173	FLUSHING NY 11354	FLUSHING_NY_11354__

### Example

### MODIFY Request Inserting Accented Letter E's

This MODIFY request enables you to enter the names of new employees containing the accented letter È, as in the name Adèle Molière, for example. The equivalent EBCDIC code for an asterisk is 92, for an È, 159.

**Note:** If you are using the Hot Screen facility, disable it with SET SCREEN=OFF in order to display the accented letter È.

The request is:

```
MODIFY FILE EMPLOYEE
CRTFORM
***** NEW EMPLOYEE ENTRY SCREEN *****
" "
"ENTER EMPLOYEE'S ID: <EMP_ID"
" "
"ENTER EMPLOYEE'S FIRST AND LAST NAME"
"SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS"
" "
"FIRST_NAME: <FIRST_NAME LAST_NAME: <LAST_NAME"
" "
"ENTER THE DEPARTMENT ASSIGNMENT: <DEPARTMENT"
MATCH EMP_ID
  ON MATCH REJECT
  ON NOMATCH COMPUTE
    FIRST_NAME/A10 = CTRAN (10, FIRST_NAME, 92, 159, 'A10');
    LAST_NAME/A15 = CTRAN (15, LAST_NAME, 92, 159, 'A15');
  ON NOMATCH TYPE "FIRST_NAME: <FIRST_NAME LAST_NAME:<LAST_NAME"
  ON NOMATCH INCLUDE
DATA
END
```

A sample execution is as follows:

```
***** NEW EMPLOYEE ENTRY SCREEN *****
ENTER EMPLOYEE'S ID: 999888777
ENTER EMPLOYEE'S FIRST AND LAST NAME
SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS
FIRST_NAME: AD*LE      LAST_NAME: MOLI*RE
ENTER THE DEPARTMENT ASSIGNMENT: SALES
```

The request processes as:

1. The CRTFORM screen prompts you for an employee ID, last name, first name, and department assignment. It requests that you substitute an asterisk (\*) whenever the accented letter È appears in a name.
2. You enter the following data:  

```
EMPLOYEE ID:      999888777
FIRST_NAME:      AD*LE
LAST_NAME:       MOLI*RE
DEPARTMENT:      SALES
```
3. The procedure searches the data source for the employee ID. If it does not find it, it continues processing the request.
4. The CTRAN subroutine converts the asterisks into È's in both the first and last names (ADÈLE MOLÌÈRE).

```
***** NEW EMPLOYEE ENTRY SCREEN *****
ENTER EMPLOYEE'S ID:
ENTER EMPLOYEE'S FIRST AND LAST NAME
SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS
FIRST_NAME:      LAST_NAME:
ENTER THE DEPARTMENT ASSIGNMENT:
FIRST_NAME: ADÈLE LAST_NAME: MOLÌÈRE
```

5. The procedure stores the data in the data source.

**Note:** If you are using the Hot Screen facility, some unusual characters cannot be displayed. If Hot Screen does not support the character you chose, enter the FOCUS command

```
SET SCREEN = OFF
RETYPE
```

and redisplay the report that will appear as regular terminal output. If your terminal can display the character, the character will appear. The display of special characters depends upon your software and hardware; not all special characters may display.

### Example

### MODIFY Request Inserting Commas

This MODIFY request adds records of new employees to the EMPLOYEE data source. The PROMPT statement prompts you for data one field at a time. The CTRAN subroutine enables you to enter commas in names without having to enclose the names in single quotation marks. Instead of typing the comma, you type a semicolon, which is converted by the CTRAN subroutine into a comma. The equivalent EBCDIC code for a semicolon is 94; for a comma, 107.

The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT
MATCH EMP_ID
    ON MATCH REJECT
    ON NOMATCH COMPUTE
        LAST_NAME/A15 = CTRAN (15, LAST_NAME, 94, 107, 'A15');
    ON NOMATCH INCLUDE
DATA
```



A sample execution is as follows:

```
>
EMPLOYEEFOCUS  A ON 04/19/96 AT 16.07.29
DATA FOR TRANSACTION    1

EMP_ID      =
224466880
LAST_NAME   =
BRADLEY; JR.
FIRST_NAME  =
JOHN
DEPARTMENT  =
MIS
DATA FOR TRANSACTION    2

EMP_ID      =
end
TRANSACTIONS:          TOTAL =      1  ACCEPTED=      1  REJECTED=      0
SEGMENTS:              INPUT =      1  UPDATED =      0  DELETED =      0
>
```

The request processes as:

1. The request prompts you for an employee ID, last name, first name, and department assignment. You enter the following data:

```
EMP_ID:      224466880
LAST_NAME:   BRADLEY; JR.
FIRST_NAME:  JOHN
DEPARTMENT:  MIS
```

2. The request searches the data source for the ID 224466880. If it does not find the ID, it continues processing the transaction.
3. The CTRAN subroutine converts the semicolon in “BRADLEY; JR.” to a comma. The last name is now “BRADLEY, JR.”
4. The request adds the transaction to the data source.

This request displays the semicolon converted as a comma:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID LAST_NAME FIRST_NAME DEPARTMENT
IF EMP_ID IS 224466880
END

NUMBER OF RECORDS IN TABLE=          1  LINES=          1

PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

PAGE          1

EMP_ID      LAST_NAME      FIRST_NAME  DEPARTMENT
-----
224466880   BRADLEY, JR.   JOHN       MIS
```

## CTRFLD: Centering a Character String

The CTRFLD subroutine centers character strings within fields. The number of leading spaces is equal to or one less than the number of trailing spaces.

The CTRFLD subroutine is useful for centering the contents of a field and its report column or a heading that consists only of an embedded field. The report phrase HEADING CENTER centers each field value including trailing spaces. To center the field value without the trailing spaces, first center the value within the field using the CTRFLD subroutine.

**Available on:** All platforms.

### Related functions and subroutines:

- LJUST
- RJUST

**Note:** Use of CTRFLD in a styled report (StyleSheets feature) generally negates the effect of this feature unless the item is also styled as a centered element.

**Syntax**

**How to Center a Character String**

`CTRFLD(infield, inlength, outfield)`

where:

*infield*

Alphanumeric

Is the input field or a string enclosed in single quotation marks.

*inlength*

Integer

Is the length of the input and output fields. This argument must be greater than 0. (A length less than 0 can cause unpredictable results.)

*outfield*

Alphanumeric

Is the name of the field to which the centered output is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Centering LAST\_NAME**

The following request prints last names left-justified and centered.

**Note:** If you are running this request on a platform where the default font is proportional (for example WebFOCUS), either use a non-proportional font, or issue SET STYLE=OFF before running the request.

```
SET STYLE=OFF

TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
CENTER_NAME/A15 = CTRFLD(LAST_NAME, 15, 'A15');
WHERE DEPARTMENT EQ 'MIS'
END
```

The request produces the following output:

```
PAGE      1

LAST_NAME      CENTER_NAME
-----      -
SMITH          SMITH
JONES          JONES
MCCOY          MCCOY
BLACKWOOD      BLACKWOOD
GREENSPAN      GREENSPAN
CROSS          CROSS
```

## DA Subroutines: Converting a Date to an Integer

The DA subroutines convert dates to the corresponding number of days elapsed since December 31, 1899. By converting dates to the number of days, you can add and subtract dates and calculate the intervals between them. You can convert the results back to date format by using the DT subroutines discussed in *Report Request Finding the Day of the Week* on page 3-71.

There are six DA subroutines; each one accepts dates in a different format.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

**Available on:** All platforms.

### Related functions and subroutines:

- CHGDAT
- DATEDIF
- DT subroutines

## Syntax

### How to Convert a Date to an Integer

*subroutine(indate, outfield)*

where:

*subroutine*

Is one of the following:

*DADMY* converts dates in day-month-year format.

*DADYM* converts dates in day-year-month format.

*DAMDY* converts dates in month-day-year format.

*DAMYD* converts dates in month-year-day format.

*DAYDM* converts dates in year-day-month format.

*DAYMD* converts dates in year-month-day format.

*indate*

Numeric

Is the input date or a field that contains the date. The date is truncated to an integer before conversion. The date format is determined by the subroutine, as explained above.

To specify the year, enter only the last two digits; the subroutine assumes the century component. If the date is invalid, the subroutine returns a 0.

*outfield*

Integer

Is the name of the field to which the number of days this century is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

### Example

### Report Request Calculating the Difference Between Two Dates

The following example shows the number of days that elapse between the time employees get raises and the time they were hired:

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
DAYS_HIRED/I8 = DAYMD(DAT_INC, 'I8') - DAYMD(HIRE_DATE, 'I8');
BY LAST_NAME BY FIRST_NAME
IF DAYS_HIRED NE 0
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The example produces the following report:

PAGE 1

LAST_NAME	FIRST_NAME	RAISE DATE	DAYS_HIRED
-----	-----	-----	-----
IRVING	JOAN	82/05/14	130
MCKNIGHT	ROGER	82/05/14	101
SMITH	RICHARD	82/05/14	130
STEVENS	ALFRED	82/01/01	578
		81/01/01	213

## DATEADD: Adding or Subtracting Date Units to or From a Date

The DATEADD function adds or subtracts units to or from a date format. A unit can be any of the following:

- **Year.**
- **Month.** If you use the month unit and create invalid dates (such as February 31), DATEADD corrects them to the last day of the month. This means that adding one month to January 31 yields February 28 or February 29 (depending on whether it is a leap year), *not* February 31.
- **Day.**
- **Weekday.** If you use the weekday unit, DATEADD does not count Saturday and Sunday when adding days. This means that one weekday past a Friday is the following Monday. If your input date is a Saturday or Sunday, DATEADD adjusts it to the following Monday before performing addition or subtraction.
- **Business day.** If you use the business day unit, DATEADD uses the BUSDAYS setting and holiday file (determined by the HDAY setting) to determine which days are working days and disregards the rest. This means that if Monday is not a working day, then one business day past a Sunday is the following Tuesday.

DATEADD can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

**Note:** You can perform non day-based date calculations (for example YM, YQ) directly (+, -) without using these functions.

**Available on:** All platforms.

## Syntax

### How to Add or Subtract Date Units to or From a Date

`DATEADD(YMMDdate, 'unit', #units)`

where:

*YMMDdate*

Date

Is any day-based new date, for example, YYMD, MDY, or JUL.

*unit*

Alphanumeric

Can be one of the following:

**Y** indicates year units.

**M** indicates month units.

**D** indicates day units.

**WD** indicates weekday units. This means that DATEADD disregards Saturday and Sunday when performing calculations.

**BD** indicates business day units. This means that DATEADD uses the BUSDAYS setting and the holidays file (determined from the HDAY setting) to determine which days are working days. DATEADD disregards non-working days when performing calculations.

*#units*

Integer

Is the number of date units you wish to add or subtract to or from the day-based new date. If this number is not a whole unit, it is rounded down to the next largest integer.

## Example

### Rounding With DATEADD

The number of units passed to DATEADD is always a whole unit. For example,

`DATEADD(DATE, 'M', 1.999)`

adds one month because the number of units is less than two.

**Example**      **Using Weekday Units**

If you use weekday units and use a Saturday or Sunday as input, DATEADD adjusts the input to Monday. Thus,

```
DATEADD(Saturday, 'WD', 1)
```

and

```
DATEADD(Sunday, 'WD', 1)
```

both yield Tuesday as a result because Saturday and Sunday are not business days, so DATEADD begins with Monday and adds one, yielding Tuesday.

**Example**      **Adding Three Business Days to a Date**

In this example, DATEADD takes NEW\_DATE, in YYMD format, and adds three weekdays to it:

```
DATEADD(NEW_DATE, 'WD', 3)
```

The following table shows sample values for NEW\_DATE and DATEADD(NEW\_DATE, 'WD', 3):

<b>NEW_DATE</b>	<b>DATEADD(NEW_DATE, 'WD', 3)</b>
1982/04/01	1982/04/06
1981/11/02	1981/11/05
1982/04/01	1982/04/06
1982/05/01	1982/05/06
1981/07/01	1981/07/06
1981/07/01	1981/07/06

Notice that in some cases, DATEADD added more than three days, because otherwise the resulting date would have been on a weekend.



**Example**

**Report Request Adding Three Business Days to a Date**

The following request adds three weekdays to HIRE\_DATE:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND HIRE_DATE AND COMPUTE
NEW_DATE/YYMD=DATECVT(HIRE_DATE, 'I6YMD', 'YYMD');
HIRE_DATE_PLUS_THREE/YYMD=DATEADD(NEW_DATE, 'WD', 3);
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

This request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	HIRE_DATE	NEW_DATE	HIRE_DATE_PLUS_THREE
BLACKWOOD	ROSEMARIE	82/04/01	1982/04/01	1982/04/06
CROSS	BARBARA	81/11/02	1981/11/02	1981/11/05
GREENSPAN	MARY	82/04/01	1982/04/01	1982/04/06
JONES	DIANE	82/05/01	1982/05/01	1982/05/06
MCCOY	JOHN	81/07/01	1981/07/01	1981/07/06
SMITH	MARY	81/07/01	1981/07/01	1981/07/06

Notice that in some cases, DATEADD added more than three days, because otherwise HIRE\_DATE\_PLUS\_THREE would have been on a weekend.

**Example**

**Report Request That Determines Whether a Date Is a Business Day**

The following example uses DATEADD to determine whether a date is a business day.

To run this example you need a DATE Master File and a DATE data source.

Assume the DATE Master File is as follows:

```
FILENAME = DATE, SUFFIX=FIX,$
SEGNAME=SEG1, SEGTYPE = S0,$
FIELD = D1_YYMD, ALIAS = D1, FORMAT=YYMD,$
```

The DATE data source should have the following records:

```
19980605
19980606
```

In CMS you *must* filedef the DATE data source. For example:

```
FILEDEF DATE DISK DATE DATA A
```

In MVS, you *must* allocate the DATE data source to ddname DATE. For example:

```
DYNAM ALLOC DD DATE DA USER1.DATE.DATA SHR REU
```

The request follows:

```
SET EMPTYREPORT=ON

DEFINE FILE DATE
  X/YYMD=DATEADD(D1_YYMD, 'BD', 0);
END

TABLE FILE DATE
HEADING
" USE DATEADD TO DETERMINE WHETHER A SMARTDATE FIELD IS A BUSINESS  "
" DAY.  THE DATA SOURCE HAS THE DATE '1998/06/05' WHICH IS A FRIDAY  "
" STORED IN FIELD D1_YYMD.  AN IF TEST IS USED TO DETERMINE IF THE  "
" DATE CORRESPONDS TO A BUSINESS DAY.                                "
  PRINT D1_YYMD X
  IF X EQ '19980605'
END

TABLE FILE DATE
HEADING
" IT WILL YIELD 0 RECORDS 0 LINES IF THE RESULTING DATE IS NOT      "
" A BUSINESS DAY.  THE DATA SOURCE ALSO HAS '1998/06/05,' A SATURDAY.  "
  PRINT D1_YYMD X
  IF X EQ '19980606'
END
```

The preceding request yields the following:

```
PAGE          1

  USE DATEADD TO DETERMINE WHETHER A SMARTDATE FIELD IS A BUSINESS
  DAY.  THE DATA SOURCE HAS THE DATE '1998/06/05' WHICH IS A FRIDAY
  STORED IN FIELD D1_YYMD.  AN IF TEST IS USED TO DETERMINE IF THE
  DATE CORRESPONDS TO A BUSINESS DAY.
D1_YYMD      X
-----     -
1998/06/05  1998/06/05
```

```
PAGE          1

  IT WILL YIELD 0 RECORDS 0 LINES IF THE RESULTING DATE IS NOT
  A BUSINESS DAY.  THE DATA SOURCE ALSO HAS '1998/06/05,' A SATURDAY.
D1_YYMD      X
-----     -
```

## DATECVT: Converting Date Formats

DATECVT converts date formats within applications without requiring intermediate calculations.

DATECVT can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

**Available on:** All platforms.

### **Related functions and subroutines:**

- CHGDAT subroutine
- DA subroutines
- DT subroutines

## Syntax

### How to Convert a Date Format

```
DATECVT(indate, 'infmt', 'outfmt')
```

where:

*indate*

Date

Is the date whose format you wish to change. If you supply an invalid old date, DATECVT returns a zero value. *Indates* with old formats obey any DEFCENT and YRTHRESH values implied for that field when performing the conversion.

*infmt* and *outfmt*

Alphanumeric

Can be one of the following:

- Any new date format (for example, YYMD, YQ, M, DMY, JUL) that matches the format of *indate*. It can also be in the format of the output value enclosed within single quotes.
- Any old date format (such as I6YMD or A8MDYY).
- Non-date formats (such as I8 or A6). Non-date formats in the *infmt* parameter function as offsets from the base date of a YYMD field (12/31/1900).

The format of the field on the left side of the equal sign must match the *outfmt* value.

Invalid formats cause DATECVT to return a zero value or blank.

## Example      Converting YYMD to DMY

For example,

```
field/DMY = DATECVT(indate, 'YYMD', 'DMY');
```

If the value of *indate* is 19991231 then *field* is set to the offset, which is 311299. *Indates* with old formats obey any DEFCENT and YRTHRESH values implied for that field when performing the conversion.

## DATEDIF: Finding the Difference Between Two Dates

DATEDIF returns the difference between two dates in the form of a whole number, expressed in terms of units. A unit can be any of the following:

- **Year.**
- **Month.**
- **Day.**
- **Weekday.** If you use the weekday unit, DATEDIF does not count Saturday and Sunday when adding days. This means that the difference between Friday, December 21, 1999 and Monday, January 3, 2000, is one day.
- **Business day.** If you use the business day unit, DATEADD uses the BUSDAYS parameter and holiday file (determined by the HDAY parameter) to determine which days are working days and disregards the rest. This means that if Friday, December 31, 1999 is a holiday and Saturday and Sunday are not business days, the difference between Thursday, December 30, 1999 and Monday, January 3, 2000, is one day. See *Date Function and Subroutine Settings* on page 3-14 for more information.

DATEDIF always returns a whole number. If the difference between two dates is not a whole number, say the number of years between March 2, 1996 and March 1, 1997, DATEDIF rounds down to the next largest integer. Thus the number of years between March 2, 1996 and March 1, 1997 is 0.

If you use month units, and one or both of your input dates is the end of the month, DATEDIF takes this into account. This means that the difference between January 31 and April 30 is three months, not two months.

If the to-date is before the from-date, DATEDIF returns a negative number.

DATEDIF can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

**Note:** You can perform non day-based date calculations (for example YM, YQ) directly (+, -) without using DATEDIF.

**Available on:** All platforms.

**Related functions and subroutines:**

- DMY, MDY, and YMD subroutines
- YM subroutine

## Syntax

### How to Return the Difference Between Two Dates

```
DATEDIF(fromYYMD, toYYMD, 'unit')
```

where:

*fromYYMD*

Date

Is the starting date from which to calculate the difference.

*toYYMD*

Date

Is the ending date from which to calculate the difference.

*unit*

Alphanumeric

Can be one of the following:

**Y** indicates year units.

**M** indicates month units.

**D** indicates day units.

**WD** indicates weekday units. This means that DATEDIF disregards Saturday and Sunday when performing calculations.

**BD** indicates business day units. This means that DATEDIF uses the BUSDAYS setting and the holidays file (determined from the HDAY setting) to determine which days are working days. DATEDIF disregards non-working days when performing calculations.

## Example

### Rounding With DATEDIF

The following expression

```
DATEDIF(19960302, 19970301, 'Y')
```

returns 0 because the difference between March 2, 1996 and March 1, 1997 is less than a whole year.

## Example Using Month Calculations

Using DATEDIF with month units yields the inverse of DATEADD. If adding one month to date X creates date Y, then the count of months via DATEDIF between date X and date Y must be one month. The rule is:

If the to-date is an end-of-month then the month difference may be rounded up (in absolute terms) to guarantee the inverse rule.

The following expressions

```
DATEDIF(19990228, 19990128, 'M')
```

```
DATEDIF(19990228, 19990129, 'M')
```

```
DATEDIF(19990228, 19990130, 'M')
```

```
DATEDIF(19990228, 19990131, 'M')
```

all return a result of minus one month.

Additional examples:

```
DATEDIF(March31, May31, 'M') yields 2.
```

```
DATEDIF(March31, May30, 'M') yields 1 (because May 30 is not the end of the month).
```

```
DATEDIF(March31, April30, 'M') yields 1.
```

The same rules apply to year math, the only difference being that February 29th plus one year is equal to February 28th.

## DATEMOV: Moving Dates to a Significant Point

DATEMOV enables you to move a date to a significant point on the calendar. DATEMOV works with a date format only.

DATEMOV is affected by the BUSDAYS parameter and the holiday file (determined by the HDAY parameter). See *Date Function and Subroutine Settings* on page 3-14 for more information.

DATEMOV can help you:

- Compute payroll dates.
- Track and ship orders.
- Ensure correct credit card transactions.

**Available on:** All platforms.

## Syntax

### How to Move a Date to a Significant Point

`DATEMOV(YYMDdate, 'move-point')`

where:

*YYMDdate*

Date

Is the date you wish to move. May be any new date format as long as it implies a day component (for example MDYY, DMY, but not YM or MYY).

*move-point*

Alphanumeric

Is the significant point to which you wish to move. Permissible move-points are:

`EOM` for end of month.

`BOM` for beginning of month.

`EOQ` for end of quarter.

`BOQ` for beginning of quarter.

`EOY` for end of year.

`BOY` for beginning of year.

`EOW` for end of week.

`BOW` for beginning of week.

`NWD` for next weekday.

`NBD` for next business day (affected by BUSDAYS setting and holiday files).

`PWD` for prior weekday.

`PBD` for prior business day (affected by BUSDAYS setting and holiday files).

`WD-` for a weekday or earlier.

`BD-` for a business day or earlier (affected by BUSDAYS setting and holiday files).

`WD+` for a weekday or later.

`BD+` for a business day or later (affected by BUSDAYS setting and holiday files).

Invalid move-points result in a zero being returned.

## Example Report Request Using DATEMOV

The following DEFINE statement defines a date called ADATE, which is May 7, 1998 and calculates significant points for this date:

```
DEFINE FILE CAR
ANUM/I5 WITH COUNTRY = ANUM+1;
ADATEX/YYMD WITH COUNTRY = 19980507;
ADATE/YMD = ADATEX+ANUM;
NWD/YMDWT = DATEMOV(ADATE, 'NWD' );
PWD/YMDWT = DATEMOV(ADATE, 'PWD' );
WDP/YMDWT = DATEMOV(ADATE, 'WD+' );
WDM/YMDWT = DATEMOV(ADATE, 'WD-' );
NBD/YMDWT = DATEMOV(ADATE, 'NBD' );
PBD/YMDWT = DATEMOV(ADATE, 'PBD' );
WBP/YMDWT = DATEMOV(ADATE, 'BD+' );
WBM/YMDWT = DATEMOV(ADATE, 'BD-' );
END
```

The following command sets the business days to Monday, Tuesday, Wednesday, and Thursday:

```
SET BUSDAY = _MTWT__
```

The following TABLE request

```
TABLE FILE CAR
HEADING
"Examples of DATEMOV"
"Business days are Monday, Tuesday, Wednesday, + Thursday "
" "
"START DATE.. | MOVE POINTS....."

PRINT ADATE/WT AS 'DOW'
NWD/WT PWD/WT WDP/WT AS 'WD+' WDM/WT AS 'WD-'
NBD/WT PBD/WT WBP/WT AS 'BD+' WBM/WT AS 'BD-'
BY ADATE
END
```

yields:

Examples of DATEMOV  
Business days are Monday, Tuesday, Wednesday, + Thursday

START DATE..		MOVE POINTS.....								
ADATE	DOW	NWD	PWD	WD+	WD-	NBD	PBD	BD+	BD-	
-----	----	----	----	----	----	----	----	----	----	
98/05/08	FRI	MON	THU	FRI	FRI	TUE	WED	MON	THU	
98/05/09	SAT	TUE	THU	MON	FRI	TUE	WED	MON	THU	
98/05/10	SUN	TUE	THU	MON	FRI	TUE	WED	MON	THU	
98/05/11	MON	TUE	FRI	MON	MON	TUE	THU	MON	MON	
98/05/12	TUE	WED	MON	TUE	TUE	WED	MON	TUE	TUE	



## DECODE: Decoding Values

The DECODE function assigns values based on the value of an input field.

Many times the value of a field is a coded value. For example, the field `SEX` may have code `F` for female employees and `M` for male employees. This allows the value to be stored more efficiently (in this case, one character instead of six for female, or four for male), greatly reducing the storage requirement for the file. One method for decoding (expanding) these values is to provide a series of nested `IF ... THEN ... ELSE` phrases. For example,

```
IF SEX IS M THEN 'MALE' ELSE 'FEMALE'
```

but this can become very cumbersome and inefficient if there are numerous codes. The DECODE function facilitates the handling of codes.

There are two ways to use DECODE: you can type your values directly into the DECODE statement, or you can read your values from a separate file.

**Available on:** All platforms.

### Syntax

#### How to Decode Values

```
DECODE fieldname(code1 result1 code2 result2...[ELSE default ]);
```

where:

*fieldname*

Alphanumeric or Numeric

Is the name of the input field.

*code*

Any supported format

Is what DECODE is searching for; once it has found the specified value, it will assign the corresponding result. If the value has embedded blanks or commas, enclose it in single quotation marks.

*result*

Any supported format

Is the value to be assigned when the field has the corresponding code. If the value has embedded blanks or commas, enclose it in single quotation marks.

*default*

Any supported format

Is the value to be assigned if the code is not found among the list of codes. If this default is omitted, DECODE will assign a blank or zero for non-matching codes.

**Note:**

- You can use up to 40 lines to define the code and result pairs for any given DECODE expression. You can use either commas or blanks to separate the code from the result, or one pair from another.
- When explicitly coded in a procedure, you can use up to 40 lines of DECODE pairs; 39 if you also use an ELSE phrase.
- DECODE may give numeric results. Negative numbers must be enclosed in single quotation marks.
- Elements that contain either a comma or a blank must be enclosed in single quotation marks.

**Syntax**

**How to Decode Values in a Separate File**

```
DECODE fieldname(ddname [ELSE default ]);
```

where:

*fieldname*

Alphanumeric or Numeric

Is the name of the input field.

*ddname*

Is a logical name or a shorthand name that points to the physical file name containing the decoded values.

*default*

Any supported format

Is the value to be assigned if the code is not found among the list of codes. If this default is omitted, DECODE will assign a blank or zero for non-matching codes.

**Note:**

- Each record in the separate file is expected to contain one pair of elements separated by a comma or blanks.
- All data is interpreted in ASCII format on UNIX and Windows, or in EBCDIC format on MVS or CMS, and converted to the USAGE formats of the DECODE pairs.
- Leading and trailing blanks are ignored.
- The remainder of each record is also ignored and can be used for comments or other data. This convention is followed in all cases, except when the file name is HOLD. In that case the file is presumed to have been created by the FOCUS HOLD command, which writes fields in their internal format, and the DECODE pairs are interpreted accordingly. In this case, extraneous data in the record is ignored.

- If each record in the file consists of only one element, this element is interpreted as the code, and the result becomes either blanks or zero, as needed.

This makes it possible to use the file to hold screening literals referenced in the screening condition

```
IF field IS (filename)
```

and as a file of literals for an IF condition specified in a computational expression. For example:

```
TAKE = DECODE SELECT (filename ELSE 1);
VALUE = IF TAKE IS 0 THEN... ELSE...;
```

TAKE will be 0 for SELECT values found in the literal file and 1 in all other cases. The VALUE computation is carried out as if the expression had been:

```
IF SELECT (filename) THEN... ELSE...;
```

- When using DECODE via a file, you can have up to 32,767 characters in the file.

*Example*

### Report Request Assigning Job Categories Based on CURR\_JOBCODE

The following request uses EDIT to extract the first character of the field CURR\_JOBCODE. It then uses DECODE to create a value for the field JOB\_CATEGORY.

```
TABLE FILE EMPLOYEE
PRINT CURR_JOBCODE AND COMPUTE
DEPX_CODE/A1 = EDIT(CURR_JOBCODE, '9$$') ; NOPRINT AND COMPUTE
JOB_CATEGORY/A15 = DECODE DEPX_CODE(A 'ADMINISTRATIVE'
                                B 'DATA PROCESSING') ;

BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

```
PAGE      1

LAST_NAME      CURR_JOBCODE  JOB_CATEGORY
-----
BLACKWOOD      B04          DATA PROCESSING
CROSS          A17          ADMINISTRATIVE
GREENSPAN      A07          ADMINISTRATIVE
JONES          B03          DATA PROCESSING
MCCOY          B02          DATA PROCESSING
SMITH          B14          DATA PROCESSING
```

**Example**

**Report Request Reading DECODE Values From a File**

The following request has two parts. The first part creates a file with a list of the employee IDs for the employees who have taken classes. The second part reads this file and assigns 0 to those employees who have taken classes and 1 to those employees who have not. (Notice that the HOLD file contains only one column of values; therefore DECODE assigns the value 0 to an employee when their EMP\_ID appears in the file and 1 when EMP\_ID does not appear in the file.)

```
TABLE FILE EDUCFILE
PRINT EMP_ID
ON TABLE HOLD
END

TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND FIRST_NAME AND
COMPUTE NOT_IN_LIST/I1 = DECODE EMP_ID(HOLD ELSE 1);
WHERE DEPARTMENT EQ 'MIS';
END
```

This request produces the following output:

```
PAGE      1
```

EMP_ID	LAST_NAME	FIRST_NAME	NOT_IN_LIST
112847612	SMITH	MARY	0
117593129	JONES	DIANE	0
219984371	MCCOY	JOHN	1
326179357	BLACKWOOD	ROSEMARIE	0
543729165	GREENSPAN	MARY	1
818692173	CROSS	BARBARA	0

**DMY, MDY, YMD: Calculating the Difference Between Two Dates**

The DMY, MDY, and YMD functions calculate the difference between two dates in integer, alphanumeric, or packed format:

**Available on:** All platforms.

**Related functions and subroutines:**

- DATEDIF
- YM

**Syntax**

**How to Calculate the Difference Between Two Dates**

*function(begin, end)*

where:

*function*

Is one of the following:

**DMY** calculates the difference between two dates in day-month-year order.

**MDY** calculates the difference between two dates in month-day-year order.

**YMD** calculates the difference between two dates in year-month-day order.

*begin*

Numeric

Is the beginning date. You may supply the actual date or the name of a field that contains the date.

*end*

Numeric

Is the end date. You may supply the actual date or the name of a field that contains the date.

**Example**

**Report Request Calculating Number of Days Between Start Date and First Pay Raise**

The following request calculates the number of days between employees' start dates and their first pay raise:

```
TABLE FILE EMPLOYEE
SUM HIRE_DATE FST.DAT_INC AS 'FIRST PAY, INCREASE' AND COMPUTE
DIFF/I4 = YMD(HIRE_DATE, FST.DAT_INC) ; AS 'DAYS, BETWEEN'
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	HIRE_DATE	FIRST PAY INCREASE	DAYS BETWEEN
BLACKWOOD	ROSEMARIE	82/04/01	82/04/01	0
CROSS	BARBARA	81/11/02	82/04/09	158
GREENSPAN	MARY	82/04/01	82/06/11	71
JONES	DIANE	82/05/01	82/06/01	31
MCCOY	JOHN	81/07/01	82/01/01	184
SMITH	MARY	81/07/01	82/01/01	184

## DOWK and DOWKL: Finding the Day of the Week

The DOWK and DOWKL subroutines find the day of the week (Sunday through Saturday) of dates. The DOWK subroutine returns the day as a 3-letter abbreviation; to display the full name of the day, specify DOWKL instead.

**Available on:** All platforms.

### Syntax

#### How to Find the Day of the Week

`DOWK(indate, outfield)`

or

`DOWKL(indate, outfield)`

where:

*indate*

Numeric

Is the input date in year-month-day format. If the date is not valid, the subroutine returns spaces. If the date specifies a 2-digit year and DEFCENT and YRTHRESH values have not been set, the subroutine assumes the 20th century.

*outfield*

DOWK: A4

DOWKL: A12

Is the name of the field to which the day of the week is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example****Report Request Finding the Day of the Week**

The following request shows on which day of the week employees were hired:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND HIRE_DATE AND COMPUTE
DATED/A4 = DOWK(HIRE_DATE, DATED);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

```
PAGE      1

EMP_ID      HIRE_DATE  DATED
-----
071382660   80/06/02   MON
119265415   82/01/04   MON
119329144   82/08/01   SUN
123764317   82/01/04   MON
126724188   82/07/01   THU
451123478   82/02/02   TUE
```

**DT Subroutines: Converting an Integer to a Date**

The DT subroutines convert numbers representing the days elapsed since December 31, 1899 to corresponding dates. The DT subroutines are useful when you are performing arithmetic on dates converted to the number of days (see *DA Subroutines: Converting a Date to an Integer* on page 3-70). The DT subroutines convert the result back to date format.

There are six DT subroutines; each one converts the numbers into dates of a different format.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

**Available on:** All platforms.

**Related functions and subroutines:**

- CHGDAT
- DA subroutines
- DATEDIF

## Syntax

## How to Convert Integers to Dates

*subroutine(number, outfield)*

where:

*subroutine*

Is one of the following:

*DTDMY* converts numbers to day-month-year dates.

*DTDYM* converts numbers to day-year-month dates.

*DTMDY* converts numbers to month-day-year dates.

*DTMYD* converts numbers to month-year-day dates.

*DTYDM* converts numbers to year-day-month dates.

*DTYMD* converts numbers to year-month-day dates.

*number*

Numeric

Is the number of days. The number is truncated to an integer.

*outfield*

Integer

Is the name of the field to which the corresponding date is returned. The date format is determined by the subroutine, as explained above. This argument can also be the format of the output value, enclosed in single quotation marks.

## Example

## Report Request Converting Integer to Date

The following request takes a date that has been converted to the number of days (34650) and converts it back to the corresponding date, in month-day-year format:

```
-* THIS PROCEDURE CONVERTS HIRE_DATE, WHICH IS IN I6YMD FORMAT,  
-* TO A DATE IN I8MDYY FORMAT.  
-* FIRST IT USES THE DAYMD SUBROUTINE TO CONVERT HIRE_DATE  
-* TO A NUMBER OF DAYS.  
-* THEN IT USES THE DTMDY SUBROUTINE TO CONVERT THIS NUMBER OF  
-* DAYS TO I8MDYY FORMAT  
-*  
DEFINE FILE EMPLOYEE  
NEWF/I8 WITH EMP_ID=DAYMD(HIRE_DATE,NEWF);  
NEW_HIRE_DATE/I8MDYY WITH EMP_ID=DTMDY(NEWF,NEW_HIRE_DATE);  
END  
TABLE FILE EMPLOYEE  
PRINT HIRE_DATE NEW_HIRE_DATE  
BY FN BY LN  
WHERE DEPARTMENT EQ 'MIS'  
END
```



The request produces the following output:

```
PAGE      1

FIRST_NAME  LAST_NAME      HIRE_DATE  NEW_HIRE_DATE
-----
BARBARA     CROSS          81/11/02   11/02/1981
DIANE       JONES          82/05/01   05/01/1982
JOHN        MCCOY          81/07/01   07/01/1981
MARY        GREENSPAN      82/04/01   04/01/1982
            SMITH          81/07/01   07/01/1981
ROSEMARIE   BLACKWOOD      82/04/01   04/01/1982
```

## EDIT: Converting the Format of a Field

You can use the EDIT function to convert an alphanumeric field that contains numeric characters to numeric format, or to convert a numeric field to alphanumeric format. This is useful when you need to manipulate the field using a command or keyword that requires a particular format.

**Note:** The EDIT function also extracts characters from or adds characters to an alphanumeric string. For more information, see *Report Request Converting Numeric Date to Full Name* on page 3-55.

**Available on:** All platforms.

**Related functions and subroutines:**

FTOA

### Syntax

#### How to Convert Field Formats

```
EDIT(fieldname);
```

where:

*fieldname*

Alphanumeric or Numeric

Is the field name, enclosed in parentheses.

When you use EDIT to assign the converted value to a field, the format of the new field must correspond to the format of the returned value. For example, if you use EDIT to convert a numeric field to alphanumeric format, and then assign the resulting value to an alphanumeric field, you must give the new field an alphanumeric format as follows:

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE);
```

When converting an alphanumeric field to numeric format, a sign or decimal point in the field is accepted and is reflected in the value stored in the numeric field.

When converting a floating-point or packed-decimal field to alphanumeric format, EDIT removes the sign, the decimal point, and any number to the right of the decimal point. It then right-justifies the remaining digits and adds leading zeros to the specified field length. Also, converting a number with more than nine significant digits in floating-point or packed-decimal format may produce an incorrect result.

*Example*

**Report Request That Converts HIRE\_DATE to Alphanumeric Format**

The following request uses the CHGDAT subroutine to spell out an employee's hire date. However, CHGDAT expects its input date to be in alphanumeric format, and the HIRE\_DATE field is numeric. Therefore, this report request defines a hidden field, ALPHA\_HIRE, containing the contents of HIRE\_DATE converted to alphanumeric format. Then CHGDAT uses ALPHA\_HIRE as input.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX',ALPHA_HIRE,'A17');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS'
END
```

The request produces the following output:

```
PAGE      1
```

LAST_NAME	FIRST_NAME	HIRE_DATE	HIRE_MDY
-----	-----	-----	-----
BLACKWOOD	ROSEMARIE	82/04/01	APRIL 01 1982
CROSS	BARBARA	81/11/02	NOVEMBER 02 1981
GREENSPAN	MARY	82/04/01	APRIL 01 1982
JONES	DIANE	82/05/01	MAY 01 1982
MCCOY	JOHN	81/07/01	JULY 01 1981
SMITH	MARY	81/07/01	JULY 01 1981

## EDIT: Extracting or Adding Characters

You can use the EDIT function to extract characters from or add characters to an alphanumeric string.

If you want to use EDIT to extract characters from a string, you can also use SUBSTR. The differences are:

- The EDIT function can extract a substring from different parts of the parent string. For example, it can extract the first two characters and the last two characters of a string to form a single substring. Also, it can insert characters into a substring.
- The SUBSTR subroutine can vary the position of the substring depending on the values of other fields.

**Note:** The EDIT function also converts the format of a field. For more information, see *Report Request Converting Integer to Date* on page 3-90.

**Available on:** All platforms.

**Related functions and subroutines:**

SUBSTR

### Syntax

### How to Extract or Add Characters

```
EDIT(fieldname, 'mask');
```

where:

*fieldname*

Alphanumeric or Numeric

Is the name of the source field.

*mask*

Alphanumeric

Is a string, enclosed in single quotation marks (').

EDIT compares the characters in the mask to the characters in the source field. When it encounters a 9 in the mask, EDIT copies the corresponding character from the source field to the new field. When it encounters a \$ (dollar sign) in the mask, EDIT ignores the corresponding character in the source field. When it encounters any other character in the mask, EDIT copies that character to the corresponding position in the new field.

**Note:** To obtain the correct results, the length of the mask, excluding any characters other than 9 and \$, must be the length of the source field. In other words, the total number of 9's and \$'s must match the length of the field.

*Example*

**Report Request Extracting First Initial of FIRST\_NAME and Adding Dashes to EMP\_ID**

The following request shows how you can use masking to extract the first initial from FIRST\_NAME and add dashes to EMP\_ID. EMP\_ID has the format A9; FIRST\_NAME has the format A10. The request produces two new fields, FIRST\_INIT and EMPIDEDIT, which contain the first initial, and an employee ID with dashes added to enhance readability, respectively.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
FIRST_INIT/A1 = EDIT(FIRST_NAME, '9$$$$$$$');
EMPIDEDIT/A11 = EDIT(EMP_ID, '999-99-9999');
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output.

PAGE 1

LAST_NAME	FIRST_INIT	EMPIDEDIT
-----	-----	-----
SMITH	M	112-84-7612
JONES	D	117-59-3129
MCCOY	J	219-98-4371
BLACKWOOD	R	326-17-9357
GREENSPAN	M	543-72-9165
CROSS	B	818-69-2173

## EXP: Raising “e” to the Nth Power

The EXP subroutine raises the value “e” (approximately 2.72) to any power you specify. This subroutine is the inverse of the LOG function, which returns an argument’s logarithm.

The subroutine calculates the answer by adding terms of an infinite series. If a term adds less than .000001 percent to the sum, the subroutine ends the calculation and returns the result as a double-precision number.

**Available on:** All platforms.

**Related subroutines:**

LOG

### Syntax

#### How to Raise “e” to the Nth Power

*EXP(power, outfield)*

where:

*power*

Numeric

Is the power that “e” is being raised to.

*outfield*

Double-precision

Is the name of the field that contains the result. This argument can also be the format of the output value, enclosed in single quotation marks.

### Example

#### Raising “e” to the Nth Power

The following Dialogue Manager procedure raises “e” to the power you specify and returns the result rounded to the nearest integer (the 0.5 added to the result is a rounding constant). To determine “e” to the third power, set the &POW variable to 3.

```
-SET &POW = '3';  
-SET &RESULT = EXP(&POW, 'D15.3') + 0.5;  
-TYPE E TO THE &POW POWER IS APPROXIMATELY &RESULT
```

The result is 20:

```
E TO THE 3 POWER IS APPROXIMATELY 20
```

## EXPN: Evaluating Scientific Notation

The EXPN function evaluates an argument expressed in scientific notation.

**Available on:** All platforms.

### Syntax

### How to Evaluate Scientific Notation

`EXPN(argument)`

where:

*argument*

Is the value on which the function operates and should have the following format

`n.nn {E|D} {+|-} p`

where:

`n.nn` is a numeric constant that consists of a whole number component, followed by a decimal point, followed by a fractional component.

`{E|D}` denotes scientific notation. E and D are interchangeable.

`p` is the power of 10 to which you want to raise the number.

You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. The expression may call a function or a subroutine.

For example, you can use scientific notation to express 103 as:

`1.03E+2`

## FEXERR: Retrieving FOCUS Error Messages

The FEXERR subroutine retrieves a specified FOCUS error message. FOCUS error messages may consist of up to four lines of text:

- The first line contains the message.
- The remaining three may contain a detailed explanation, if it exists.

The subroutine retrieves the first line, the message portion. This subroutine is especially useful in procedures when the display of output messages is suppressed for a FOCUS session. Examples of commands that suppress messages are the CMS halt typing command (SET CMSTYPE HT) or the FOCUS terminal output command (SET TRMOUT=OFF).

**Available on:** All platforms.

**Syntax****How to Retrieve FOCUS Error Messages**

```
FEXERR(nnnnn, 'A72')
```

where:

*nnnnn*

Numeric

Is the FOCUS error number, up to five digits.

'A72'

Is the format of the output value, enclosed in single quotation marks, which contains the retrieved message. The maximum length of FOCUS error messages is 72 characters. For Maintain, specify the field name.

**Example****Retrieving FOCUS Error Messages**

This Dialogue Manager request initiates a global variable (&&MSGVAR) to store the retrieved message as a concatenated string, assigns the FOCUS error number 650 to a local variable (&ERR), and displays the message. The FEXERR subroutine retrieves the message for the error number specified as a variable.

The request is:

```
-SET &ERR = 650;
-SET &&MSGVAR = FEXERR(&ERR, 'A72');
-TYPE &&MSGVAR
```

When you execute this request, it displays the message for FOCUS error number 650:

```
(FOC650) THE DISK IS NOT ACCESSED
>
```

**FINDMEM: Finding a Member of a Partitioned Data Set**

The FINDMEM subroutine, used on MVS or batch only, determines if a specific member of a partitioned data set (PDS) exists. This subroutine is especially useful in Dialogue Manager procedures.

In order to use this subroutine, the PDS must be allocated to a ddname, because the ddname is specified in the subroutine call. You can search multiple partitioned data sets with one subroutine call if the partitioned data sets are concatenated to one ddname.

**Available on:** MVS.

**Related functions and subroutines:**

GETPDS

## Syntax

### How to Find a Member of a Partitioned Data Set

`FINDMEM(ddname, member, outfield)`

where:

*ddname*

A8

Is the ddname to which the PDS is allocated. This argument must be eight characters long or a variable. If you are using a literal for this argument, enclose it in single quotation marks. If it is less than eight characters, pad the literal with trailing blanks.

*member*

A8

Is the member you are searching for. This argument must be eight characters long. If you are using a literal for this argument that has less than eight characters, pad the literal with trailing blanks.

*outfield*

A1

Is the name of the field that contains the result of the search: Y, N, or E. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

The subroutine searches the PDS for a specified member and returns the letter Y, N, or E:

Y

The member exists in the PDS.

N

The member does not exist in the PDS.

E

An error occurred. This can occur for two reasons:

1. The data set is not allocated to the ddname.
2. The data set allocated to the ddname is not a PDS (and may be a sequential file).



**Example**

**Finding the Member of a Partitioned Data Set**

This Dialog Manager procedure executes a report request if the EMPLOYEE Master File exists. The FINDMEM subroutine searches the PDS allocated to ddname MASTER for the EMPLOYEE Master File. If the subroutine does not find the description, the procedure returns the appropriate message.

The procedure is:

```
-SET &FINDCODE = FINDMEM('MASTER ', 'EMPLOYEE', 'A1');
-IF &FINDCODE EQ 'N' GOTO NOMEM;
-IF &FINDCODE EQ 'E' GOTO NOPDS;
- TYPE MEMBER EXISTS, RETURN CODE = &FINDCODE
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
WHERE RECORDLIMIT EQ 4
END
-EXIT
-NOMEM
- TYPE EMPLOYEE NOT FOUND IN MASTER FILE PDS
-EXIT
-NOPDS
- TYPE ERROR OCCURRED IN SEARCH
- TYPE CHECK IF FILE IS A PDS ALLOCATED TO DDNAME MASTER
-EXIT
```

In this sample execution, the procedure finds the member and displays the report:

```
MEMBER EXISTS, RETURN CODE = Y
> NUMBER OF RECORDS IN TABLE= 4 LINES= 4

PAUSE.. PLEASE ISSUE CARRIAGE RETURN WHEN READY

PAGE 1
```

LAST_NAME	FIRST_NAME	CURR_SAL
-----	-----	-----
JONES	DIANE	\$18,480.00
SMITH	MARY	\$13,200.00
	RICHARD	\$9,500.00
STEVENS	ALFRED	\$11,000.00

## FTOA: Converting a Number to Alphanumeric Format

The FTOA subroutine converts numbers from numeric format to alphanumeric format.

The EDIT function also converts numbers from numeric to alphanumeric format, but there are differences between FTOA and EDIT:

- FTOA retains the decimal portions of numbers, whereas EDIT truncates numbers to integers.
- FTOA stores numbers right-justified with leading spaces, whereas EDIT stores numbers right-justified with leading zeros.
- FTOA enables you to add edit options to the converted number; whereas EDIT does not.
- FTOA can process any number up to 16 digits; EDIT can process any number up to nine digits and certain numbers up to ten digits. (The limit for EDIT is due to the internal representation of the number as a 4-byte integer.)

**Available on:** All platforms.

### Syntax

### How to Convert Numbers to Characters

```
FTOA(number, '(format)', outfield)
```

where:

*number*

Numeric

Is the number to be converted or the field containing the number.

'*(format)*'

Alphanumeric

Is the format of the number as it is stored in numeric format, enclosed in both single quotation marks and parentheses. Only F and D formats are supported. Include any edit options that you want to appear in the output.

**Note:** If you are using a field for this argument, specify the field name without quotation marks or parentheses. The values in the field must be enclosed in parentheses.

*outfield*

Alphanumeric

Is the name of the field to which the number in alphanumeric format is returned. This argument can also be the format of the output value, enclosed in single quotation marks. The length of this argument must be greater than the length of the *number* argument and must account for edit options and a possible negative sign. The D format automatically supplies commas.

**Example**

**Report Request Converting GROSS Salary to Alphanumeric Format**

The following request converts the GROSS salary field from decimal to alphanumeric format:

```
TABLE FILE EMPLOYEE
PRINT GROSS AND COMPUTE
ALPHA_GROSS/A14 = FTOA(GROSS, '(D12.2)', ALPHA_GROSS);
BY LAST_NAME
END
```

The request produces the following output:

```
PAGE      1
```

LAST_NAME	GROSS	ALPHA_GROSS
-----	-----	-----
BLACKWOOD	\$1,815.00	1,815.00
CROSS	\$2,255.00	2,255.00
IRVING	\$2,238.50	2,238.50
JONES	\$1,540.00	1,540.00
MCKNIGHT	\$1,342.00	1,342.00
ROMANS	\$1,760.00	1,760.00
SMITH	\$1,100.00	1,100.00
STEVENS	\$916.67	916.67

**GETPDS: Determining if a Member of a Partitioned Data Set Exists**

The GETPDS subroutine determines if a specific member of a partitioned data set (PDS) exists and, if so, returns the PDS name. This subroutine is especially useful in Dialogue Manager procedures. The FINDMEM subroutine is almost identical to the GETPDS subroutine, except that the GETPDS subroutine provides either the PDS name or different status codes.

In order to use this subroutine, the PDS must be allocated to a ddname, because the ddname is specified in the subroutine call. You can search multiple partitioned data sets with one subroutine call if the partitioned data sets are concatenated to one ddname.

**Available on:** MVS.

**Related functions and subroutines:**

FINDMEM

## Syntax

### How to Determine if a Member Exists

`GETPDS(ddname, member, outfield)`

where:

*ddname*

A8

Is the ddname to which the PDS is allocated. This argument must be eight characters long or a variable. If you are using a literal for this argument, enclose it in single quotation marks. If it is less than eight characters, pad the literal with trailing blanks.

*member*

A8

Is the member you are searching for. This argument must be eight characters long. If you are using a literal for this argument that has less than eight characters, pad the literal with trailing blanks.

*outfield*

A44

Is the name of the field that contains the result of the search. This argument must be 44 characters long, because the maximum length for a PDS name is 44. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

The subroutine searches the PDS for a specified member and returns one of four values:

*PDS name*

If the specified member exists, the PDS name that contains it.

\*D

The ddname is not assigned (allocated) to a data set.

\*M

The member does not exist in the PDS.

\*E

An error occurred. This often occurs because the data set allocated to the ddname is not a PDS (and may be a sequential file).

**Example**

**Determining if a Member Exists**

This Dialogue Manager procedure returns the name of the PDS if the EMPLOYEE Master File exists. The GETPDS subroutine searches the PDS allocated to ddname MASTER for the EMPLOYEE Master File.

```
-SET &DDNAME = 'MASTER  ' ;
-SET &MEMBER = 'EMPLOYEE' ;
-SET &PNAME = '          ' ;
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,&PNAME) ;
-IF &PNAME EQ '*D' THEN GOTO DDNOAL ;
-IF &PNAME EQ '*M' THEN GOTO MEMNOF ;
-IF &PNAME EQ '*E' THEN GOTO DDERROR ;
-*
-TYPE MEMBER &MEMBER IS FOUND IN
-TYPE THE PDS &PNAME
-TYPE ALLOCATED TO &DDNAME
-*
-EXIT
-DDNOAL
-*
-TYPE DDNAME &DDNAME NOT ALLOCATED
-*
-EXIT
-MEMNOF
-*
-TYPE MEMBER &MEMBER NOT FOUND UNDER DDNAME &DDNAME
-*
-EXIT
-DDERROR
-*
-TYPE ERROR IN GETPDS; DATA SET PROBABLY NOT A PDS.
-*
-EXIT
```

A sample execution is:

```
MEMBER EMPLOYEE IS FOUND IN
THE PDS USER1.MASTER.DATA
ALLOCATED TO MASTER
> >
```

## Example Using GETPDS With TED

In this example, the GETPDS subroutine searches for a specified member in the production MASTER.DATA partitioned data set and returns the PDS name. The DYNAM commands copy the specified member from the production PDS to your local PDS. Then, the TED editor enables you to edit the member. The ddnames are allocated earlier in the session: the production PDS is allocated to the ddname MASTER; your local PDS to ddname MYMASTER.

```
-* If the MASTER file in question is in the 'production' pds, it must
-* be copied to a 'local' pds, which has been allocated previously to the
-* ddname MYMASTER before any changes can be made.
-* Assume the MASTER in question is supplied via a -CRTFORM, with
-* a length of 8 characters, as &MEMBER
-*
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = &MEMBER;
-SET &PNAME = '                                ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,&PNAME);
-IF &PNAME EQ '*D' OR '*M' OR '*E' THEN GOTO DDERROR;
-*
DYNAM ALLOC FILE XXXX DA -
      &PNAME MEMBER &MEMBER SHR
DYNAM COPY XXXX MYMASTER MEMBER &MEMBER
-RUN
TED MYMASTER(&MEMBER)
-EXIT
-*
-DDERROR
-*
-TYPE Error in GETPDS; Check allocation for &DDNAME for
-TYPE proper allocation.
-*
-EXIT
```

Earlier in the FOCUS session, allocate the ddnames:

```
> > tso alloc f(master) da('wibfoc.p7009505.master.data') shr
> > tso alloc f(mymaster) da('user1.master.data') shr
```

After you execute the procedure, specify the EMPLOYEE member. The member is copied to your local PDS and you enter TED.

PLEASE SUPPLY VALUES REQUESTED

MEMBER= > employee

```
MYMASTER(EMPLOYEE)                SIZE=37    LINE=0

00000 * * * TOP OF FILE * * *
00001 FILENAME=EMPLOYEE, SUFFIX=FOC
00002 SEGNAME=EMPINFO,  SEGTYPE=S1
00003 FIELDNAME=EMP_ID,    ALIAS=EID,    FORMAT=A9,    $
00004 FIELDNAME=LAST_NAME, ALIAS=LN,    FORMAT=A15,   $
00005 FIELDNAME=FIRST_NAME, ALIAS=FN,    FORMAT=A10,   $
00006 FIELDNAME=HIRE_DATE, ALIAS=HDT,    FORMAT=I6YMD, $
00007 FIELDNAME=DEPARTMENT, ALIAS=DPT,    FORMAT=A10,   $
```

### Example

### Using GETPDS With Query Commands

Suppose you wanted to review the attributes of the PDS that contained a specific member. This Dialogue Manager procedure searches for the EMPLOYEE member in the PDS allocated to the ddname MASTER and, based on its existence, allocates the PDS name to the ddname TEMPMAST. Dialogue Manager MVS system variables are used to display the attributes.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '          ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,&PNAME);
-IF &PNAME EQ '*D' OR '*M' OR '*E' THEN GOTO DDERROR;
-*
DYNAM ALLOC FILE TEMPMAST DA -
      &PNAME SHR
-RUN
-? MVS DDNAME TEMPMAST
-TYPE The data set attributes include:
-TYPE Data set name is: &DSNAME
-TYPE Volume is: &VOLSER
-TYPE Disposition is: &DISP
-EXIT
-*
-DDERROR
-TYPE Error in GETPDS; Check allocation for &DDNAME for
-TYPE proper allocation.
-*
-EXIT
```

A sample execution follows:

```
> THE DATA SET ATTRIBUTES INCLUDE:  
DATA SET NAME IS: USER1.MASTER.DATA  
VOLUME IS: USERMO  
DISPOSITION IS: SHR  
>
```

When you execute this procedure, it searches the PDS allocated to ddname MASTER for the member EMPLOYEE. Since the procedure locates the member, it displays the attributes for the MASTER PDS.

## GETTOK: Getting a Token From a String

The GETTOK subroutine divides a character string where a specific character, called the delimiter, occurs in the string. It then returns one of the substrings, called a token.

For example, suppose you want to extract the fourth word from a sentence. The subroutine divides the sentence into words using spaces as delimiters, then extracts the fourth word. If the string is not divided by a delimiter character, use the PARAG subroutine.

**Available on:** All platforms.

**Related functions and subroutines:**

PARAG



## Syntax

### How to Divide a Character String

```
GETTOK(infield, inlen, toknum, 'delim', outlen, outfield)
```

where:

*infield*

Alphanumeric

Is the field containing the parent character string.

*inlen*

Integer

Is the length of the parent string. If this argument is less than or equal to 0, the subroutine returns spaces.

*toknum*

Integer

Is the number of the token you want extracted. If this argument is positive, the tokens are numbered from left to right. If this argument is negative, the tokens are numbered from the right to left (for example, an argument of -2 indicates the second to the last token in the string). If this argument is 0, the subroutine returns spaces. Leading and trailing null tokens are ignored.

*delim*

Alphanumeric

Is the delimiter character in the parent string, enclosed in single quotation marks. If you specify more than one character, only the first character is used.

*outlen*

Integer

Is the maximum size of the token. If this argument is less than or equal to 0, the subroutine returns spaces. If the token is longer than this argument, it is truncated; if it is shorter, it is padded with trailing spaces.

*outfield*

Alphanumeric

Is the name of the field to which the token is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Note:** The delimiter is not included in the token.

#### Tip:

In Dialogue Manager, to prevent the conversion of a delimiter blank character (' ') to a double precision zero, include a non-numeric character after the blank (for example, '%'). GETTOK uses only the first character (the blank) as a delimiter and the extra character (%) prevents conversion to double precision.

## Example

### Report Request Extracting Zip Code From Address

The following request uses a single blank space as a delimiter to break an address line into tokens and returns the last token, which is the zip code:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
LAST_TOKEN/A10 = GETTOK(ADDRESS_LN3, 20, -1, ' ', 10, LAST_TOKEN) ;
AS 'LAST TOKEN,(ZIP CODE)'
WHERE TYPE EQ 'HSM'
END
```

The request produces the following output:

```
PAGE      1

                LAST TOKEN
ADDRESS_LN3    (ZIP CODE)
-----
RUTHERFORD NJ 07073  07073
NEW YORK NY 10039   10039
FREEPORT NY 11520   11520
NEW YORK NY 10001   10001
FREEPORT NY 11520   11520
ROSELAND NJ 07068   07068
JERSEY CITY NJ 07300 07300
FLUSHING NY 11354   11354
```

## GETUSER: Retrieving the User ID

The GETUSER subroutine retrieves the user ID (userid) of the connected user.

In MVS FOCUS, it can also retrieve the name of an MVS batch job if you run it from the batch job. To retrieve a logon ID for MSO, use the MSOINFO subroutine described in the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*.

**Available on:** All platforms.

## Syntax

### How to Retrieve the User ID

```
GETUSER(outfield)
```

where:

*outfield*

Alphanumeric eight bytes

Is the name of the field that contains the user ID. Specify a field that is eight bytes long. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Returning User ID of Person Executing Request**

The following request returns employees' department assignments and salaries; the report heading returns the user ID of the person executing the request:

```
DEFINE FILE EMPLOYEE
USERID/A8 WITH EMP_ID = GETUSER(USERID);
END

TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES'
BY DEPARTMENT
HEADING
"SALARY REPORT RUN FROM USERID: <USERID>"
" "
END
```

The request produces the following output:

```
PAGE      1

SALARY REPORT RUN FROM USERID: USER1

DEPARTMENT    TOTAL SALARIES
-----
MIS            $108,002.00
PRODUCTION    $114,282.00
```

**GREGDT: Converting From Julian to Gregorian Format**

The GREGDT subroutine converts dates in Julian format to year-month-day format. Dates in Julian format are 5- or 7-digit numbers. The first two or four digits are the year; the last three digits are the number of the day counting from January 1. For example, January 1, 1987 in Julian format is either 87001 or 1987001, and December 31, 1987 is either 87365 or 1987365.

Depending on the format of the output, GREGDT converts Julian dates to either YMD or YYMD format, using the DEFCENT and YRTHRESH settings.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine (which only supports 5-digit input dates and produces only YMD output), change the DATEFNS setting to OFF.

**Available on:** All platforms.

**Related functions and subroutines:**

JULDAT

## Syntax

## How to Convert Julian Format Dates to Gregorian Format

`GREGDT(indate, outfield)`

where:

*indate*

Numeric

Is the Julian date, which is truncated to an integer before conversion. Each value must be a 5- or 7-digit number after truncation. The first two or four digits represent the year, the last three digits must be between 001 and 365 (366 for a leap year). If the date is invalid, the subroutine returns a 0.

*outfield*

Integer at least I6

Is the name of the field to which the date in year-month-day format is returned. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

GREGDT returns dates in the following format:

	<b>If the format is I6 or I7</b>	<b>If the format is I8 or greater</b>
<b>If DATEFNS=ON (the default)</b>	YMD	YYMD (GREGDT uses the DEFCENT and YRTHRESH settings to determine the century, if necessary).
<b>If DATEFNS=OFF</b>	YMD	YMD

## Example

**Example**

**Report Request Converting Date to Julian and Gregorian Date**

The following request converts HIRE\_DATE to both a Julian date and a Gregorian date:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND
COMPUTE JULIAN/I5 = JULDAT(HIRE_DATE, JULIAN); AND
COMPUTE GREG_DATE/I8 = GREGDT(JULIAN, 'I8');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	HIRE_DATE	JULIAN	GREG_DATE
BANNING	JOHN	82/08/01	82213	19820801
IRVING	JOAN	82/01/04	82004	19820104
MCKNIGHT	ROGER	82/02/02	82033	19820202
ROMANS	ANTHONY	82/07/01	82182	19820701
SMITH	RICHARD	82/01/04	82004	19820104
STEVENS	ALFRED	80/06/02	80154	19800602

Notice that GREGDT determines the century (using the DEFCENT and YRTHRESH settings).

## HADD: Incrementing a Date-Time Field

The HADD subroutine in an expression to increment a date-time field by a given number of units.

### Syntax

### How to Increment a Date-Time Field

```
HADD (dtfield, 'component', increment, length, 'Hformat')
```

where:

*dtfield*

Is the date-time value to increment. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*component*

Is the name of the component to be incremented, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 3-8 for a list of supported components.

*increment*

Is the number of units by which to increment the specified component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

*length*

Is the length of the returned date-time value. Valid values are:

8 for time values down to milliseconds.

10 for time values down to microseconds.

*Hformat*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

### Example

### Incrementing the Month Component of a Date-Time Field

The following adds two months to the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ADD_MONTH
-----	-----	-----
1118	2000/06/26 05:45	2000/08/26 05:45:00
1237	2000/02/05 03:30	2000/04/05 03:30:00

If necessary, the day is adjusted to be valid for the resulting month.

## HCNVRT: Converting a Date-Time Field to Alphanumeric Format

The HCNVRT subroutine converts a date-time field to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.

### Syntax

### How to Convert a Date-Time Field to Alphanumeric Format

```
HCNVRT (dtfield, '(Hfmt)', rlength, 'Ann')
```

where:

*dtfield*

Is the date-time value to convert. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*Hfmt*

Is the USAGE format of the date-time field being converted, enclosed in parentheses and single quotation marks.

*rlength*

Is the length of the alphanumeric field returned. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value. If *rlength* is smaller than the number of characters needed to display the alphanumeric field, a blank field is returned.

*Ann*

Is the USAGE format of the returned alphanumeric value, enclosed in single quotation marks.

### Example

### Converting a Date-Time Field to Alphanumeric Format

The following converts the TRANSDATE field to alphanumeric format:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME1/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
ALPHA_DATE_TIME2/A20 = HCNVRT (TRANSDATE,'(HYYMDS)', 20, 'A20');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ALPHA_DATE_TIME1	ALPHA_DATE_TIME2
-----	-----	-----	-----
1118	2000/06/26 05:45	20000626054500000	2000/06/26 05:45:00
1237	2000/02/05 03:30	20000205033000000	2000/02/05 03:30:00

## HDATE: Converting the Date Portion of a Date-Time Field to a Date Format

The HDATE subroutine extracts the date portion of a date-time field and converts it to a date format.

### Syntax

### How to Convert the Date Portion of a Date-Time Field to a Date Format

```
HDATE (dtfield, 'dateformat')
```

where:

*dtfield*

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*dateformat*

Is the USAGE format of the returned date field (for example, YYMD), enclosed in single quotation marks.

### Example

### Converting the Date Portion of the TRANSDATE Field to a Date Format

The following request converts the date portion of the TRANSDATE field to date format YYMD:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	TRANSDATE_DATE
1118	2000/06/26 05:45	2000/06/26
1237	2000/02/05 03:30	2000/02/05



## HDIFF: Finding the Number of Units Between Two Date-Time Values

You can use the HDIFF subroutine in an expression to find the number of boundaries of a given type crossed in going from date 2 to date 1.

### Syntax

### How to Find the Number of Units Between Two Date-Time Values

```
HDIFF (dtfield1, dtfield2, 'component', 'Dformat')
```

where:

*dtfield1*

Is the ending date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*dtfield2*

Is the starting date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*component*

Is the name of the component to be used in the calculation, enclosed in single quotation marks. If the unit is weeks, the WEEKFIRST setting is used in the calculation. See *Component Names and Values for Use With Date-Time Functions* on page 3-8 for a list of supported components.

*Dformat*

Is the USAGE format of the resulting number of units, enclosed in single quotation marks. The format type must be D.

### Example

### Finding the Number of Days Between Two Date-Time Fields

The following request finds the number of days between the ADD\_MONTH and TRANSDATE fields:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
DIFF_DAYS/D12.2 = HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ADD_MONTH	DIFF_DAYS
----	-----	-----	-----
1118	2000/06/26 05:45	2000/08/26 05:45:00	61.00
1237	2000/02/05 03:30	2000/04/05 03:30:00	60.00

## HDTTM: Converting a Date field to a Date-Time Field

You can use the HDTTM subroutine in an expression to convert a date field to a date-time field. The time portion is set to midnight.

### Syntax

#### How to Convert a Date field to a Date-Time Field

```
HDTTM (datefield, {8|10}, Hformat)
```

where:

*datefield*

Is the date value to be converted. You can supply the name of a date field, a date constant, or an expression that returns a date value.

*8 | 10*

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for time values down to microseconds.

*Hformat*

Is the USAGE format of the returned date-time value.

### Example

#### Converting a Date Field to a Date-Time Field

The following request converts the date field TRANSDATE\_DATE to a date-time field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	TRANSDATE_DATE	DT2
1118	2000/06/26 05:45	2000/06/26	2000/06/26 12:00AM
1237	2000/02/05 03:30	2000/02/05	2000/02/05 12:00AM

## HEXBYT: Converting a Number to a Character

The HEXBYT subroutine allows you to obtain the ASCII or EBCDIC character equivalent of a decimal integer value. This subroutine returns a single alphanumeric character in the ASCII or EBCDIC character set. You can use this subroutine to produce characters that are not on your keyboard, similar to the CTRAN subroutine.

**Note:** The display of special characters depends upon your software and hardware; not all special characters may display. Printable EBCDIC and ASCII characters and their integer equivalents are listed in character charts.

**Available on:** All platforms.

### Related functions and subroutines:

- BYTVAL
- CTRAN

### Syntax

### How to Convert a Number to a Character

`HEXBYT(input, output)`

where:

*input*

Numeric

Is the decimal value to be translated to a single character. A value greater than 255 is treated as the remainder of  $(input/256)$ .

*output*

Alphanumeric

Is the resulting alphanumeric character.

### Example

### Report Request Determining Decimal Value of Character

The following request uses BYTVAL to determine the ASCII or EBCDIC code for the first letter of LAST\_NAME, and then uses HEXBYT to convert the code back to a letter.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3');
COMPUTE LAST_INIT/A1 = HEXBYT(LAST_INIT_CODE, LAST_INIT);
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output on ASCII platforms:

```
PAGE          1

LAST_NAME          LAST_INIT_CODE  LAST_INIT
-----          -
SMITH                83      S
JONES                74      J
MCCOY                77      M
BLACKWOOD           66      B
GREENSPAN           71      G
CROSS                67      C
```

The request produces the following output on EBCDIC platforms:

```
PAGE          1

LAST_NAME          LAST_INIT_CODE  LAST_INIT
-----          -
SMITH                226     S
JONES                209     J
MCCOY                212     M
BLACKWOOD           194     B
GREENSPAN           199     G
CROSS                195     C
```

### Example

### Report Request Inserting Braces

The following request displays the names of employees and their salaries. The names of employees earning less than \$12,000 a year are enclosed in braces. The braces are produced by the HEXBYT subroutine. The integer equivalent for the left brace is 192; for the right, 208.

```
DEFINE FILE EMPLOYEE
BRACE/A17 = HEXBYT(192, 'A1') | LAST_NAME | HEXBYT(208, 'A1');
BNAME/A17 = IF CURR_SAL LT 12000 THEN BRACE
           ELSE LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT BNAME CURR_SAL BY EMP_ID
END
```

The resulting output is:

```
PAGE      1

EMP_ID      BNAME                      CURR_SAL
-----
071382660  {STEVENS              }      $11,000.00
112847612  SMITH                  }      $13,200.00
117593129  JONES                  }      $18,480.00
119265415  {SMITH                }      $9,500.00
119329144  BANNING                }      $29,700.00
123764317  IRVING                 }      $26,862.00
126724188  ROMANS                 }      $21,120.00
219984371  MCCOY                  }      $18,480.00
326179357  BLACKWOOD              }      $21,780.00
451123478  MCKNIGHT               }      $16,100.00
543729165  {GREENSPAN            }      $9,000.00
818692173  CROSS                  }      $27,062.00
```

**Note:** If you are using the Hot Screen facility, some unusual characters cannot be displayed. If Hot Screen does not support the character you chose, enter the FOCUS command

```
SET SCREEN = OFF
RETYPE
```

and redisplay the output which will appear as regular terminal output. If your terminal can display the character, the character will appear. The display of special characters depends upon your software and hardware; not all special characters may display.

## HGETC: Storing the Current Date and Time in a Date-Time Field

You can use the HGETC subroutine in an expression to store the current date and time in a date-time field. If millisecond or microsecond values are not available in your operating environment, the value returned for these components is zero.

### Syntax

#### How to Store the Current Date and Time in a Date-Time Field

```
HGETC ({8|10}, 'Hformat')
```

where:

8 | 10

Is the length of the returned date-time value. Use 8 for time values down to milliseconds, 10 for input time values down to microseconds.

*Hformat*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

### Example

## Storing the Current Date and Time in a Date-Time Field

The following request stores the current date and time in field DT2:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DT2/HYYMDm = HGETC(10, 'HYYMDm');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	DT2
-----	-----	---
1118	2000/06/26 05:45	2000/10/03 15:34:24.000000
1237	2000/02/05 03:30	2000/10/03 15:34:24.000000

## HHMMSS: Returning the Current Time

The HHMMSS subroutine retrieves the current time from the system. It returns the time as an eight-character string with embedded periods separating the hours, minutes, and seconds.

### Note:

- &TOD returns the current time of day.
- Compiled MODIFY procedures must use the HHMMSS subroutine to obtain the time; they cannot use the &TOD variable. The &TOD variable is made current only when you execute a MODIFY, SCAN, or FSCAN procedure.

**Available on:** All platforms.

### Syntax

## How to Retrieve the Current Time

```
HHMMSS(outfield)
```

where:

*outfield*

Alphanumeric

Is the name of the field to which the time (in HH.MM.SS format) is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

## Example Report Request Displaying Current Time

The following request retrieves the current time and displays it in a report footing:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES' AND COMPUTE
NOWTIME/A8 = HHMMSS(NOWTIME); NOPRINT
BY DEPARTMENT
FOOTING
"SALARY REPORT RUN AT TIME <NOWTIME"
END
```

The request produces the following output:

```
PAGE      1

DEPARTMENT  TOTAL SALARIES
-----
MIS          $108,002.00
PRODUCTION  $114,282.00

SALARY REPORT RUN AT TIME 15.21.14
```

## HINPUT: Converting an Alphanumeric String to a Date-Time Value

The HINPUT subroutine converts an alphanumeric string to a date-time value.

### Syntax How to Convert and Alphanumeric String to a Date-Time Value

```
HINPUT (inputlength, 'inputstring', length, 'Hfmt')
```

where:

*inputlength*

Is the length of the alphanumeric string to convert. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

*inputstring*

Is the alphanumeric string to convert. You can supply the actual string enclosed in single quotation marks, the name of an alphanumeric field, or an expression that returns an alphanumeric value. The alphanumeric string can consist of any valid date-time input value as described in the *Describing Data* manual.

*length*

Is the length of the returned date-time value. Valid values are:

8 for time values down to milliseconds.

10 for time values down to microseconds.

*Hfmt*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

**Example**

**Converting an Alphanumeric String to a Date-Time Value**

The following request converts the TRANSDATE field to alphanumeric format (using the HCNVRT function) and then uses the HINPUT routine to convert the alphanumeric string to a date-time value:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ALPHA_DATE_TIME1	ALPHA_DATE_TIME2
1118	2000/06/26 05:45	20000626054500000	2000/06/26 05:45:00
1237	2000/02/05 03:30	20000205033000000	2000/02/05 03:30:00

**HMIDNT: Setting the Time Portion of a Date-Time Field to Midnight**

The HMIDNT subroutine changes the time portion of a date-time field to midnight (all zeroes). This function can be used for testing date-time fields for a given date.

**Syntax**

**How to Set the Time Portion of a Date-Time Field to Midnight**

```
HMIDNT (dtfield, length, 'Hformat')
```

where:

*dtfield*

Is date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*length*

Is the length of the returned date-time value. Valid values are:

- 8 for time values down to milliseconds.
- 10 for time values down to microseconds.

*Hformat*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.



**Example**      **Setting the Time to Midnight**

The following request sets the time portion of the TRANSDATE field to midnight in both the 24- and 12-hour systems:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_MID_24/HYYMDS = HMIDNT(TRANSDATE, 8, 'HYYMDS');
TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	TRANSDATE_MID_24	TRANSDATE_MID_12
-----	-----	-----	-----
1118	2000/06/26 05:45	2000/06/26 00:00:00	2000/06/26 12:00:00AM
1237	2000/02/05 03:30	2000/02/05 00:00:00	2000/02/05 12:00:00AM

**HNAME: Extracting a Date-Time Component in Alphanumeric Format**

The HNAME subroutine extracts a specified component from a date-time field and returns it in alphanumeric format.

**Syntax**      **How to Extract a Date-Time Component in Alphanumeric Format**

```
HNAME (dtfield, 'component', Aformat)
```

where:

*dtfield*

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*component*

Is the name of the component to be extracted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 3-8 for a list of supported components.

*Aformat*

Is the alphanumeric USAGE format of the returned component, enclosed in single quotation marks. All other components are converted to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

**Example**

**Extracting the Day Component in Alphanumeric Format From a Date-Time Field**

The following request extracts the day in alphanumeric format from the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/A2 = HNAME(TRANSDATE, 'DAY', 'A2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	DAY_COMPONENT
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	05

**Example**

**Extracting the Week Component With Different WEEKFIRST Settings**

The following request extracts the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST setting changes the value of the extracted component:

```
SET WEEKFIRST = 7
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	WEEK_COMPONENT
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	06

Running the same request setting WEEKFIRST to 3 produces the following output (see Chapter 1, *Customizing Your Environment*):

CUSTID	DATE-TIME	WEEK_COMPONENT
1118	2000/06/26 05:45	25
1237	2000/02/05 03:30	05

## HPART: Returning a Date-Time Component in Numeric Format

The HPART subroutine extracts a specified component from a date-time field and returns it in numeric format.

### Syntax

### How to Return a Date-Time Component in Numeric Format

```
HPART (dtfield, 'component', 'Iformat')
```

where:

*dtfield*

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*component*

Is the name of the component to be extracted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 3-8 for a list of supported components.

*Iformat*

Is the integer USAGE format of the returned component, enclosed in single quotation marks. The year is always four digits, and the hour assumes the 24-hour system.

### Example

### Extracting the Day Component in Numeric Format From a Date-Time Field

The following request extracts the day in integer format from the TRANSDATE field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/I2 = HPART(TRANSDATE, 'DAY', 'I2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	DAY_COMPONENT
-----	-----	-----
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	5

## HSETPT: Inserting a Component Into a Date-Time Field

The HSETPT subroutine inserts the numeric value of a specified component into a date-time field.

### Syntax

### How to Insert a Component Into a Date-Time Field

```
HSETPT (dtfield, 'component', value, length, 'Hformat')
```

where:

*dtfield*

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*component*

Is the name of the component to be inserted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 3-8 for a list of supported components.

*value*

Is the numeric value to use for the requested component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

*length*

Is the length of the returned date-time value. Valid values are:

8 for time values down to milliseconds.

10 for time values down to microseconds.

*Hformat*

Is the USAGE format of the returned date-time value, enclosed in single quotation marks.

### Example

### Inserting the Day Component Into a Date-Time Field

The following request inserts the day into the ADD\_MONTH field:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	ADD_MONTH	INSERT_DAY
-----	-----	-----	-----
1118	2000/06/26 05:45	2000/08/26 05:45:00	2000/08/28 05:45:00
1237	2000/02/05 03:30	2000/04/05 03:30:00	2000/04/28 03:30:00

## HTIME: Converting the Time Portion of a Date-Time Field to a Number

The HTIME subroutine converts the time portion of a date-time field to a numeric number of milliseconds (if the first argument is 8) or microseconds (if the first argument is 10). For microseconds, the input date-time field must be a 10-byte field.

### Syntax

### How to Convert the Time Portion of a Date-Time Field to a Number

`HTIME (length, dtfield, 'Dformat')`

where:

*length*

Is the length of the input date-time value. Valid values are:

8 for time values down to milliseconds.

10 for input time values down to microseconds.

*dtfield*

Is the date-time value to use for extracting the time. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

*Dformat*

Is the USAGE format of the returned number of milliseconds or microseconds, enclosed in single quotation marks.

### Example

### Converting the Time Portion of a Date-Time Field to a Number

The following request converts time portion of the TRANSDATE field to a number of milliseconds:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

CUSTID	DATE-TIME	MILLISEC
-----	-----	-----
1118	2000/06/26 05:45	20,700,000.00
1237	2000/02/05 03:30	12,600,000.00

## IMOD, FMOD, and DMOD: Calculating the Remainder From a Division

The MOD subroutines calculate the remainder from a division. There are three MOD subroutines:

- IMOD returns the remainder as an integer.
- FMOD returns the remainder as a floating-point number.
- DMOD returns the remainder as a decimal number.

The three subroutines use the formula:

```
remainder = dividend - INT(dividend/divisor) * divisor
```

**Available on:** All platforms.

**Related functions and subroutines:**

INT

### Syntax

### How to Calculate the Remainder From a Division

```
subroutine(dividend, divisor, outfield)
```

where:

*subroutine*

Is one of the following:

IMOD returns the remainder as an integer.

FMOD returns the remainder as a floating-point number.

DMOD returns the remainder as a decimal number.

*dividend*

Numeric

Is the dividend.

*divisor*

Numeric

Is the divisor.

*outfield*

Numeric

Is the name of the field to which the remainder is returned. Remember that the subroutine name (IMOD, FMOD, or DMOD) determines the format. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example****Report Request Extracting Last Three Digits From Account Number**

The following request extracts the last three digits from the employee bank account numbers by dividing by 1000 and finding the remainder:

```
TABLE FILE EMPLOYEE
PRINT ACCTNUMBER AND
COMPUTE LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
BY LAST_NAME BY FIRST_NAME
WHERE ( ACCTNUMBER NE 000000000) AND (DEPARTMENT EQ 'MIS');
END
```

The request produces the following output:

```
PAGE      1

LAST_NAME      FIRST_NAME  ACCTNUMBER  LAST3_ACCT
-----
BLACKWOOD      ROSEMARIE  122850108   108
CROSS          BARBARA    163800144   144
GREENSPAN      MARY       150150302   302
JONES          DIANE      040950036   036
MCCOY          JOHN       109200096   096
SMITH          MARY       027300024   024
```

**INT: Finding the Greatest Integer**

The INT function returns the integer part of its argument.

**Available on:** All platforms.

**Related functions and subroutines:**

IMOD, FMOD, and DMOD

**Syntax****How to Calculate the Greatest Integer**

```
INT(argument)
```

where:

*argument*

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

**Example**

**Report Request Calculating the Greatest Integer in DED\_AMT**

The following request calculates the greatest integer in the DED\_AMT field:

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND COMPUTE
INT_DED_AMT/I11=INT(DED_AMT);
BY LAST_NAME BY FIRST_NAME
WHERE (DEPARTMENT EQ 'MIS') AND (PAY_DATE EQ 820730);
END
```

The request produces the following output:

```
PAGE      1
```

LAST_NAME	FIRST_NAME	DED_AMT	INT_DED_AMT
-----	-----	-----	-----
BLACKWOOD	ROSEMARIE	\$1,261.40	1261
CROSS	BARBARA	\$1,668.69	1668
GREENSPAN	MARY	\$127.50	127
JONES	DIANE	\$725.34	725
SMITH	MARY	\$334.10	334

**ITONUM: Converting Large Binary Integers to Double-Precision**

The ITONUM subroutine converts large binary integers in non-FOCUS files to double-precision format. Some programming languages and some non-FOCUS data storage systems use large binary integer formats. Large binary integers (more than 4 bytes in length) are not supported in the Master File syntax and, therefore, require conversion to double-precision format.

The ITONUM subroutine processes a large byte binary format input string and converts it to a double-precision number. The user specifies how many of the rightmost bytes in the input string are significant. The output of ITONUM is an 8-byte double-precision field.

**Available on:** All platforms.

**Related functions and subroutines:**

ITOPACK



**Syntax****How to Convert Large Binary Integers to Double-Precision**

```
ITONUM(sigbytes, infield, outfield)
```

where:

*sigbytes*

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign. Valid values are:

- 5 The left-most 3 bytes are ignored.
- 6 The left-most 2 bytes are ignored.
- 7 The left-most byte is ignored.

*infield*

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats must be A8.

*outfield*

Numeric

Is the name of the field that contains the double-precision number. This argument can also be the format of the output value, enclosed in single quotation marks. The format must be specified as Dn or Dn.d.

**Example****Report Request Converting a Large Binary Integer to Double-Precision**

Suppose a binary number in an external file has the following COBOL format:

```
PIC 9(8)V9(4) COMP
```

It is defined in the EUROCAR Master File as a field called BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The field can be converted to a double-precision number using the following request:

```
DEFINE FILE EUROCAR
MYFLD/D12.2 = ITONUM(6, BINARYFLD, MYFLD);
END
TABLE FILE EUROCAR
PRINT MYFLD BY CAR
END
```

## ITOPACK: Converting Large Binary Integers to Packed-Decimal Format

The ITOPACK subroutine converts large binary integers in non-FOCUS files to packed-decimal format. Some programming languages and some non-FOCUS data storage systems use double-word binary integer formats. These are similar to the single-word binary integers used by FOCUS, but they allow larger numbers. Large binary integers (more than 4 bytes in length) are not supported in the Master File syntax and, therefore, require conversion to packed format.

The ITOPACK subroutine processes an 8-byte binary format input string and converts it to a packed number. The user specifies how many of the rightmost bytes in the input string are significant. The output of ITOPACK is an 8-byte packed field of up to 15 significant numeric positions (for example, P15 or P16.d).

**Available on:** All platforms.

**Related functions and subroutines:**

ITONUM

### Syntax

### How to Convert Large Binary Integers to Packed-Decimal Format

*ITOPACK(sigbytes, infield, outfield)*

where:

*sigbytes*

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign. Valid values are:

- 5 Up to 11 significant positions (the first 3 bytes are ignored).
- 6 Up to 14 significant positions (the first 2 bytes are ignored).
- 7 Up to 15 significant positions (the first byte is ignored).

*infield*

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats must be A8.

*outfield*

Numeric

Is the name of the field that contains the packed number. This argument can also be the format of the output value, enclosed in single quotation marks. The format must be specified as Pn or Pn.d.

**Note:** The only restriction is that for a field defined as ‘PIC 9(15) COMP’ or the equivalent (15 significant digits), the maximum number that can be translated is 167,744,242,712,576.

### *Example*

## Report Request Converting a Large Binary Integer to Packed-Decimal Format

Suppose a binary number in an external file has the following COBOL format:

```
PIC 9(8)V9(4) COMP
```

It is defined to FOCUS in the EUROCAR Master File as a field called BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The field can be converted to a packed number using the following request:

```
DEFINE FILE EUROCAR  
PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, PACKFLD);  
END  
TABLE FILE EUROCAR  
PRINT PACKFLD BY CAR  
END
```

## IT0Z: Converting to Zoned Format

The IT0Z subroutine converts numbers in numeric format to zoned format. Although FOCUS does not process zoned numbers, FOCUS requests can write zoned fields to extract files for use by external programs.

**Available on:** All platforms.

### Syntax

### How to Convert to Zoned Format

*IT0Z(outlength, number, outfield)*

where:

*outlength*

Numeric

Is the length of the zoned number in bytes, up to 15 bytes. The last byte includes the sign.

*number*

Numeric

Is the number to be converted or the field that contains the number. The number is truncated to an integer before it is converted.

*outfield*

Alphanumeric

Is the name of the field that contains the zoned number. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Converting to Zoned Format**

The following request prepares an extract file containing employee IDs and salaries in zoned format for a COBOL program. The request is:

```
DEFINE FILE EMPLOYEE
ZONE_SAL/A8 = ITOZ(8, CURR_SAL, ZONE_SAL);
END
TABLE FILE EMPLOYEE
PRINT ZONE_SAL BY EMP_ID
ON TABLE SAVE AS SALARIES
END
```

The resulting extract file is:

```
NUMBER OF RECORDS IN TABLE=      12  LINES=      12

EBCDIC RECORD NAMED SALARIES
FIELDNAME                ALIAS          FORMAT          LENGTH

EMP_ID                    EID          A9              9
ZONE_SAL                  A8          A8              8

TOTAL                      17
DCB USED WITH FILE SALARIES IS DCB=(RECFM=FB,LRECL=00017,BLKSIZE=00340)
>
```

**JULDAT: Converting From Gregorian to Julian Format**

The JULDAT subroutine converts dates from year-month-day format to Julian (year-day) format. Dates in Julian format are 5- or 7-digit numbers. The first two or four digits are the year, the last three digits are the number of the day counting from January 1. For example, January 1, 1987 in Julian format is either 87001 or 1987001, and December 31, 1987 is 87365 or 1987365.

Depending on the format of the output, JULDAT converts dates to either YYNNN or YYYYNNN format, using the DEFCENT and YRTHRESH settings.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine (which only produces 5-digit dates), change the DATEFNS setting to OFF.

**Available on:** All platforms.

**Related functions and subroutines:**

GREGDT

## Syntax

## How to Convert a Gregorian Date to a Julian Date

`JULDAT(indate, outfield)`

where:

*indate*

Numeric

Is the date or field containing the date in year-month-day format (YMD or YYMD).

*outfield*

Integer at least I5

Is the field to which the Julian date is returned. This argument can also be the format of the output value, enclosed in single quotation marks (I5 or I7). For Maintain, specify the field name.

JULDAT returns dates in the following format:

	If the format is I5 or I6	If the format is I7 or greater
<b>If DATEFNS=ON (the default)</b>	YYNNN	YYYYNNN (JULDAT uses the DEFCENT and YRTHRESH settings to determine the century, if necessary).
<b>If DATEFNS=OFF</b>	YYNNN	YYNNN

## Example

### Report Request Converting Gregorian Date to Julian Date

The following request prints employee names and hire dates, in both year-month-day and Julian formats for the PRODUCTION department:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND
COMPUTE JULIAN/I7 = JULDAT(HIRE_DATE, JULIAN);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

```
PAGE      1
LAST_NAME      FIRST_NAME  HIRE_DATE  JULIAN
-----
BANNING        JOHN        82/08/01  1982213
IRVING         JOAN        82/01/04  1982004
MCKNIGHT       ROGER       82/02/02  1982033
ROMANS         ANTHONY     82/07/01  1982182
SMITH          RICHARD     82/01/04  1982004
STEVENS        ALFRED      80/06/02  1980154
```

Notice that JULDAT determines the century (using the DEFCENT and YRTHRESH settings).

## LAST: Retrieving the Preceding Value

The LAST function retrieves the preceding value selected for a field.

**Available on:** All platforms.

### Syntax

### How to Retrieve the Preceding Value

*LAST fieldname*

where:

*fieldname*

Alphanumeric or Numeric

Is the field name.

The effect of the keyword LAST depends on whether it appears in a DEFINE or COMPUTE. In a DEFINE, the LAST value is that of the previous record retrieved from the file before sorting takes place. In a COMPUTE, the LAST value is that of the record in the previous line in the report.

LAST cannot be used with -SET in Dialogue Manager.

### Example

### Report Request Displaying Running Total of Current Salaries by Department

The following request produces a running total of the CURR\_SAL field within departments. It uses LAST to determine whether the previously retrieved value of DEPARTMENT equals the current value. If the values are equal, CURR\_SAL is added to RUN\_TOT. If the values are different, the department has changed and RUN\_TOT starts with the value of the first CURR\_SAL in the new department.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME CURR_SAL AND COMPUTE
RUN_TOT/D12.2M = IF DEPARTMENT EQ LAST DEPARTMENT THEN
                (RUN_TOT + CURR_SAL) ELSE CURR_SAL ;
AS 'RUNNING,TOTAL,SALARY'
BY DEPARTMENT SKIP-LINE
END
```

The request produces the following output:

PAGE 1

DEPARTMENT	LAST_NAME	CURR_SAL	RUNNING TOTAL SALARY
-----	-----	-----	-----
MIS	SMITH	\$13,200.00	\$13,200.00
	JONES	\$18,480.00	\$31,680.00
	MCCOY	\$18,480.00	\$50,160.00
	BLACKWOOD	\$21,780.00	\$71,940.00
	GREENSPAN	\$9,000.00	\$80,940.00
	CROSS	\$27,062.00	\$108,002.00
PRODUCTION	STEVENS	\$11,000.00	\$11,000.00
	SMITH	\$9,500.00	\$20,500.00
	BANNING	\$29,700.00	\$50,200.00
	IRVING	\$26,862.00	\$77,062.00
	ROMANS	\$21,120.00	\$98,182.00
	MCKNIGHT	\$16,100.00	\$114,282.00

## LCWORD: Converting Letters in a Word to Mixed Case

The LCWORD subroutine converts the letters in the given string to mixed case. The subroutine converts to lowercase every alphanumeric character except:

- The first letter of each new word.
- The first letter after a single or double quotation mark. For example, O'CONNOR is converted to O'Connor and JACK'S to Jack'S (not Jack's).

The rest of the word is converted to lowercase. The result is a word with an initial uppercase character followed by lowercase characters.

If the subroutine encounters a number in the string, the subroutine treats it as an uppercase character and continues to convert the following alphabetic characters to lowercase.

**Available on:** All platforms.

**Related functions and subroutines:**

- LOCASE
- UPCASE



**Syntax**

**How to Convert Letters to Mixed Case**

`LCWORD(inlength, infield, outfield)`

where:

*inlength*

Integer

Is the length of the input field.

*infield*

Alphanumeric

Is the name of the input field or the input string enclosed in single quotation marks.

*outfield*

Alphanumeric

Is the name of the output field. The length of the outfield must be greater than or equal to the length of the infield.

**Example**

**Report Request Converting LAST\_NAME to Mixed Case**

The following request converts LAST\_NAME values, which are all uppercase, to mixed case:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
MIXED_CASE/A15 = LCWORD(15, LAST_NAME, MIXED_CASE) ;
WHERE DEPARTMENT EQ 'PRODUCTION' ;
END
```

The request produces the following output:

```
PAGE          1

LAST_NAME      MIXED_CASE
-----
STEVENS        Stevens
SMITH          Smith
BANNING        Banning
IRVING         Irving
ROMANS         Romans
MCKNIGHT       Mcknight
```

## LJUST: Left-justifying a String

The LJUST subroutine left-justifies a character string within a field. All leading spaces become trailing spaces. The LJUST subroutine is helpful in left-justifying character strings previously right-justified or centered.

**Available on:** All platforms.

**Related functions and subroutines:**

- CTRLFLD
- RJUST

**Note:** LJUST will not have any visible effect in a report that uses StyleSheets (SET STYLE=ON) unless you center the item.

### Syntax

### How to Left-justify a String

`LJUST(inlength, infield, outfield)`

where:

`inlength`

Integer

Is the length of infield and outfield.

`infield`

Alphanumeric

Is the name of the data field to be left-justified or the input string enclosed in single quotation marks.

`outfield`

Alphanumeric

Is the name of the field to which the output is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Left-justifying a Formerly Numeric Field**

The following request converts current salaries from numeric to alphanumeric format using the FTOA subroutine. It then left-justifies the resulting alphanumeric values.

**Note:** If you are running this request on a platform where StyleSheets are turned on by default (for example WebFOCUS), issue SET STYLE=OFF before running the request.

```
SET STYLE=OFF

TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND COMPUTE
SAL_STRING/A12 = FTOA(CURR_SAL, '(D8.2M)', SAL_STRING);
LEFT_SAL/A12 = LJUST(12, SAL_STRING, LEFT_SAL);
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS'
END
```

The request produces the following output:

```
PAGE      1
```

LAST_NAME	FIRST_NAME	SAL_STRING	LEFT_SAL
BLACKWOOD	ROSEMARIE	\$21,780.00	\$21,780.00
CROSS	BARBARA	\$27,062.00	\$27,062.00
GREENSPAN	MARY	\$9,000.00	\$9,000.00
JONES	DIANE	\$18,480.00	\$18,480.00
MCCOY	JOHN	\$18,480.00	\$18,480.00
SMITH	MARY	\$13,200.00	\$13,200.00

## LOCASE: Converting Text to Lowercase

The LOCASE subroutine converts the alphabetical text in a field to lowercase.

This is useful for converting input fields from FIDEL CRTFORMs and from non-FOCUS applications to lowercase.

**Note:** This subroutine used to be named LOWCASE on the Windows platform. For upward compatibility, you can issue the command LET LOCASE = LOWCASE.

**Available on:** All platforms.

### Related functions and subroutines:

- LCWORD
- UPCASE

### Syntax

### How to Convert Text to Lowercase

`LOCASE(inlength, infield, outfield)`

where:

*inlength*

Integer

Is the length of the input and output fields. It must be greater than 0. The length, in characters, must be equal for both arguments; otherwise, an error occurs.

*infield*

Alphanumeric

Is the name of the field to convert or the input string enclosed in single quotation marks.

*outfield*

Alphanumeric

Is the name of the field in which to store the converted text. This can be the same as the infield. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Converting LAST\_NAME to Lowercase**

The following request converts LAST\_NAME values, which are all uppercase, to lowercase:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWER_NAME/A15 = LOCASE(15, LAST_NAME, LOWER_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

```
PAGE      1

LAST_NAME      LOWER_NAME
-----
SMITH          smith
JONES          jones
MCCOY          mccoy
BLACKWOOD     blackwood
GREENSPAN     greenspan
CROSS         cross
```

**LOG: Calculating the Natural Logarithm**

The LOG function returns the natural logarithm of its argument.

**Available on:** All platforms.

**Related functions and subroutines:**

EXP

**Syntax**

**How to Calculate the Natural Logarithm**

```
LOG(argument)
```

where:

*argument*  
 Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation. If you enter an argument less than or equal to 0, LOG returns 0.

**Example**

## Report Request Calculating Natural Logarithm of Current Salary

The following request calculates the log of employees' current salaries:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
LOG_CURR_SAL/D12.2 = LOG(CURR_SAL);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION' ;
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	CURR_SAL	LOG_CURR_SAL
BANNING	JOHN	\$29,700.00	10.30
IRVING	JOAN	\$26,862.00	10.20
MCKNIGHT	ROGER	\$16,100.00	9.69
ROMANS	ANTHONY	\$21,120.00	9.96
SMITH	RICHARD	\$9,500.00	9.16
STEVENS	ALFRED	\$11,000.00	9.31

## MAX and MIN: Finding the Maximum or Minimum Value

The MAX and MIN functions return either the maximum or minimum value (respectively) from a list of arguments.

**Available on:** All platforms.

**Syntax**

### How to Find the Maximum or Minimum Value

The syntax for MAX is

```
MAX(argument1, argument2, ...)
```

and the syntax for MIN is

```
MIN(argument1, argument2, ...)
```

where:

```
argument1, argument2
```

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

**Example**

**Report Request Determining Minimum of ED\_HRS and 30**

The following request finds the minimum value from the ED\_HRS field and the value 30:

```
TABLE FILE EMPLOYEE
PRINT ED_HRS AND COMPUTE
MIN_EDHRS_30/D12.2=MIN(ED_HRS, 30);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

This request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	ED_HRS	MIN_EDHRS_30
BLACKWOOD	ROSEMARIE	75.00	30.00
CROSS	BARBARA	45.00	30.00
GREENSPAN	MARY	25.00	25.00
JONES	DIANE	50.00	30.00
MCCOY	JOHN	.00	.00
SMITH	MARY	36.00	30.00

**MVSDYNAM: Passing a DYNAM Command to the Command Processor**

The MVSDYNAM subroutine transfers a specified FOCUS DYNAM command to the DYNAM command processor. A zero (0) return code indicates successful processing; non-zero codes indicate failure. This is useful in compiled MODIFY procedures after the CASE AT START statement to pass allocation statements to the processor.

**Available on:** MVS.

## Syntax

### How to Pass a DYNAM Command to the Command Processor

`MVSDYNAM(command, length, rc)`

where:

*command*

Alphanumeric

Is the DYNAM command, enclosed in single quotation marks, or a field or variable that contains the command. The subroutine converts lowercase input to uppercase.

*length*

Numeric

Is the command length from 1 to 256 characters long.

*rc*

I4

Is the name of the field that contains the return code. This argument can also be the format of the output value, enclosed in single quotation marks. For Maintain, specify the field name.

MVSDYNAM returns one of three possible types of codes:

0

The DYNAM command transferred and successfully executed.

*positive number*

FOCUS error number corresponding to a FOCUS error.

*negative number*

FOCUS error number corresponding to DYNAM failure (from the SVC).



*Example*

## Executing the DYNAM FREE Command

In this MODIFY request, the MVSDYNAM subroutine transfers the DYNAM FREE command to the processor. Query commands display the results before and after the DYNAM FREE command is specified. The successful return code of zero (0) is stored in the RES field.

```
-* THE RESULT OF ? TSO DDNAME CAR WILL BE BLANK AFTER ENTERING
-* 'FREE FILE CAR' AS YOUR COMMAND
DYNAM ALLOC FILE CAR DS USER1.CAR.FOCUS SHR REUSE
? TSO DDNAME CAR
-RUN
-PROMPT &XX. ENTER A SPACE TO CONTINUE.
MODIFY FILE CAR
COMPUTE LINE/A60=;
      RES/I4 = 0;
CRTFORM
" ENTER DYNAM COMMAND BELOW:"
" <LINE>"
COMPUTE
RES = MVSDYNAM(LINE, 60, RES);
GOTO DISPLAY

CASE DISPLAY
  CRTFORM LINE 1
  " THE RESULT OF DYNAM WAS <D.RES>"
GOTO EXIT
ENDCASE
DATA
END
? TSO DDNAME CAR
```

The first query command displays the allocation that results from the DYNAM ALLOCATE command. Type one space and press the Enter key to continue.

```
DDNAME      = CAR
DSNAME      = USER1.CAR.FOCUS
DISP        = SHR
DEVICE      = DISK
VOLSER      = USERMN
DSORG       = PS
RECFM       = F
SECONDARY   = 100
ALLOCATION   = BLOCKS
BLKSIZE     = 4096
LRECL       = 4096
TRKTOT      = 8
EXTENTSUSED = 1
BLKSPERTRK = 12
TRKSPERCYL = 15
CYLSPERDISK = 2227
BLKSWITTEN = 96
FOCUSPAGES = 8
ENTER A SPACE TO CONTINUE >
```

Then, enter the DYNAM FREE command. (The DYNAM keyword is assumed.)

```
ENTER DYNAM COMMAND BELOW:
free file car
```

The subroutine successfully transfers the DYNAM FREE command to the processor and the return code displays. Press the Enter key to continue.

```
THE RESULT OF DYNAM WAS 0
```

Then, the second query command indicates that the allocation has been freed.

```
DDNAME      = CAR
DSNAME      =
DISP        =
DEVICE      =
VOLSER      =
DSORG       =
RECFM       =
SECONDARY   = ****
ALLOCATION   =
BLKSIZE     = 0
LRECL       = 0
TRKTOT      = 0
EXTENTSUSED = 0
BLKSPERTRK = 0
TRKSPERCYL = 0
CYLSPERDISK = 0
BLKSWITTEN = 0
>
```

## OVLAY: Overlaying a Substring Within a String

The OVLAY subroutine overlays a substring on another character string. When specified in MODIFY procedures, the subroutine enables you to edit a part of an alphanumeric field without replacing the field entirely.

**Available on:** All platforms.

**Related functions and subroutines:**

- EDIT
- POSIT
- SUBSTR

### Syntax

### How to Overlay a Substring

`OVLAY(base, baselen, substring, sublen, position, outfield)`

where:

*base*

Alphanumeric

Is the character string to be overlaid.

*baselen*

Integer

Is the length of the base and outfield strings. If this argument is less than or equal to 0, unpredictable results occur.

*substring*

Alphanumeric

Is the substring to overlay the base string.

*sublen*

Integer

Is the length of the substring. If this argument is less than or equal to 0, the subroutine returns spaces.

*position*

Integer

Is the position in the base string where the overlay is to begin. If this argument is less than or equal to 0, the subroutine returns spaces. If the argument is larger than baselen, the subroutine returns the base string.

*outfield*

Alphanumeric

Is the name of the field to which the overlaid string is returned. If the overlaid string is longer than the output field, the string is truncated to fit the field. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Replacing Last Three Characters of EMP\_ID**

The following request replaces the last three characters of EMP\_ID (starting at the 7th position) with the three-character job code found in CURR\_JOBCODE, creating a new security identification code:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND CURR_JOBCODE AND COMPUTE
NEW_ID/A9 = OVLAY(EMP_ID, 9, CURR_JOBCODE, 3, 7, NEW_ID);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	EMP_ID	CURR_JOBCODE	NEW_ID
BLACKWOOD	ROSEMARIE	326179357	B04	326179B04
CROSS	BARBARA	818692173	A17	818692A17
GREENSPAN	MARY	543729165	A07	543729A07
JONES	DIANE	117593129	B03	117593B03
MCCOY	JOHN	219984371	B02	219984B02
SMITH	MARY	112847612	B14	112847B14

**Example****MODIFY Request Using OVRLAY**

This MODIFY procedure prompts for input using a CRTFORM screen and updates first names in the EMPLOYEE data source. The CRTFORM LOWER option enables you to update the names in lowercase, but the procedure ensures that the first letter of each name is capitalized. The procedure is:

```

MODIFY FILE EMPLOYEE
CRTFORM LOWER
  "ENTER EMPLOYEE'S ID: <EMP_ID"
  "ENTER FIRST_NAME IN LOWER CASE: <FIRST_NAME"
MATCH EMP_ID
ON NOMATCH REJECT
ON MATCH COMPUTE
  F_UP/A1 = UPCASE (1, FIRST_NAME, 'A1');
  FIRST_NAME/A10 = OVRLAY (FIRST_NAME, 10, F_UP,
    1, 1, 'A10');
  ON MATCH TYPE "CHANGING FIRST NAME TO <FIRST_NAME "
  ON MATCH UPDATE FIRST_NAME
DATA
END

```

The COMPUTE statement invokes two subroutines:

- The UPCASE subroutine extracts the first letter and converts it to uppercase.
- The OVRLAY subroutine replaces the present first letter in the name with the uppercase initial.

A sample execution is:

```

ENTER EMPLOYEE'S ID: 071382660
ENTER FIRST_NAME IN LOWER CASE: alfred

```

The procedure processes as:

1. The procedure prompts you from a CRTFORM screen for an employee ID and a first name. You type the following data and press the Enter key:

```

EMPLOYEE'S ID: 071382660
FIRST NAME:    alfred

```

2. The procedure searches the data source for the ID 071382660. If it finds the ID, it continues processing the transaction. In this case, the ID exists and belongs to Alfred Stevens.
3. The UPCASE subroutine extracts the letter a from alfred and converts it to the letter A.

4. The OVRLAY subroutine overlays the letter A on alfred. The first name is now Alfred.

```
ENTER EMPLOYEE'S ID:  
ENTER FIRST_NAME IN LOWER CASE:  
  
CHANGING FIRST NAME TO Alfred
```

5. The procedure updates the first name in the data source.
6. When you exit the procedure with PF3, the FOCUS transaction message indicates that one update occurred.

```
TRANSACTIONS:          TOTAL =      1  ACCEPTED=      1  REJECTED=      0  
SEGMENTS:              INPUT =      0  UPDATED =      1  DELETED =      0
```

## PARAG: Dividing Text Into Smaller Lines

The PARAG subroutine divides lines of text into smaller lines by marking them off with a delimiter character. The GETTOK subroutine can then place the smaller lines, called sublines, into different fields.

The PARAG subroutine works by scanning a specific number of characters from the beginning of the line and replacing the last space in this group with a delimiter. It then scans the next group of characters starting from the delimiter and replaces the last space in this group with a second delimiter. It repeats this process until the end of the line. Each group of characters marked off by the delimiter becomes a subline.

If the subroutine finds no spaces in the group it scans, it replaces the next character after the group with the delimiter. Therefore, be sure that no word of text is longer than the number of characters scanned by the subroutine (the maximum subline length).

**Available on:** All platforms.

**Related functions and subroutines:**

GETTOK

## Syntax

### How to Divide Text Into Smaller Lines

`PARAG(inlen, infield, 'delim', subsize, outfield)`

where:

*inlen*

Integer

Is the length of input string and the outfield.

*infield*

Alphanumeric

Is the input string.

*delim*

Alphanumeric

Is the delimiter character. Choose a character that does not appear in the text.

*subsize*

Integer

Is the maximum length of the subline.

*outfield*

Alphanumeric

Is the name of the field to which the delimited text is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Note:** If the input lines of text are roughly equal in length, you can keep the sublines equal by specifying a subline length that evenly divides into the length of the text lines. For example, if you are dividing text lines 120 characters long, you can divide each of them into two sublines 60 characters long, three sublines 40 characters long, and so on. This enables you to print lines of text in paragraph form.

However, if you divide the lines evenly, you may create more sublines than you intend. For example, suppose you divide 120-character text lines into two lines of 60 characters maximum length. One line is divided so that the first subline is 50 characters long and the second is 55. This leaves room for a third subline 15 characters long.

To correct this, insert a space (using weak concatenation) at the beginning of the extra subline, then append this subline (using strong concatenation) to the end of the one before it.

**Example**

**Report Request Dividing Address Line Into Smaller Lines**

The following request divides an address line into smaller lines using commas as delimiters:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN2 AND COMPUTE
PARA_ADDR/A20 = PARAG(20, ADDRESS_LN2, ',', 10, PARA_ADDR);
BY LAST_NAME
WHERE TYPE EQ 'HSM';
END
```

The request produces the following output:

```
PAGE      1

LAST_NAME      ADDRESS_LN2      PARA_ADDR
-----
BANNING        APT 4C          APT 4C ,
CROSS          147-15 NORTHERN BLD 147-15,NORTHERN,BLD
GREENSPAN     13 LINDEN AVE.   13 LINDEN,AVE.
IRVING         123 E 32 ST.    123 E 32,ST. ,
JONES         235 MURRAY HIL PKWY 235 MURRAY,HIL PKWY
MCKNIGHT      117 HARRISON AVE. 117 ,HARRISON,AVE.
ROMANS        271 PRESIDENT ST. 271 ,PRESIDENT,ST.
SMITH         136 E 161 ST.   136 E 161,ST.
```

**PCKOUT: Writing Packed Numbers of Different Lengths**

The PCKOUT subroutine enables requests to write packed numbers (where the operating system supports it) of different lengths to extract files (HOLD and SAVE files). When a request saves packed fields in extract files, it writes them as 8- or 16-byte fields regardless of their format specifications. With the PCKOUT subroutine, you can vary their lengths between 1 to 16 bytes.

**Available on:** All platforms.

**Related functions and subroutines:**

- CHKPCK
- ITOPACK



**Syntax****How to Write Packed Numbers of Different Lengths**

```
PCKOUT(infield, outlength, outfield)
```

where:

*infield*

Numeric

Is the input field that contains the values. The field can be packed, integer, floating-point or double-precision. If the field is not integer, its values are rounded to the nearest integer.

*outlength*

Numeric

Is the output field length from 1 to 16 bytes.

*outfield*

Alphanumeric

Is the name of the output field written to the extract file. The subroutine returns the field as alphanumeric although it contains packed data. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example****Writing Packed Numbers of Different Lengths**

This request writes names, salaries, and dates of hire to a SAVE file. The salaries from the CURR\_SAL field (USAGE=D12.2M) are converted and written to the 5-byte packed field SHORT\_SAL:

```
DEFINE FILE EMPLOYEE
SHORT_SAL/A5 = PCKOUT(CURR_SAL, 5, SHORT_SAL);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME SHORT_SAL HIRE_DATE
ON TABLE SAVE
END
```

After FOCUS creates the SAVE file, the FOCUS message returns the fields and their lengths:

```
>
NUMBER OF RECORDS IN TABLE=      12  LINES=      12

EBCDIC RECORD NAMED  SAVE
FIELDNAME                ALIAS                FORMAT                LENGTH

LAST_NAME                 LN                A15                   15
SHORT_SAL                 A5                   5
HIRE_DATE                 HDT                I6YMD                 6

TOTAL                                26
DCB USED WITH FILE SAVE      IS DCB=(RECFM=FB,LRECL=00026,BLKSIZE=00520)
>
```

## POSIT: Finding Substring Position

The POSIT subroutine finds the starting positions of substrings within larger strings. For example, the position of the substring DUCT in the character string PRODUCTION is position 4.

If the substring is not in the parent string, the subroutine returns the value 0.

**Available on:** All platforms.

### **Related functions and subroutines:**

OVRLAY

## Syntax

### How to Find a Substring Position

*POSIT*(*parent*, *inlength*, *substring*, *sublength*, *outfield*)

where:

*parent*

Alphanumeric

Is the field containing the parent character string.

*inlength*

Integer

Is the parent field length. If this argument is less than or equal to 0, the subroutine returns 0.

*substring*

Alphanumeric

Is the substring whose position you wish to find, enclosed in single quotation marks, or a field that contains the string.

*sublength*

Integer

Is the length of *substring*. If this argument is less than or equal to 0, or if it is greater than the *inlength* argument, the subroutine returns a 0.

*outfield*

Integer

Is the name of the field to which the position is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

### Example

## Report Request Determining First Position of the Letter I in LAST\_NAME

The following request displays the positions of the first capital letter I in last names:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2');
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following output:

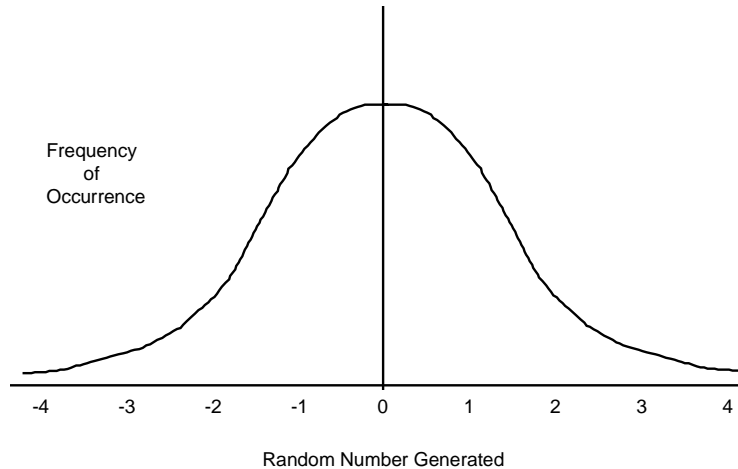
PAGE 1

LAST_NAME	I_IN_NAME
-----	-----
STEVENS	0
SMITH	3
BANNING	5
IRVING	1
ROMANS	0
MCKNIGHT	5

## PRDNOR, PRDUNI, RDNORM, and RDUNIF: Generating Random Numbers

The PRDNOR, PRDUNI, RDNORM, and RDUNIF subroutines generate random numbers:

- RDNORM generates double-precision random numbers that are normally distributed with an arithmetic mean of 0 and a standard deviation of 1. If you use the RDNORM subroutine to generate a large set of numbers (between 1 and 32768), it has the following properties:
  - The numbers in the set lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, which means that there are more numbers close to 0 than farther away.



- The average of the set is close to 0.
- The set can contain numbers of any size, but most of the numbers are between 3 and -3.
- PRDNOR does the same thing as RDNORM, except that the set of random numbers is *reproducible*.
- RDUNIF generates double-precision random numbers uniformly distributed between 0 and 1 (that is, any random number it generates has an equal probability of being anywhere between 0 and 1).
- PRDUNI does the same thing as RDUNIF, except that the set of random numbers is *reproducible*.

**Available on:** All platforms.

## Syntax

### How to Use RDNORM and RDUNIF to Generate Random Numbers

*subroutine(outfield)*

where:

*subroutine*

Is one of the following:

**RDNORM** generates normally distributed random numbers with an arithmetic mean of 0 and a standard deviation of 1.

**RDUNIF** generates random numbers uniformly distributed between 0 and 1.

*outfield*

Double-precision

Is the name of the double-precision field that contains the random numbers. This argument can also be the format of the output value, enclosed in single quotation marks.

## Syntax

### How to Use PRDNOR and PRDUNI to Generate Random Numbers

*subroutine(seed, outfield)*

where:

*subroutine*

Is one of the following:

**PRDNOR** generates reproducible normally distributed random numbers with an arithmetic mean of 0 and a standard deviation of 1.

**PRDUNI** generates reproducible random numbers uniformly distributed between 0 and 1.

*seed*

Numeric

Is the seed or the field that contains the seed, up to nine bytes. The seed is truncated to an integer.

*outfield*

Double-precision

Is the name of the field that contains the random numbers. This argument can also be the format of the output value, enclosed in single quotation marks.

**Note:** For the PRDUNI subroutine, CMS behavior differs from MVS behavior. In CMS, the seed number changes upon multiple executions as the subroutine is reloaded. In MVS, the subroutine is loaded once. To keep the subroutine loaded for the duration of the session, we recommend assigning the subroutine to a temporary field using a DEFINE command. The subroutine remains loaded in memory until the DEFINE is cleared.

*Example*

**Report Request Using RDNORM to Generate Random Numbers**

Suppose you want to randomly pick five employees in your company to participate in a survey. The following request generates a random number for each employee and then chooses the top five:

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = RDNORM(RAND);
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

(You could also use RDUNIF to generate RAND.)

The request produces output similar to the following (your output will appear differently, since each time RDNORM generates different random numbers):

```
PAGE      1

RAND  LAST_NAME      FIRST_NAME
----  -
.65   CROSS           BARBARA
.20   BANNING         JOHN
.19   IRVING          JOAN
.00   BLACKWOOD       ROSEMARIE
-.14  GREENSPAN       MARY
```

*Example*

**Report Request Using PRDNOR to Generate Random Numbers**

This is the same request used for the RDNORM subroutine, except, that every time you execute it, the PRDNOR subroutine produces the same results. To change the results, change the seed, specified here as 40. The request is:

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = PRDNOR(40, RAND);
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

The request produces the following output:

```
PAGE      1

RAND  LAST_NAME      FIRST_NAME
----  -
1.38  STEVENS         ALFRED
1.12  MCCOY             JOHN
.55   SMITH            RICHARD
.21   JONES            DIANE
.01   IRVING          JOAN
```

**RJUST: Right-justifying a String**

The RJUST subroutine right-justifies a character string within a field. All trailing spaces become leading spaces. This subroutine is helpful when you display alphanumeric fields containing numbers.

**Available on:** All platforms.

**Related functions and subroutines:**

- CTRFLD
- LJUST

**Note:** RJUST will not have any visible effect in a report that uses StyleSheets (SET STYLE=ON) unless you center the item.

## Syntax

### How to Right-justify a String

```
RJUST(inlength, infield, outfield)
```

where:

*inlength*

Integer

Is the length of *infield* and *outfield*.

*infield*

Alphanumeric

Is the input field or string enclosed in single quotation marks.

*outfield*

Alphanumeric

Is the name of the field to which the output is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

To avoid justification problems, *inlength* and *infield* must be the same length.

## Example

### Report Request Right-justifying a Field

The following request shows last names left-justified and right-justified.

**Note:** If you are running this request on a platform where StyleSheets are turned on by default (for example WebFOCUS), issue SET STYLE=OFF before running the request.

```
SET STYLE=OFF  
  
TABLE FILE EMPLOYEE  
PRINT LAST_NAME AND COMPUTE  
RIGHT_NAME/A15 = RJUST(15, LAST_NAME, RIGHT_NAME);  
WHERE DEPARTMENT EQ 'MIS'  
END
```

The request produces the following output:

```
PAGE      1  
  
LAST_NAME      RIGHT_NAME  
-----  
SMITH          SMITH  
JONES          JONES  
MCCOY          MCCOY  
BLACKWOOD     BLACKWOOD  
GREENSPAN     GREENSPAN  
CROSS         CROSS
```



## SQRT: Calculating the Square Root

The SQRT function calculates the square root of its argument.

**Available on:** All platforms.

### Syntax

### How to Calculate the Square Root

`SQRT(argument)`

where:

*argument*

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

### Example

### Report Request Calculating Square Root of Movies' List Price

The following request calculates the square root of a movie's list price:

```
TABLE FILE MOVIES
PRINT LISTPR AND COMPUTE
SQRT_LISTPR/D12.2 = SQRT(LISTPR);
BY TITLE
WHERE CATEGORY EQ 'MUSICALS';
END
```

This request produces the following output:

```
PAGE          1

TITLE                                LISTPR      SQRT_LISTPR
-----                                -
ALL THAT JAZZ                        19.98        4.47
CABARET                               19.98        4.47
CHORUS LINE, A                       14.98        3.87
FIDDLER ON THE ROOF                 29.95        5.47
```

## SUBSTR: Extracting a Substring

The SUBSTR subroutine extracts a substring from a large character string called a parent string.

Another way to extract substrings is to use the EDIT function. The differences are:

- The EDIT function can extract a substring from different parts of the parent string. For example, it can extract the first two characters and the last two characters of a string to form a single substring. Also, it can insert characters into a substring.
- The SUBSTR subroutine can vary the position of the substring depending on the values of other fields.

**Available on:** All platforms.

**Related functions and subroutines:**

EDIT

### Syntax

### How to Extract a Substring

```
SUBSTR(inlength, parent, start, end, sublength, outfield)
```

where:

*inlength*

Integer

Is the length of the parent string.

*parent*

Alphanumeric

Is the field containing the parent string or the parent string enclosed in single quotation marks.

*start*

Integer

Is the starting position of the substring in the parent string. If this argument is less than 1, the subroutine returns spaces.

*end*

Integer

Is the ending position of the substring. If this argument is less than the *start* argument or greater than the *inlength* argument, the subroutine returns spaces.

*sublength*

Integer

Is the length of the substring (normally *end* - *start* + 1). If the *sublength* is longer than *end*

- *start* + 1, the substring is padded with trailing spaces. If it is shorter, the substring is truncated. This value should be the declared length of *outfield*. Only *sublength* characters will be processed.

*outfield*

Alphanumeric

Is the name of the field to which the substring is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Example**

**Report Request Extracting Three Characters From LAST\_NAME Beginning With the Letter I**

The following request uses the POSIT subroutine to determine the position of the first letter I in LAST\_NAME. Then the report extracts the letter I and the next two characters.

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2'); AND
COMPUTE I_SUBSTR/A3 = SUBSTR(15, LAST_NAME, I_IN_NAME, I_IN_NAME+2, 3,
I_SUBSTR);
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The request produces the following report:

```
PAGE      1
```

LAST_NAME	I_IN_NAME	I_SUBSTR
-----	-----	-----
BANNING	5	ING
IRVING	1	IRV
MCKNIGHT	5	IGH
ROMANS	0	
SMITH	3	ITH
STEVENS	0	

Notice that since Stevens and Romans have no I in their names, SUBSTR extracts a blank string.

## TODAY: Returning the Current Date

The TODAY subroutine retrieves the current date from the system in the format MM/DD/YY or MM/DD/YYYY, depending on the format of the output field.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine (which always returns the current date in the format MM/DD/YY), change the DATEFNS setting to OFF.

**Available on:** All platforms.

**Related functions and subroutines:**

HHMMSS

### Syntax

### How to Retrieve the Current Date

`TODAY(outfield)`

where:

*outfield*

Alphanumeric, at least A8

Is the name of the field to which the current date in MM/DD/YY[YY] format is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

If DATEFNS=ON and *outfield* is A8 or A9, TODAY returns the 2-digit year. If DATEFNS=ON and *outfield* is A10 or greater, TODAY returns the 4-digit year. If DATEFNS=OFF, TODAY returns the 2-digit year, regardless of the format of *outfield*.

#### Note:

- You can retrieve the date in the same format (separated by slashes) by using the system variable &DATE. You can retrieve the date without the slashes using the system variables &YMD, &MDY, and &DMY. The system variable &DATEfmt retrieves the date in a specified format.
- You can remove the embedded slashes using the EDIT function.
- Compiled MODIFY procedures cannot use Dialogue Manager system variables. They must use the TODAY subroutine to obtain the date.

The TODAY subroutine always returns a date that is current. Therefore, if you are running an application late at night, you may want to use the TODAY subroutine.

*Example*

## Report Request Displaying the Current Date

The following request retrieves the current date and displays it in a report heading:

```
DEFINE FILE EMPLOYEE
DATE/A10 WITH EMP_ID=TODAY( DATE ) ;
END

TABLE FILE EMPLOYEE
SUM CURR_SAL BY DEPARTMENT
HEADING
"PAGE <TABPAGENO  "
"SALARY REPORT RUN ON <DATE  "
END
```

The request produces the following output:

```
PAGE 1
SALARY REPORT RUN ON 12/13/1999
DEPARTMENT      CURR_SAL
-----
MIS              $108,002.00
PRODUCTION      $114,282.00
```

## UFMT: Converting Alphanumeric to Hexadecimal

The UFMT subroutine converts characters in alphanumeric field values to hexadecimal (HEX) representation.

This subroutine is especially useful for examining data of unknown format. As long as the length of the data is known, its content can be examined.

**Available on:** MVS, OpenVMS, VM/CMS, WebFOCUS.

### *Syntax*

### How to Convert Alphanumeric to Hexadecimal

```
UFMT(infield, inlength, outfield)
```

where:

*infield*

Alphanumeric

Is the input field or an alphanumeric string enclosed in single quotation marks.

*inlength*

Numeric

Is the input field length.

*outfield*

Alphanumeric

Is the name of the field that contains the HEX equivalent. The format of the *outfield* argument must be alphanumeric and have a length that is twice as long as the *inlength* argument ( $2 * inlength$ ). This argument can also be the format of the output value, enclosed in single quotation marks.

*Example*

**Report Request Converting JOBCODE to Hexadecimal**

The following request uses the UFMT subroutine to convert the values in the JOBCODE field to their HEX representation and store them in the HEXCODE temporary field. Notice that the format of the temporary field is twice as large as the *inlength* argument:

```

DEFINE FILE JOBFIL
HEXCODE/A6 = UFMT(JOBCODE, 3, HEXCODE);
END
TABLE FILE JOBFIL
PRINT JOBCODE HEXCODE
END
    
```

The resulting output is:

```

PAGE          1

JOBCODE      HEXCODE
-----
A01          C1F0F1
A02          C1F0F2
A07          C1F0F7
A12          C1F1F2
A14          C1F1F4
A15          C1F1F5
A16          C1F1F6
A17          C1F1F7
B01          C2F0F1
B02          C2F0F2
B03          C2F0F3
B04          C2F0F4
B14          C2F1F4
    
```

## UPCASE: Converting Text to Uppercase

The UPCASE subroutine converts a string of characters to uppercase.

One reason you might use UPCASE is when you are sorting on a field that contains both mixed case and uppercase values. In these cases, sorting uses the ASCII or EBCDIC sorting order, which may cause unpredictable results. To obtain consistent results, define a new field with all of the values in uppercase, and sort on that.

In FIDEL, CRTFORM LOWER retains the case of entries as they were typed. You can use the UPCASE subroutine to convert entries for particular fields to uppercase.

**Available on:** All platforms.

### **Related functions and subroutines:**

- LCWORD
- LOCASE
- MXCASE

## *Syntax*

### How to Convert Text to Uppercase

`UPCASE(length, input, output)`

where:

*length*

Integer

Is the length of both the *input* and the *output* strings.

*input*

Alphanumeric

Is the mixed-case input string or field.

*output*

Alphanumeric

Is the uppercase output string or field. This argument can also be the format of the output value, enclosed in single quotation marks.



*Example*

**Report Request Converting Mixed Case Names to Uppercase**

Suppose you are sorting on a field that contains both uppercase and mixed case values. The following request defines a field called LAST\_NAME\_MIXED that contains both uppercase and mixed case values:

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
  LCWORD (15 , LAST_NAME, 'A15');
END
```

Suppose you execute a request that sorts by this field:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME BY LAST_NAME_MIXED
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
END
```

On an EBCDIC-based platform, this request produces the following output:

```
PAGE      1

LAST_NAME_MIXED  FIRST_NAME
-----
Banning          JOHN
BLACKWOOD        ROSEMARIE
CROSS            BARBARA
Mcknight       ROGER
MCCOY         JOHN
Romans           ANTHONY
```

On an ASCII-based platform, this request produces the following output:

```
PAGE      1

LAST_NAME_MIXED  FIRST_NAME
-----
BLACKWOOD     ROSEMARIE
Banning       JOHN
CROSS            BARBARA
MCCOY           JOHN
Mcknight        ROGER
Romans           ANTHONY
```

In the first example, Mcknight appears before MCCOY, since the EBCDIC sorting order places lowercase letters before uppercase letters. In the second example, Blackwood appears before Banning, since the ASCII sorting order places uppercase letters before lowercase letters. In either case, this is not how you would expect your report to be sorted.

The solution is to create a new field with all uppercase letters and sort using this field:

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
  LCWORD (15, LAST_NAME, 'A15');
LAST_NAME_UPPER/A15=UPCASE (15, LAST_NAME_MIXED, 'A15') ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME_MIXED AND FIRST_NAME BY LAST_NAME_UPPER
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
END
```

Now, when you execute the request, the names are sorted correctly:

```
PAGE      1
```

LAST_NAME_UPPER	LAST_NAME_MIXED	FIRST_NAME
-----	-----	-----
BANNING	Banning	JOHN
BLACKWOOD	BLACKWOOD	ROSEMARIE
CROSS	CROSS	BARBARA
MCCOY	MCCOY	JOHN
MCKNIGHT	Mcknight	ROGER
ROMANS	Romans	ANTHONY

If you don't want to see the field with all uppercase values, you can NOPRINT it.

## Example

### MODIFY Request Using UPCASE

Suppose your company decided to store employee names in mixed case and the department assignments in uppercase in the EMPLOYEE data source.

To enter records of new employees, execute this MODIFY procedure:

```
MODIFY FILE EMPLOYEE
CRTFORM LOWER
"ENTER EMPLOYEE'S ID : <EMP_ID"
"ENTER LAST_NAME: <LAST_NAME FIRST_NAME: <FIRST_NAME"
"TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET"
" "
"ENTER DEPARTMENT ASSIGNMENT: <DEPARTMENT"
MATCH EMP_ID
  ON MATCH REJECT
  ON NOMATCH COMPUTE
    DEPARTMENT = UPCASE (10, DEPARTMENT, 'A10');
  ON NOMATCH INCLUDE
  ON NOMATCH TYPE "DEPARTMENT VALUE CHANGED TO UPPERCASE: <DEPARTMENT"
DATA
END
```

A sample execution is as follows:

```
ENTER EMPLOYEE'S ID : 444555666
ENTER LAST_NAME: Cutter          FIRST_NAME: Alan
TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET
```

```
ENTER DEPARTMENT ASSIGNMENT: sales
```

The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department on a CRTFORM screen. The CRTFORM LOWER option retains the case of entries as they were typed.
2. You type the following data and press the ENTER key:

```
EMPLOYEE'S ID:          444555666
LAST_NAME:              Cutter
FIRST_NAME:             Alan
DEPARTMENT ASSIGNMENT: sales
```

3. The procedure searches the data source for the ID 444555666. If it does not find the ID, it continues processing the transaction.
4. The UPCASE subroutine converts the DEPARTMENT entry “sales” to “SALES.”

```
ENTER EMPLOYEE'S ID :
ENTER LAST_NAME:          FIRST_NAME:
TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET
```

```
ENTER DEPARTMENT ASSIGNMENT:
```

```
DEPARTMENT VALUE CHANGED TO UPPERCASE: SALES
```

5. The procedure adds the transaction to the data source.
6. When you exit the procedure with PF3, the FOCUS transaction message indicates the number of transactions accepted or rejected.

```
TRANSACTIONS:          TOTAL =      1  ACCEPTED=      1  REJECTED=      0
SEGMENTS:              INPUT =      1  UPDATED =      0  DELETED =      0
```

## YM: Calculating Elapsed Months

The YM subroutine calculates the number of months that elapse between two dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT subroutine or the EDIT function.

This subroutine has been rewritten to support Year 2000 dates. To use the old version of this subroutine, change the DATEFNS setting to OFF.

**Available on:** All platforms.

**Related functions and subroutines:**

- CHGDAT
- DATEDIF
- DMY, MDY, YMD

### Syntax

### How to Calculate Elapsed Months

*YM(fromdate, todate, outfield)*

where:

*fromdate*

Numeric

Is the starting date in year-month format (for example, I4YM). If the date is not valid, the subroutine returns a 0.

*todate*

Numeric

Is the ending date in year-month format. If the date is not valid, the subroutine returns a 0.

*outfield*

Integer

Is the name of the field to which the number of months between the two dates is returned. This argument can also be the format of the output value, enclosed in single quotation marks.

**Tip:**

If the input date is in integer year-month-day format (I6YMD or I8YYMD), simply divide the date by 100 to convert to year-month format and set the result to be an integer. This causes the day portion of the date, which is now after the decimal point, to be dropped.

**Example**

**Report Request Calculating Difference in Months Between Two Dates**

The following request shows the number of months that elapse between the time employees get raises and the time they were hired. Note that the COMPUTE expression converts the dates from year-month-day to year-month format by dividing the dates by 100.

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100; NOPRINT AND COMPUTE
MONTH_INC/I4YM = DAT_INC/100; NOPRINT AND COMPUTE
MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
BY LAST_NAME BY FIRST_NAME BY HIRE_DATE
IF MONTHS_HIRED NE 0
WHERE DEPARTMENT EQ 'MIS';
END
```

The request produces the following output:

PAGE 1

LAST_NAME	FIRST_NAME	HIRE_DATE	RAISE DATE	MONTHS_HIRED
CROSS	BARBARA	81/11/02	82/04/09	5
GREENSPAN	MARY	82/04/01	82/06/11	2
JONES	DIANE	82/05/01	82/06/01	1
MCCOY	JOHN	81/07/01	82/01/01	6
SMITH	MARY	81/07/01	82/01/01	6

---

## CHAPTER 4

# Managing Applications With Dialogue Manager

### Topics:

- Overview of Dialogue Manager Capabilities
- Creating and Storing Procedures
- Executing Procedures
- Including Comments in a Procedure
- Overview of Dialogue Manager Commands
- Sending a Message to the User: -TYPE
- Controlling Execution: -RUN, -EXIT, and -QUIT
- Branching
- Looping
- Using Expressions: -SET
- Additional Facilities
- Using Variables in Procedures
- Supplying Values for Variables at Run Time
- Dialogue Manager Quick Reference

This topic describes how to make report procedures more dynamic by using Dialogue Manager control commands and variables.

# Overview of Dialogue Manager Capabilities

Dialogue Manager enables you to execute stored procedures. In the FOCUS community, stored procedures are referred to as FOCEXECs. In this topic, they are referred to simply as *procedures*.

Dialogue Manager helps you build and manage the execution of procedures, giving you flexibility in application design. You can use Dialogue Manager control commands to determine the sequence in which FOCUS commands (such as TABLE) execute. Dialogue Manager also enables you to use variables in your procedures and supply values for those variables at run time. You can create a dialogue between the user and the terminal through various prompting methods, including full-screen forms, menus and windows that you design yourself, and system queries, as well as supplying values directly in the procedure.

Using Dialogue Manager control commands and variables, your application can respond to user input and environment conditions at run time. It is important to understand how Dialogue Manager processes an application's commands and variables.

## 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 that 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 themselves call for no special processing, so in this case 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

```
  .  
  .  
  .
```

Since there is no 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 this:

```
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.

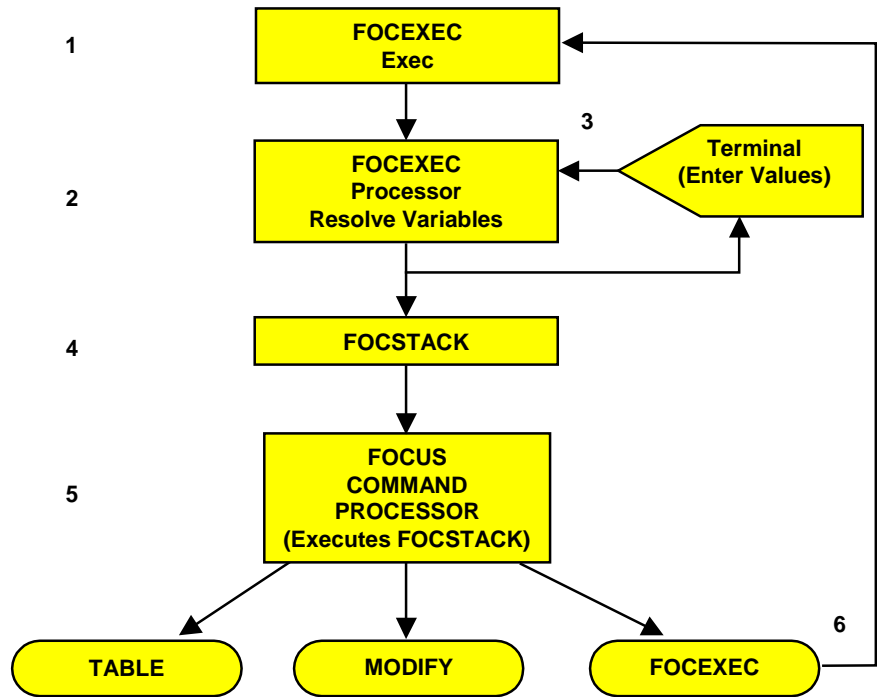


Figure 4-1. Schematic Diagram of Procedure Processing

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 their current values.
3. If a variable is missing a value, FOCUS issues a prompt. The user then supplies the missing value.

4. When a command line that contains 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.
6. In the previous example the FOCUS Command Processor routes execution to the TABLE module and executes the TABLE request that was stacked.

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.

**Note:** Any FOCUS command can be placed in a procedure. This includes the EXEC command itself. When an EXEC command is processed in a procedure, the commands from the new procedure are first stacked and then executed. Multiple levels of nesting are permitted when you use the EXEC command, while only four levels of nesting are permitted when you use `-INCLUDE`.

## Overview of Dialogue Manager Variables

You can write procedures that contain variables whose values are unknown until run time; this technique allows a user to customize the procedure by supplying different values each time it executes. Variables fall into two categories:

- Local and global variables, whose values must be supplied at run time. Local variables retain their values only for one procedure. Global variables retain their values across procedures unless you explicitly clear them. They lose their 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, whose 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 about variables, see *Using Variables in Procedures* on page 4-49.

## Creating and Storing Procedures

You can create procedures with your system editor or with the FOCUS integrated text editor, TED. TED has two features that make it particularly useful for creating and editing procedures:

- If you type TED without specifying a procedure name, the last executed procedure is automatically selected. This is convenient when developing and testing new procedures.
- You can test the execution of the procedure by typing RUN on the TED command line. This automatically saves the procedure and executes it. If there is an error in your procedure, type TED. This brings you back into the editor and places you directly on the line in which the error was detected.

These options complement the FILE and SAVE options that are common to other editors.

Follow these general rules when you are 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 (-) in the first position. The continuation line can begin immediately after the hyphen, or you may insert a space between the hyphen and the rest of the line.
- Procedures must have the record format RECFM=F and the logical record length (LRECL) 80.

# Executing Procedures

Procedures are generally initiated from the FOCUS prompt (>). Type the command EXEC, or its abbreviation EX, followed by the name of the procedure.

## *Example*

### Executing a Procedure

Either of the following commands

```
EXEC SLRPT
```

or

```
EX SLRPT
```

will summon the procedure named SLRPT for execution.

## Controlling Access to Data

You can set a password in a procedure and tie it to different portions of a procedure.

## *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.

## Including Comments in a Procedure

It is good practice to include comments in procedures for the benefit of others who may read or refine them at a later date. Comments are particularly recommended as procedure headers to give version, date, and other relevant information. It is easier to track and maintain large software applications when they are carefully commented. Comments are ignored during actual execution.

To add comment lines to a Dialogue Manager procedure, precede them with a hyphen and an asterisk (-\*). Any text whatsoever may immediately follow the -\*. You can place comment lines anywhere in a procedure.

Comments do not appear on the terminal nor do they trigger any 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 option ECHO = ALL.

### *Example*

### Including Comments in a Procedure

The following example contains two comment lines:

```
-* Version 1 6/30/85 SLRPT FOCEXEC
-* Component of Retail Sales Reporting Module
TABLE FILE SALES
HEADING CENTER
"MONTHLY SALES FOR STAMFORD"
.
.
.
```

# Overview of Dialogue Manager Commands

Dialogue Manager provides commands for accomplishing the following tasks:

- Sending messages to the user.
- Displaying values.
- Controlling the values of variables, including reading variables from and writing values to an external file.
- Testing conditions and branching.
- Controlling the execution of stacked commands.
- Calling another procedure.
- Issuing operating system commands specific to your environment.

## Reference

### Summary of Dialogue Manager Commands

The following pages describe all Dialogue Manager commands. They are listed in alphabetical order. The categories used to describe them in the quick reference at the end of this topic are briefly outlined below:

<b>Command</b>	Lists the name of the command.
<b>Syntax</b>	Shows exactly how the command components must appear in a procedure.
<b>Function</b>	Outlines the meaning and purpose of the command.
<b>Similar Command</b>	Describes the relationship between the Dialogue Manager command and other FOCUS commands (for example, -TYPE and TYPE).

<b>Command</b>	<b>Meaning</b>
-*	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.
-CMS	Executes a CMS command from within Dialogue Manager.
-CMS RUN	In CMS, loads and executes a user-written subroutine.
-CRTCLEAR	Clears the screen display.
-CRTFORM	Initiates full-screen variable data entry.

Command	Meaning
<code>-DEFAULT</code> <code>-DEFAULTS</code>	Presets initial values for variable substitution.
<code>-EXIT</code>	Executes stacked commands and returns to the FOCUS prompt.
<code>-GOTO</code>	Establishes an unconditional branch.
<code>-HTMLFORM</code>	For use with the Web Interface to FOCUS.
<code>-IF</code>	Tests and branches control based on test results.
<code>-INCLUDE</code>	Dynamically incorporates one procedure in another.
<code>-label</code>	Is a user-supplied name that identifies the target for <code>-GOTO</code> or <code>-IF</code> .
<code>-MVS RUN</code>	Same as <code>-TSO RUN</code> .
<code>-PASS</code>	Sets password directly.
<code>-PROMPT</code>	Types a prompt message on the screen and reads a reply.
<code>-QUIT</code>	Exits the procedure without executing it.
<code>-READ</code>	Reads records from a non-FOCUS file.
<code>-REPEAT</code>	Executes a loop.
<code>-RUN</code>	Executes all stacked FOCUS commands and returns to procedure for further processing.
<code>-SET</code>	Assigns a value to a variable.
<code>-TSO RUN</code>	In MVS/TSO, loads and executes a user-written subroutine.
<code>-TYPE</code>	Types informative message to screen or other output device.
<code>-WINDOW</code>	Invokes Window Painter, transferring control from the procedure to the specified window file.
<code>-WRITE</code>	Writes a record to a non-FOCUS file.
<code>"..."</code>	Brackets contents for <code>-CRTFORM</code> display line.
<code>-? SET SETCOMMAND</code> <code>&amp;myvar</code>	Captures the setting of SETCOMMAND in &myvar.
<code>-? &amp;[string]</code>	Displays the values of currently defined amper variables.

## Sending a Message to the User: -TYPE

The Dialogue Manager command -TYPE enables you to send informative messages to the screen while a procedure is processing. These messages serve a variety of functions. They can explain the purpose of the procedure, the results of computations or calculations, or preface prompts requesting information from the terminal. -TYPE triggers these messages.

### *Syntax*

### How to Send a Message to the User

```
-TYPE[+] text  
-TYPE[0] text  
-TYPE[1] text
```

where:

- +** 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.

*text*

Is all succeeding text including variable values supplied on the same command line. It sends the text to the screen, followed by a line feed. It remains on screen until scrolled off or replaced by a new screen.

The options +, 0, and 1 are used to pass printer control characters to the output device and are particularly useful for character printers. Options + and 1 do not work on IBM 3270-type terminals. -TYPE sends the text to the terminal as soon as it is encountered in the processing of a procedure.



## Example      Sending a Message to the Users

The following is an example of the use of -TYPE:

```
-* Version 1 6/30/85 SALERPT FOCSEXEC
-TYPE This report calculates percentage of returns
TABLE FILE SALES
.
.
.
```

**Note:** The -TYPE message need not be enclosed in quotation marks, since FOCUS understands that -TYPE signals a following textual message. If you use quotation marks, they will appear along with the message. This differs from the use of TYPE in MODIFY, where quotation marks are used as delimiters and must enclose informative text.

## Controlling Execution: -RUN, -EXIT, and -QUIT

Dialogue Manager enables you to manage the flow of execution with these commands:

- -RUN executes stacked commands and continues the procedure.
- -EXIT executes stacked commands and exits the procedure.
- -QUIT cancels execution and exits the procedure.

## Executing Stacked Commands and Continuing the Procedure: -RUN

The Dialogue Manager command -RUN causes immediate execution of all stacked FOCUS commands and closes any external files opened with -READ or -WRITE. Following execution, processing of the procedure continues with the line that follows -RUN.

### Example

### Executing Stacked Commands and Continuing the Procedure

The following example illustrates the use of -RUN to execute stacked code and then return to the procedure.

```
1. -TYPE This report calculates percentage of returns.
2. TABLE FILE SALES
   .
   .
   .
   END
3. -RUN
4. -TYPE This routine reports on data in the employee file.
   TABLE FILE EMPLOYEE
   .
   .
   .
   END
```

The procedure processes as follows:

1. The command -TYPE generates a message.
2. The FOCUS code is stacked.
3. The command -RUN causes the stacked commands to be executed and the output returned.
4. Processing continues with the line following -RUN. In this case, another message is sent and another TABLE request is initiated.

### Executing Stacked Commands and Exiting the Procedure: -EXIT

-EXIT forces execution of stacked FOCUS commands as soon as it is encountered. However, instead of returning to the procedure, -EXIT closes all external files, terminates the procedure, and, either returns you to the FOCUS prompt or to the calling procedure.

## Example Executing Stacked Commands and Exiting the Procedure

In the following example, either the first TABLE request or the second TABLE request will execute, but not both:

```
1. -TYPE This report calculates percentage of returns.
2. -IF &PROC EQ 'EMPLOYEE' GOTO EMPLOYEE;
3. -SALES
   TABLE FILE SALES
   .
   .
   .
   END
4. -EXIT
   -EMPLOYEE
   TABLE FILE EMPLOYEE
   .
   .
   .
   END
```

The procedure processes as follows:

1. The command -TYPE generates a message.
2. Assume the value passed to &PROC is SALES.  
The -IF test checks the value of &PROC. Since it is not equal to EMPLOYEE, control passes to the label -SALES.
3. The FOCUS code is stacked. Control passes to the next line, -EXIT.
4. The command -EXIT executes the stacked commands. The output is sent to the terminal or output device and the procedure is exited.

The TABLE 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 the -EXIT. The TABLE FILE EMPLOYEE request would execute and the procedure would automatically terminate.

## Canceling Execution of the Procedure: -QUIT

-QUIT cancels execution of any stacked commands and causes an immediate exit from the procedure.

This command is useful if tests or computations generate results that make additional processing unnecessary.

### *Example*

### Canceling Execution of the Procedure

The following example illustrates the use of -QUIT to cancel execution based on the results of an -IF test.

1. `-TYPE This report calculates percentage of returns.`  
`TABLE FILE SALES`  
`.`  
`.`  
`.`  
`END`
2. `-IF &CODE GT 'B10' OR &CODE EQ 'DONE' GOTO QUIT;`
3. `-QUIT`

The procedure processes as follows:

1. The command -TYPE generates a message. The FOCUS code is stacked.
2. Assume that the value of &CODE is B11.

The command -IF tests the value and passes control to -QUIT.

3. The command -QUIT cancels execution of the stacked commands and exits the procedure.

### Exiting FOCUS and Setting Return Codes: -QUIT FOCUS

The Dialogue Manager command -QUIT FOCUS causes an immediate exit not only from the procedure, but from FOCUS as well. It returns you to the operating system and sets a return code.

## Syntax

### How to Exit FOCUS and Set a Return Code

`-QUIT FOCUS [n|8]`

where:

`n|8`

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 value).

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.

## Branching

The execution flow of a procedure is determined with the following commands:

- `-GOTO`. Used for unconditional branching, `-GOTO` transfers control to a label.
- `-IF...GOTO`. Used for conditional branching, `-IF...GOTO` transfers control to a label depending on the outcome of a test.

## -GOTO Processing

Dialogue Manager processes a `-GOTO` as follows:

- It searches forward through the procedure for the target label. If it reaches the end without finding the label, it continues the search from the beginning of the procedure.
- The first time through a procedure, Dialogue Manager notes the addresses of all the labels so that they can be found immediately if needed again.
- 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 Unconditionally Branch With -GOTO**

```
-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 words that can be confused with functions or arithmetic or logical operations.

The label may precede or follow the -GOTO command in the procedure.

TYPE *text*

Optionally sends a message to a client application.

**Example****Unconditional Branching With -GOTO**

The following example “comments out” all the FOCUS code using an unconditional branch rather than -\* in front of every line:

```
-START TYPE PROCESSING BEGINS
-GOTO DONE
TABLE FILE SALES
PRINT UNIT_SOLD RETURNS
WHERE PROD_CODE BETWEEN '&CODE1' AND '&CODE2'
AND PRODUCT = '&PRODUCT'
BY PROD_CODE,CITY
END
-RUN
-DONE
```

## Syntax

### How to Conditionally Branch With -IF...GOTO

```
-IF expression [THEN] GOTO label1; [ELSE IF...;]  
                                [ELSE GOTO label2;
```

where:

*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.

GOTO *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 words that can be confused with functions or arithmetic or logical operations.

The label may precede or follow the -IF command in the procedure.

ELSE IF

Optionally specifies a compound -IF test. See *Compound -IF Tests* on page 4-19.

ELSE GOTO

Optionally passes control to *label2* when the -IF test fails.

The command -IF must end with a semicolon (;) to signal that all logic has been specified. Continuation lines must begin with a hyphen (-).

**Example****Conditional Branching With -IF...GOTO**

In the following example, control passes to the label -PRODSALES if &OPTION is equal to S. Otherwise, control falls through to the label -PRODRETURNS, the line following the -IF test.

```
-IF &OPTION EQ 'S' GOTO PRODSALES;
-PRODRETURNS
    TABLE FILE SALES
    .
    .
    .
    END
-EXIT
-PRODSALES
    TABLE FILE SALES
    .
    .
    .
    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 (-).

**Compound -IF Tests**

You can use compound -IF tests provided each test specifies a target label.

**Example****Using Compound -IF Tests**

In the following example, if the value of &OPTION is neither R nor S, the procedure terminates (GOTO QUIT). The -QUIT serves both as a target label for the GOTO and as an executable command.

```
-IF &OPTION EQ 'R' THEN GOTO PRODRETURNS ELSE IF
- &OPTION EQ 'S' THEN GOTO PRODSALES ELSE
- GOTO QUIT;
.
.
.
-QUIT
```



## Using Operators and Functions in -IF Tests

Expressions in an -IF test can include all FOCUS arithmetic and logical operators, as well as available functions or subroutines. See the *Creating Reports* manual for details.

### *Example*      **Testing System and Statistical Variables**

You can use system and statistical variables in -IF tests.

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
.
.
.
-IF &LINES NE 0 GOTO PRODSALES;
-EXIT
-PRODSALES
.
.
```

## Screening Values With -IF Tests

To ensure that a supplied value is valid in a procedure, you can test it for the following:

- Presence
- Length
- Type

For instance, you would not want to perform a numerical computation on a variable for which alphanumeric data has been supplied.

**Syntax****How to Test for the Presence of a Value**

```
-IF &name.EXIST [expression ]GOTO label...;
```

where:

*&name*

Is a user-supplied variable.

*.EXIST*

Indicates that you are testing 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.

*expression*

Is the remainder of a valid expression that uses &name.EXIST as an amper variable.

*GOTO label*

Specifies a label to branch to.

**Example****Testing for the Presence of a Variable**

In the following example, 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
```

## Syntax

### How to Test for the Length of a Value

```
-IF &name.LENGTH expression GOTO label...;
```

where:

*&name*

Is a user-supplied variable.

*.LENGTH*

Indicates that you are testing 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.

*expression*

Is the remainder of a valid expression after *&name* is expanded.

*GOTO label*

Specifies a label to branch to.

## Example

### Testing for Variable Length

In the following example, if the length of *&OPTION* is greater than one, 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;
```

**Syntax****How to Test for the Type of a Value**

```
-IF &name.TYPE expression GOTO label...;
```

where:

*&name*

Is a user-supplied variable.

TYPE

Indicates that you are testing 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  $10^9-1$  and can be 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) or the letter U (undefined) is passed to the expression.

*expression*

Is the remainder of a valid expression after *&name* is expanded.

GOTO *label*

Specifies a label to branch to.

**Example****Testing for Variable Type**

In the following example, 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.
```

## 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:

	<b>NORMAL</b>	<b>NODISPLAY</b>	<b>ERROR</b>
? HOLD	0	8	
? SU*	0	8	16
? JOIN	0	8	
? COMBINE	0	8	
? DEFINE	0	8	
? USE	0	8	
? LOAD	0	8	
? FILEDEF	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.

For example, if you are using Simultaneous Usage (SU), you must know if the FOCUS Database Server is available before you can begin 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
```

# Looping

The Dialogue Manager command -REPEAT allows looping in a procedure.

## Syntax

### How to Specify a Loop

```
-REPEAT label                n TIMES  
-REPEAT label                WHILE condition  
-REPEAT label                FOR &variable [FROM fromval] [TO toval] [STEP s]
```

where:

*label*

Identifies the end of 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 the only way to end the loop early is with -QUIT or -EXIT (you cannot change the number of times to execute the loop) or to branch out of the loop.

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.

FOR &*variable*

Is a variable that is tested at the start of each execution of the loop. It is compared with the value of *fromval* and *toval* (if supplied). The loop is executed only if &*variable* is less than or equal to *toval* (STEP is positive), or greater than or equal to *toval* (STEP is negative).

FROM *fromval*

Is a constant that is compared with &*variable* at the start of each execution of the loop. The default value is 1.

TO *toval*

Is a value that is compared with &*variable* at the start of each execution of the loop. The default is 1,000,000.

STEP *s*

Is a constant used to increment &*variable* at the end of each execution of the loop. It may be positive or negative. The default value is 1.

The parameters FROM, TO, and STEP can appear in any order.

## Example Using -REPEAT to Create a Loop

These examples illustrate how to write each of the syntactical elements of -REPEAT.

1. `-REPEAT label n TIMES`

Example:

```
-REPEAT LAB1 2 TIMES
-TYPE INSIDE
-LAB1 TYPE OUTSIDE
```

The output is:

```
INSIDE
INSIDE
OUTSIDE
```

2. `-REPEAT label WHILE condition`

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
```

3. `-REPEAT label FOR &variable FROM fromval TO toval STEP s`

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
```

## Ending a Loop

A loop can end in one of three ways:

- It executes in its entirety.
- A -QUIT or -EXIT is issued.
- A -GOTO is issued to a label outside of the loop.

**Note:** If you later issue another -GOTO to return to the loop, the loop proceeds from the point it left off.

## Using Expressions: -SET

The Dialogue Manager command -SET can be used in various ways to define values for variables in Dialogue Manager. You can compute new variables or recompute existing ones using arithmetic and logical expressions. You can also control loops, set indexes for variables, and call subroutines.

The following is a list of what can be included in a -SET expression and some specific rules for computations when using amper variables. Some calculations and special functions require that the amper variables have numeric values. FOCUS substitutes the value before placing the calculation in the stack. The variable does not have to have an I (integer) format, but the value for the variable must not contain alphanumeric characters. Note that the LAST operator used for reporting has no meaning in Dialogue Manager, nor do special MODIFY functions like FIND or LOOKUP.

- You can perform concatenations with the concatenation symbol. You must insert a space separating the amper variable from the concatenation symbol.
- You can use the DECODE function.
- You can use the EDIT function; however, its use is limited to the mask option.
- You can use the TRUNCATE function
- You can use the date functions.
- You can use subroutines.

For more information on expressions, functions, and subroutines, see the *Creating Reports* manual.



## Computing a New 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 Compute a New 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.

### 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
```

Note that literals are enclosed by single quotation marks. These are optional unless the literal contains embedded commas or blanks. To produce a literal that includes a single quotation mark, place two single quotation marks where you want one to appear.

## Using the DECODE Function

You can use the DECODE function to change a variable to an associated value.

### Example

### Assigning a Value to a Variable With DECODE

In the following 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 *Incorporating Multiple Procedures* on page 4-37) 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 *Creating Reports* manual.

## Using 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

### Using the EDIT Function With Amper Variables

In the following 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.

## Using the TRUNCATE Function

The Dialogue Manager TRUNCATE function removes trailing blanks from Dialogue Manager ampers 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 subroutine 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 subroutine of the same name can exist without conflict.

### Syntax

### How to Use the TRUNCATE Function

```
-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. However, 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

### Using the Dialogue Manager TRUNCATE Function

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
```

## Controlling a Loop With -SET

You can use the -SET command to control the repetition limit of a loop.

### Example

#### Controlling a Loop With -SET

In the following example, the variable &N is incremented using -SET and tested to terminate the loop:

```
1. -DEFAULTS &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
```

Execution proceeds in this way:

1. The -DEFAULTS command initializes the loop-controlling variable &N to 0.
2. -START is a Dialogue Manager label that begins the loop. It is the target of an unconditional -GOTO.

3. The -SET command increments the value of &N by one each time through the loop.
4. The FOCUS command EX SLRPT is stacked. The command -RUN then calls for the execution of the stacked command.
5. This -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 user and forces an exit. If the value of &N is 5 or less, control falls through to the next Dialogue Manager command.
6. The unconditional Dialogue Manager command -GOTO START causes the loop to repeat.

## Setting a Date

Natural date literals can be used in Dialogue Manager. They provide a way to take advantage of the powerful date handling capabilities of FOCUS. For more information on the FOCUS DATE format, see the *Creating Reports* manual.

### Example

### Setting Dates and Computing the Difference in Days

Consider the following example:

```
-SET &NOW= 'MAR 11 1999';  
-SET &LATER= '2000 11 MAR';  
-SET &DELAY = &LATER - &NOW;
```

The value of &DELAY is set to the difference, in days, between &LATER and &NOW.

#### Note:

- A computation that adds or subtracts a fixed number of days from a variable in DATE format is not yet supported.
- A date given to Dialogue Manager cannot exceed 20 characters, including spaces.
- Dialogue Manager accepts only full-format dates (that is, MDY or MDYY, in any order).

## Calling a Subroutine

Any function name encountered in a Dialogue Manager expression which is not recognized as a system standard name or FOCUS function is assumed to be a subroutine. These subroutines are externally programmed by users and stored in a library that is available at the time they are 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 their values from subroutines through -SET.

## Syntax

### How to Set a Variable Value Based on the Result From a Subroutine

```
-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 subroutine.

*argument*

Represents the argument(s) that must be passed to the subroutine. These arguments are converted to decimal format.

*format*

Is the predefined format of the result. This is used to convert the numeric format back to character representation. It must be enclosed in single quotation marks.

## Example

### Setting a Variable Value Based on the Result From a Subroutine

In the following example, FOCUS invokes the subroutine 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 Subroutine

The following is an alternate way of calling subroutines. The Dialogue Manager command causes the subroutine to be loaded and then executed. The syntax is

```
{-CMS } RUN routine, argument, ...
{-TSO } RUN routine, argument, ...
{-MVS } RUN routine, argument, ...
```

where:

*routine*

Is the name of the subroutine.

*argument*

Represents the argument(s) that must be passed to the subroutine. Arguments that are variables must have sizes predefined in prior -SET commands.

The numeric arguments to the subroutine are not automatically converted to D format in this syntax. Any required conversion must be done externally by the user or in the subroutine.

## Example

### Loading and Executing a Subroutine

The following is an example of the preceding syntax:

```
-PROMPT &MYCODE.A3.
-SET &MYNAME = ' ';
-SET &MYFACTOR = ' ' ;
-CMS RUN CODENAME, &MYCODE, &MYNAME, &MYFACTOR
```

In this example the program is CODENAME. The arguments that are variables are either prompted for or set at the beginning of the procedure and the values are then supplied for the arguments. Note that in this syntax the user program may use 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.



## Additional Facilities

Dialogue Manager supports a number of facilities for building applications. These facilities include:

- Creating startup files (profiles) that set overall environment conditions, which apply throughout your working session with FOCUS.
- Using `-INCLUDE` and `EXEC` to dynamically insert a procedure in another procedure, or to nest them up to four levels.
- Creating windows and menus for displaying information and collecting data in a procedure.
- Debugging procedures.
- Managing data integrity and security.
- Transferring data to and from non-FOCUS files.

## Establishing Startup Conditions

FOCUS supports a startup 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 FOCEXEC named PROFILE.

**Note:** It is possible to use an alternate FOCEXEC 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

Note the following example of a profile (under CMS):

```
USE
SALES FOCUS A1
MASTER FOCUS C1
END
CMS FILEDEF MYSAV DISK SAVE TEMP (LRECL 304 RECFM V
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 showing the current date and time is displayed:

```
FOCUS SESSION ON 03/11/99 AT 14:21:06
```

## Incorporating Multiple Procedures

Dialogue Manager supports dynamic inclusion of other procedures into a stored procedure at run time to enhance efficiency. There are two ways to do this:

- You can use the EXEC command in a procedure. The command will be stacked with other FOCUS commands and executed when an appropriate Dialogue Manager command forces execution of the stack. The procedure must be a fully executable procedure.
- The -INCLUDE command incorporates a file, which may be whole or partial procedures. A partial procedure could not be executed alone, but can be saved in a file and included in a calling procedure. This is particularly useful for procedures containing common header text, or partial processing cases that can be included at run time, based on tests and branches initiated in the original procedure. You can nest -INCLUDEs up to four levels.

The major difference between these two methods is when the procedure is executed. An EXEC command would be stacked and subsequently executed when the appropriate Dialogue Manager command is encountered, whereas -INCLUDE occurs immediately.

## Using -INCLUDE

Lines inserted from a -INCLUDE are incorporated into the calling procedure as if they had originally been placed there.

There are many more uses for -INCLUDE files:

- As a control over the user environment. The included procedure must be present and some switches set before the present procedure continues execution.
- As a security mechanism. The included procedure can be encrypted and a direct password set. For more information, see the *Describing Data* manual.
- The name of the included file can be determined by the procedure (for example, -INCLUDE &NEWLINES, where NEWLINES is a variable whose value is a file name). This can shorten the main procedure when there are many alternate procedures.

## Syntax

### How to Incorporate a File

```
-INCLUDE filename [filetype [filemode]]
```

where:

*filename*

Is the name of a FOCUS procedure.

*filetype*

Is the procedure's file type. If none is included, a file type of FOCEXEC is assumed.

*filemode*

Is the procedure's file mode. If none is included, a file mode of A is assumed.

**Example****Incorporating a File**

In this example, -INCLUDE searches for a file named DATERPT:

```
-IF &OPTION EQ S GOTO PRODSALES
-ELSE GOTO PRODRETURNS;
.
.
.
-PRODRETURNS
-INCLUDE DATERPT
-RUN
.
.
.
```

Assume that DATERPT is a procedure containing the following TABLE request:

```
TABLE FILE SALES
PRINT PROD_CODE UNIT_SOLD
BY STORE_CODE
IF PROD_CODE IS &PRODUCT
END
```

-INCLUDE incorporates this request into the calling procedure. FOCUS prompts for a value for the variable &PRODUCT as soon as the -INCLUDE is encountered. The ensuing -RUN forces the execution of this included TABLE request.

**Example****Incorporating Non-Executable Code**

You can use -INCLUDE to call files containing code that is not executable. For instance, a common heading used throughout all reports can be stored in a separate file and incorporated into any procedure as needed. For example,

```
TABLE FILE SALES
-INCLUDE SALEHEAD
SUM UNIT_SOLD AND RETURNS AND COMPUTE ...
```

where the SALEHEAD file contains:

```
HEADING
"THE ABC CORPORATION"
"RETAIL SALES DIVISION"
"MONTHLY SALES REPORT"
```

## Example Incorporating a Defined Field

As another example, a defined field can be placed in a separate file and called from a procedure as follows

```
-INCLUDE DEFRATIO
  TABLE FILE SALES
-INCLUDE SALEHEAD
  SUM UNIT_SOLD AND RETURNS AND RATIO
  BY CITY
.
.
.
```

where the DEFRATIO file contains:

```
DEFINE FILE SALES
RATIO/D5.2=(RETURNS/UNIT_SOLD);
END
```

This DEFINE will be dynamically included before the TABLE request executes.

## Nesting Procedures With -INCLUDE

Any number of different procedures can be invoked from a single calling procedure. You can also nest -INCLUDE commands within each other, up to four levels deep.

```
-PRODSALES
-INCLUDE FILE1
-RUN

      FILE1
      -INCLUDE FILE2
      -RUN

            FILE2
            -INCLUDE FILE3
            -RUN

                  FILE3
                  -INCLUDE FILE4
                  -RUN

                          FILE4
                          -RUN
```

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.

## Using EXEC

A procedure can also call another one with the command EXEC (EX). The called procedure must be fully executable. You can also pass values to variables on the command line.

### *Example* Using EXEC to Call a Procedure

In the following example, a procedure calls DATERTPT:

```
-IF &OPTION EQ 'S' GOTO PRODSALES ELSE GOTO PRODRETURNS;
.
.
.
-PRODRETURNS
  EX DATERTPT
.
.
.
-RUN
```

**Note:** If the last executable command in the called procedure is a -CRTFORM, control will not be 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 is 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 the procedure lines. The terminal then “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. Depending on what you want, you could supply:

```
BY STORE_CODE
END
```

Or, alternatively:

```
IF CITY IS STAMFORD
BY STORE_CODE
END
```

## Debugging With &ECHO

It can be helpful to display command lines as they execute in order to test and debug procedures. The variable &ECHO is available for this purpose.

### Syntax      How to Display Command Lines as They Execute

```
&ECHO = display
```

Valid values are:

ON

Displays lines that are expanded and stacked for execution.

ALL

Displays Dialogue Manager commands as well as lines that are expanded and stacked for execution.

OFF

Suppresses display of both stacked lines and Dialogue Manager commands. This value is the default.

You can set &ECHO through -DEFAULTS, -SET, or on the command line. For example, you can set ECHO to ALL for the execution of the procedure SLRPT using any of the following commands:

```
-DEFAULTS &ECHO = ALL
```

or

```
-SET &ECHO = ALL;
```

or

```
EX SLRPT ECHO = ON
```

If you use -SET or -DEFAULTS and place it in the procedure, display begins from that point in the procedure, and can be turned off and on again at any other point in the procedure.

Note that if the procedure is encrypted, &ECHO automatically receives the value OFF, regardless of the value that is assigned explicitly.

## Testing Dialogue Manager Command Logic With &STACK

To test the logic of Dialogue Manager commands, you can run the procedure but prevent actual execution of the stacked commands by setting the &STACK variable.

### *Syntax*

### How to Test Dialogue Manager Command Logic

```
&STACK = {ON|OFF}
```

where:

ON

Results in normal execution of stacked commands. This value is the default.

OFF

Prevents 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.



You can set &STACK through -DEFAULTS, -SET, or on the command line. For example, you can set &STACK to OFF for the execution of the procedure SLRPT using any of the following commands:

```
-DEFAULTS &STACK = OFF
```

or

```
-SET &STACK = OFF;
```

or

```
EX SLRPT STACK = OFF
```

This 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.

## Locking Procedure Users Out of FOCUS

Normally, 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. Then 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.

## Writing to Files: -WRITE

In addition to conducting a dialogue with the user, Dialogue Manager can read from and write to files. For information on reading values from files, see *Supplying Values Without Prompting* on page 4-68.

The Dialogue Manager -WRITE enables you to write lines of text to a file.

### Syntax

#### How to Write to a File

```
-WRITE ddname [NOCLOSE] text
```

where:

*ddname*

Is the logical name of the file as defined to FOCUS using FILEDEF, 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. If you wish 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 under CMS to add new text:

```
-CMS FILEDEF PASS DISK PASS DATA (DISP MOD
-WRITE PASS &DIV &RED &TEST RESULT IS,
- &RECORDS AT END OF RUN
```

## Example

## Reading From and Writing to Sequential Files

The following example illustrates reading from and writing to sequential files and the use of operating system commands (in this example, CMS). The numbers in the margin refer to notes that follow the example.

1. -TOP
2. -PROMPT &CITY. ENTER NAME OF CITY -- TYPE QUIT WHEN DONE.
3. -CMS FILEDEF PASS DISK PASS DATA A (LRECL 80 RECFM FB
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. -CMS FILEDEF LOG DISK LOG DATA A1 (LRECL 80 RECFM FB  
    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 paragraphs 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. `FILEDEF` defines and opens a file named `PASS`.
4. `-WRITE` writes the value of `&CITY` to the non-FOCUS file named `PASS`. In this case the value written is `STAMFORD`.

- RUN executes the stacked TABLE request. In this case, a non-FOCUS file named INFO is created with the SAVE command. This is a sequential file, containing the result of the TABLE request as shown below.

```
NUMBER OF RECORDS IN TABLE= 7 LINES= 7
(BEFORE TOTAL TESTS)

EBCDIC 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

DEFAULT FILEDEF ISSUED

FILEDEF INFO DISK INFO FOCTEMP A1 (LRECL 30 BLKSIZE 300 RECFM F6)
>
>
```

- FILEDEF defines a log file for the subsequent MODIFY request.
- 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.foc ON 04/23/93 AT 12.28.38

TRANSACTIONS: TOTAL=      1  ACCEPTED=      1  REJECTED=      0
SEGMENTS:      INPUT=      0  UPDATED =      0  DELETED =      1
```

- The next -RUN executes another procedure called SLRPT3.
- READ reads the value for &CITY from the non-FOCUS 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.

PAGE 1					
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
CALCULATING AS OF 04/23/93					

11. -GOTO TOP routes control to the top.
12. When the user types QUIT, processing ends.

## Using Variables in Procedures

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.

Variables can be used only in procedures. They are ignored if you use them while creating reports live at the terminal. Values for variables may be supplied either directly on the command line when you execute the procedure, or through the -DEFAULTS command, the -SET command, or a -READ command in the procedure itself.

This topic describes how to use amper variables in procedures and how to supply values for them. Variables fall into two classifications:

- 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).
- System, statistical, and special variables have values that the system automatically resolves whenever you request them.

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.

The maximum number of local, global, system, statistical, special and index variables available in a procedure is 512. Approximately 30 are reserved for use by FOCUS.

Additionally, Dialogue Manager supports four types of prompting. You can alter the execution flow of your procedure, or change the substance of the request based on the values entered. These are

- **Direct Prompting with -PROMPT:** You can request a set of values before they are actually needed. You can write your own text for these prompts and then validate the entered values to confirm that they fit a preset list of acceptable items or match a predefined format.
- **Full-Screen Data Entry with -CRTFORM:** 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.

-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.

- **Selecting Items from a Menu with -WINDOW:** 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.
- **Implied 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.

## Querying the Values of Variables

Amper variable values can be queried during execution.

### Syntax

#### How to Query the Values of Variables

```
-? &[string]
```

where:

*string*

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 their values:

```
CURRENTLY DEFINED & VARIABLES:
```

Note that this is a Dialogue Manager query. Since local variables do not exist outside a procedure, no similar query is available from the FOCUS command line.

#### Querying Parameter Value Settings

There is a Dialogue Manager query that enables you to capture previously defined SET parameter values in amper variables.

### Syntax

#### How to Query Parameter Value Settings

```
-? SET parameter ampervar
```

where:

*parameter*

Is any valid FOCUS setting that may be queried with the ? SET or ? SET ALL command.

*ampervar*

Is the name of the variable where the value is to be stored.

### Example

#### Querying a Parameter Value Setting

For example, 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.



## 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**

In the following example, &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 values when you execute the procedure:

```
EX SLRPT CITY = STAMFORD, CODE1=B10, CODE2=B20
```

The procedure looks like this before it processes:

```
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
```

Values supplied for local variables remain current in the procedure. That is, all instances of the variables receive the values supplied. However, the values are not passed to other procedures containing the same variables (that is, &CODE1 and &CODE2 in another procedure). The values disappear after SLRPT has finished processing.

## 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 Chapter 5, *Defining a Word Substitution*. They 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 will 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 is an example of a procedure containing global variables:

```
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
```

### Syntax

#### How to Query the Values of Global Variables

Since global variable values remain current throughout the FOCUS session, it is helpful to be able to display their values on demand. Do this by issuing the following command,

```
? &&
```

which displays the values of all global variables in use during the FOCUS session.

## Example Querying the Values of Global Variables

The following example displays the values of three global variables:

```
>
? &&
  &&CITY          STAMFORD
  &&CODE1         B10
  &&CODE2         B20
>
```

## System Variables

FOCUS automatically substitutes values for system variables encountered in a Dialogue Manager request. System-supplied variables cannot be overridden. For example, you can use the system variable &DATE to automatically incorporate the system date in your request.

## 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.
&DATEfmt	Any date format.	Returns the current date, where fmt can be any combination of YYMD, MDYY, etc.
&DMY	DDMMYY	Returns the current date.
&DMYY	DDMMCCYY	Returns the current (four-digit year) date.
&FOCCPU	milliseconds	Calculates the OS CPU time. MVS only. In CMS, this returns the same value as &FOCTIME.
&FOEXTTRM	ON OFF	Indicates the availability of extended terminal attributes.
&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; NEW means that they are; and NOTRUNC means that they are supported, but unique truncations of field names cannot be used.

Variable	Format or Value	Description
&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.
&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	CMS CRJE MSO OS TSO	Identifies the operating environment.
&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).
&FOCSBORDER	ON OFF	Whether solid borders will be used in full-screen mode.
&FOCSYSTYP	HIPER CP/A	CMS system type.

Variable	Format or Value	Description
&FOCTMPDSK	A ... Z	Identifies the disk where FOCUS places temporary work files (for example, HOLD files). CMS only.
&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.
&FOCTTIME	milliseconds	Calculates total CPU time. CMS only.
&FOCVTIME	milliseconds	Calculates virtual CPU time. CMS only.
&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.
&MDY	MMDDYY	Returns the current date. The format makes this variable useful for numerical comparisons.
&MDYY	MMDDCCYY	Returns the current (four-digit year) date.
&RETCODE	numeric	Returns the return code set upon execution of an operating system command. Executes all FOCUS commands in the FOCSTACK just as the -RUN command would.
&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 subroutine.
&YMD	YYMMDD	Returns the current date.
&YYMD	CCYYMMDD	Returns the current (four-digit year) date.

**Example**      **Using the System Variable &DATE**

The following example illustrates the use of a system variable in a request:

```
TABLE FILE SALES
.
.
.
FOOTING "CALCULATED AS OF &DATEMDYY"
END
-EXIT
```

The system variable &DATEMDYY ensures that the date that appears in the report is always the current system date.

**Example**      **Using the System Variable &FOCFOCEXEC**

This next 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 will produce the following:

PAGE	1
REPORT: DA0219	-- EMPLOYEE SALARIES
EMP_ID	CURR_SAL
-----	-----
071382660	\$11,000.00
112847612	\$13,200.00
115360218	\$.00
117593129	\$18,480.00
119265415	\$9,500.00
119329144	\$29,700.00
121495681	\$.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
```

## Displaying a Date Variable Containing a Four-Digit Year

You can display a date variable containing a 4-digit year without separators. The variables are &YYMD, &MDYY, and &DMYY. These variables complement the 2-digit year variables &YMD, &MDY, and &DMY.

### *Example*

#### Using the System Variable &YYMD

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 18, 1998 is:

```
PAGE      1

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 can automatically supply 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.
&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. This variable can be displayed or tested 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 will hold the number of the most recent error. If no error occurred, &FOCERRNUM will have 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.

Variable	Description
&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.
&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****Using &LINES to Control Execution of a Request**

The following example illustrates how to use the statistical variable &LINES to control execution of a request:

```
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
ON TABLE HOLD
END
-RUN
-IF &LINES EQ 0 GOTO NORECORDS;
MODIFY FILE SALES
.
.
.
DATA ON HOLD
END
-RUN
-NORECORDS
-TYPE No record satisfies this report request
-QUIT
```

In the example, the system calculates the statistical variable &LINES (the number of lines produced by the TABLE request). If the number is 0, there are no lines in the report; -QUIT tells FOCUS to halt processing and the user is returned to the FOCUS prompt. If &LINES is greater than 0, processing continues to the MODIFY request.

## Syntax

### How to Query the Values of Statistical Variables

You can query the current value of all statistical variables except &FOCDISORG and &FOCERRNUM by typing the query command

? STAT

from the FOCUS prompt.

## 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.
&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 Chapter 9, <i>Designing Windows With Window Painter</i> ).
&WINDOWVALUE	Holds the return value of the last window activated by the most recently executed -WINDOW command (see Chapter 9, <i>Designing Windows With Window Painter</i> ).

## Using Variables to Alter Commands

A variable can refer to a FOCUS command or to a particular field. In this way, the command structure of a procedure can be determined by the value of the variable.

### *Example*      **Using a Field Variable**

In the following example, the variable &FIELD determines the field to print in the TABLE request. For example, &FIELD could have the value RETURNS, DAMAGED, or UNIT\_SOLD from a file named SALES.

```
TABLE FILE SALES
.
.
.
PRINT &FIELD
BY PROD_CODE
.
.
.
```

## Evaluating a Variable Immediately

The .EVAL operator enables you to evaluate a variable's value immediately, making it possible to change a procedure dynamically. It is used for substitution and re-evaluated by Dialogue Manager.

### *Syntax*      **How to Evaluate a Variable**

.EVAL uses the following syntax

```
[&]&variable.EVAL
```

where:

*variable*

Is a local or global amper variable.

When the command procedure is executed, the expression is replaced with the value of the specified variable before any other action is performed.

## Example Excluding and Including the .EVAL Operator

Without the .EVAL operator, an amper variable cannot be used in place of some FOCUS commands, as shown by the following example:

```
-SET &A='-TYPE';  
&A HELLO
```

This example's output shows that FOCUS does not recognize the value of &A:

```
UNKNOWN FOCUS COMMAND -TYPE
```

Appending the .EVAL operator to the &A amper variable makes it possible for FOCUS to interpret the variable correctly. For example, adding the .EVAL operator as follows,

```
-SET &A='-TYPE';  
&A.EVAL HELLO
```

produces the following output:

```
HELLO  
>>
```

## Example Evaluating a Variable Immediately

The .EVAL operator is particularly useful in modifying code at run time. 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:

```
1. -SET &R='IF COUNTRY IS ENGLAND';  
2. -IF &Y EQ 'YES' THEN GOTO START;  
3. -SET &R = '-*';  
   -START  
4.   TABLE FILE CAR  
     PRINT CAR BY COUNTRY  
5.   &R.EVAL  
     END
```

The procedure executes as follows:

1. The procedure sets the value of &R to 'IF COUNTRY IS ENGLAND'.
2. If the &Y is YES, the procedure branches to the START label, bypassing the second -SET command.
3. If the &Y is NO, the procedure continues to the second -SET command, which sets &R to '-\*', which is a comment.
4. The report request is stacked.

5. The procedure evaluates &R's value. If the user wanted a record selection test, &R's value is 'IF COUNTRY IS ENGLAND' and this line is stacked.  
  
If the user did not want a record selection test, &R's value is '-\*' and this line is ignored.

## Concatenating Variables

You can append a variable to a character string or you can combine two or more variables and/or literals. See the *Creating Reports* manual for full details 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 removes any trailing blanks from *&name1*. Conversely, weak concatenation, indicated by the symbol |, preserves any trailing blanks in *&name1*.

## Supplying Values for Variables at Run Time

When you design a Dialogue Manager procedure, you must decide how the variables in the procedure will acquire values. Values for variables can be supplied in two ways:

- When you call a procedure. You can include the variable names and their corresponding values as parameters in an EXEC command that calls one procedure from another.
- Directly in a procedure. The Dialogue Manager commands -DEFAULTS, -SET, and -READ enable you to supply values directly in a procedure.

### Example

### Supplying Values for Variables

The example in this topic illustrates the use of the commands -DEFAULTS and -SET to supply values for variables. In the example, the user supplies the value of &CODE1, &CODE2, and &REGIONMGR as prompted by an HTML form.

The numbers to the left of the example apply to the notes that follow:

1. -DEFAULTS &VERB='SUM'
2. -SET &CITY=IF &CODE1 GT 'B09' THEN 'STAMFORD' ELSE 'UNIONDALE';
3. -TYPE REGIONAL MANAGER FOR &CITY
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 &DATEMDYY"  
       END
6. -RUN

The procedure executes as follows:

1. The -DEFAULTS command sets the value of &VERB to SUM.
2. The -SET command supplies the value for &CITY depending on the value for &CODE1 typed by the user on the form. Because the user typed B10 for &CODE1, the value for &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 on the form. FOCUS supplies the current date 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 03/11/99"
END
```

## Reference

### General Rules for Supplying Variable Values

The following general rules apply to values for variables:

- The maximum length of a variable value to be displayed on the screen is 80 characters.
- A physical FOCSTACK line with values substituted for variables cannot exceed 80 characters; therefore, you should not use variable values longer than 80 characters.
- If a value contains an embedded space, comma (,) or equal sign (=), you must enclose the variable name in single quotation marks when you use it in an expression. For example, if the value for &CITY is NY, NY, you must refer to the variable as '&CITY' 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.



## Supplying Values Without Prompting

There are several ways to supply values for local and global variables besides prompting methods. These are outlined below:

- Supplying values on the command line: You can supply values when you execute the procedure.
- Supplying values with -DEFAULTS: You supply initial default values in the procedure to ensure that you will not be implicitly prompted for the value.
- Supplying values with -SET: You supply values by setting them in the procedure using the -SET command. The values can be constants or the result of an expression.
- Supplying values with -READ: You can supply values by reading them in from a sequential file.

### Supplying Values on the Command Line

When the user knows the values required by a procedure, they can be typed on the command line following the name of the procedure itself. This saves time, since FOCUS now has values to pass to each local or global variable and the user will not be prompted to supply them.

#### *Example*

### Supplying Values on the Command Line

Consider the following procedure:

```
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
```

In order to execute this procedure and supply values for the variables on the command line, the user would type the following:

```
EX SLRPT CITY = STAMFORD, CODE1=B10, CODE2=B20
```

**Syntax****How to Supply Values on the Command Line**

Each name-value pair must have the syntactic form

```
name=value
```

and pairs must be separated by commas. It is not necessary to enter the name-value pairs in the order that they are encountered in the procedure.

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
```

It is acceptable to supply some but not all values on the command line, in which case, values not supplied will trigger prompts to the terminal.

To supply global amper variable values on the command line, you must supply the double ampersand prefix, as in the following example:

```
EX SLRPT &&GLOBAL=value, CITY = STAMFORD, CODE1=B10, CODE2=B20
```

**Example****Using Positional Variables**

When the variable is numbered (a positional variable; for example, &1, &2, &3) there is no need to specify the name, in this case a number, on the command line. FOCUS matches the values, one by one to the positional variables as they are encountered in the procedure. Therefore, it is vital to enter the appropriate value for each variable, in the proper order.

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 command line for entry of positional values should read:

```
EX SLRPT STAMFORD, B10, B20
```

## **Example**      **Mixing Named and Positional Variables**

You can mix named and positional variables freely on the command line, providing that names are associated with values for named variables and values are supplied for positional variables in the order that these variables are numbered in the procedure. For example:

```
EX SLRPT CITY = STAMFORD, B10, B20, VERB=COUNT
```

## **Supplying Values With -DEFAULTS**

The Dialogue Manager command `-DEFAULTS` supplies an initial (default) value for a variable that had no value before the command was processed. It ensures that values will be passed to variables whether or not they are provided elsewhere.

### **Syntax**      **How to Supply Default Values**

```
-DEFAULTS &[&]name=value [...]
```

where:

`&name`

Is the name of the variable.

`value`

Is the default value assigned to the variable.

### **Example**      **Supplying Default Values**

In the following example, `-DEFAULTS` sets default values for `&CITY` and `&REGIONMGR`.

```
-DEFAULTS &CITY=STAMFORD, &REGIONMGR=SMITH  
TABLE FILE SALES  
.  
.  
.
```

### **Overriding Default Values**

You can override default values by supplying new values on the command line or by an explicit prompt.

## Supplying Values With -SET

With -SET, you can assign a value computed in an expression.

### Syntax How to Set a Variable Value

```
-SET [&]name=expression;
```

where:

*&name*

Is the name of the variable.

*expression;*

Is a valid literal, arithmetic, or logical expression. Expressions can occupy several lines, so you should end the command with a semicolon (;).

### Example Setting Variable Values

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

Single quotation marks around a literal is optional unless it contains embedded blanks, commas, or equal signs, in which case you must include them as shown:

```
-SET &NAME='JOHN DOE';
```

To assign a literal value that includes a single quotation mark, place two single quotation marks where you want one to appear:

```
-SET &NAME='JOHN O''HARA';
```

## Supplying Values With -READ

You can supply values for variables by reading them from a sequential file.

### Syntax

### How to Supply Values With -READ

```
-READ ddname[,] [NOCLOSE] &name[.format.][,] ...
```

where:

*ddname*

Is the logical name of the file as defined to FOCUS using FILEDEF. (When using MVS, use 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 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. 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 Data and Testing a System Variable

The example below reads data from EXTFILE, a fixed format file that contains the following data:

```
STAMFORDB10B20
```

The example tests the system variable &IORETURN. If there is no record to be read, the value of &IORETURN is not equal to zero and the procedure branches to the label after the TABLE request.

```
-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
.
.
.
```

## Direct Prompting 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.

## Full-Screen Data Entry 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.

## Selecting Data From 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 can collect data by prompting a user to select a value, to enter a value, or to press a program function (PF) key.

## Implied Prompting

If a value is not supplied by any other means for a variable, FOCUS automatically prompts the user for the value. This is known as an implied prompt. These 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
REGIONMGR=
smith
>
```

At the point when all variables have values, FOCUS processes the report request.

## Verifying Input Values

Input values can be verified in the following ways:

- Format conditions can be specified against which the entered values are compared.
- Lists of acceptable values can be specified against which the entered values are compared.
- Text can be supplied that either explains what type of value is needed or lists choices of acceptable values on the screen.

## Using Format Specifications

You can specify variables with format conditions against which the entered values are compared. If the entered values do not have the specified format, FOCUS prints error messages and prompts the user again for the value(s).

Alphanumeric formats are described by the letter A followed by the number of characters. The number of characters can be from 1 to 255. Integer formats are described by the letter I followed by the number of digits to be entered. The number can be from 1 to 9 (value must be less than  $2^{31}-1$ ).

The description of the format must be enclosed by periods.

If you test field names against input variable values, we recommend that you 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**

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:

```
STORECODE=> cc14
(FOC291) VALUE IN PROMPT REPLY EXCEEDS 03 CHARS:CC14
STORECODE=>
```

### **Using Lists of Value Ranges**

Variables can be further customized by providing lists of values describing the acceptable range of prompted responses. 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.

### *Example*      **Providing a List of Valid Values**

For example:

```
-PROMPT &CITY.(STAMFORD,UNIONDALE,NEWARK).
```

A message is printed if the user does not respond with one of the replies on the list. This is followed by a display of the value list. Finally, another prompt is issued for the needed value. For example:

```
CITY>union
PLEASE CHOOSE ONE OF THE FOLLOWING:
STAMFORD, UNIONDALE, NEWARK
CITY>
```

## Syntax

### How to Use a Variable to Provide the Reply List

You can also use a variable to provide the reply list, in conjunction with the -SET command. 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.

## Example

### Using a Variable to Provide the Reply List

For example:

```
-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.

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**

A variable can be further specified with customized text explaining the prompt at the screen.

For example:

```
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
```

## Dialogue Manager Quick Reference

This topic describes all the Dialogue Manager commands in alphabetical order. The following commands are included:

-*	-?	-CLOSE
-CLOSE *	-CMS	-CMS RUN
-CRTCLEAR	-CRTFORM	-DEFAULTS
-DEFAULTS	-EXIT	-GOTO
-IF	-INCLUDE	-label
-MVS RUN	-PASS	-PROMPT
-QUIT	-READ	-REPEAT
-RUN	-SET	-TSO RUN
-TYPE	-WINDOW	-WRITE
_"_"		

**Command:**        -\*

**Function:**        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.

**Syntax:**            -*\* text*

where:

*text*

Is a comment. A space is not required between -\* and text.

**Command:**        -?

**Function:**        The command -? displays the current value of a local variable.

**Syntax:**            -? *&[string]*

where:

*string*

Is an optional 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.

<b>Command:</b>	-CLOSE
Function:	-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.
Syntax:	<code>-CLOSE <i>ddname</i>   *</code> where: <code><i>ddname</i></code> Is the <i>ddname</i> of the open file described to FOCUS via an allocation (TSO, MSO) or FILEDEF (CMS) command.  * Closes all -READ and -WRITE files that are currently open.
<b>Command:</b>	-CMS
Function:	CMS executes a CMS operating system command from within Dialogue Manager.
Syntax:	<code>CMS <i>command</i></code> where: <code><i>command</i></code> Is a CMS command.
<b>Command:</b>	-CMS RUN
Function:	In CMS, loads and executes the specified user-written subroutine. SET can also execute user-written programs.
Syntax:	<code>-CMS RUN <i>subroutine</i></code> where: <code><i>subroutine</i></code> Is a FOCUS user-written subroutine.
<b>Command:</b>	-CRTCLEAR
Function:	Clears the current screen display.
Syntax:	<code>-CRTCLEAR</code>

<b>Command:</b>	-CRTFORM
<b>Function:</b>	<p>Creates forms to prompt the user for values for variables.</p> <p>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.</p> <p>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.</p> <p>All FIDEL facilities are available to -CRTFORM except HEIGHT, WIDTH, and LINE.</p> <p>CRTFORM in MODIFY functions identically to -CRTFORM in Dialogue Manager.</p> <p>See -PROMPT.</p>
<b>Syntax:</b>	<p>-CRTFORM [TYPE <i>n</i>] [BEGIN END [LOWER UPPER]]</p> <p>where:</p> <p>-CRTFORM Invokes FIDEL and signals the beginning of the screen form.</p> <p>TYPE <i>n</i> Enables you to define the number of lines (<i>n</i>) to reserve for messages. You can specify a number from 1 to 4. The default is 4.</p> <p>BEGIN Supports the use of other Dialogue Manager commands to help build the form.</p> <p>END Signals the end of the -CRTFORM. Used with -CRTFORM BEGIN.</p> <p>LOWER Reads lowercase data from the screen. Once you specify LOWER, every screen thereafter is a lowercase screen until you specify otherwise.</p> <p>UPPER Translates lowercase letters to uppercase. This is the default.</p>

<b>Command:</b>	-DEFAULTS
<b>Syntax:</b>	<code>-DEFAULTS &amp;name=value, &amp;name=value...</code> where: <code>name</code> Is the variable name. <code>value</code> Is the variable value.
<b>Function:</b>	Sets initial values for the named variables in the procedure.  You can override -DEFAULTS 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 -DEFAULTS.  -DEFAULTS guarantees that the variables are always given a value and therefore that it will execute correctly.  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. See also -SET.
<b>Command:</b>	-EXIT
<b>Function:</b>	-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. In other words, the procedure ends after the last line is read.
<b>Syntax:</b>	<code>-EXIT</code>



**Command:** -GOTO

**Function:** -GOTO forces an unconditional branch to the 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.

**Syntax:** `-GOTO label`  
`:`  
`:`  
`-label [TYPE text]`

**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 and logical operations, and so on.  
`TYPE text`  
Optionally sends a message to the client application.

**Command:** -HTMLFORM

**Function:** For use with the Web Interface to FOCUS.

**Syntax:** `-HTMLFORM`

**Command:** -IF

**Function:** -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.

**Syntax:** `-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

Optionally passes control to *label2* when the -IF test fails.

ELSE IF

Optionally specifies a compound -IF test.

The semicolon (;) is required at the end of the command.

Continuation lines must begin with a hyphen (-).

<b>Command:</b>	<b>-INCLUDE</b>
<b>Function:</b>	<p>Specifies another procedure to be attached 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 <b>-INCLUDE</b>.</p> <p>When using <b>-INCLUDE</b>, you may not branch to a label outside of the specified procedure.</p> <p>A procedure may contain more than one <b>-INCLUDE</b>. Up to four <b>-INCLUDEs</b> may be nested.</p> <p>You may use any valid command in a <b>-INCLUDE</b>.</p> <p><b>EXEC</b> may also be used to execute a procedure inside another procedure.</p>
<b>Syntax:</b>	<p><code>-INCLUDE filename [filetype [filemode]]</code></p> <p>where:</p> <p><i>filename</i> Is the procedure to be incorporated in the calling procedure.</p> <p><i>filetype</i> Is the procedure's file type. If none is included, a file type of <b>FOCEXEC</b> is assumed.</p> <p><i>filemode</i> Is the procedure's file mode. If none is included, a file mode of <b>A</b> is assumed.</p>
<b>Command:</b>	<b>-label</b>
<b>Function:</b>	A label is the target of a <b>-GOTO</b> or <b>-IF</b> command.
<b>Syntax:</b>	<p><code>-label [TYPE message]</code></p> <p>where:</p> <p><i>label</i> Is a user-supplied name of up to 12 characters that identifies the target for a branch.</p> <p>Do not use embedded blanks or the name of any other Dialogue Manager command except <b>-QUIT</b> or <b>-EXIT</b>. Do not use words that can be confused with functions, arithmetic or logical operations, and so on.</p> <p><i>TYPE message</i> Optionally sends a message to the client application.</p>

**Command:** -MVS RUN

Function: Same as -TSO RUN.

Syntax: *-MVS RUN*

**Command:** -PASS

Function: Passwords can be directly issued and controlled by the Dialogue Manager. This is especially useful to specify 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 a password value is prompted for and assigned.

The PASS command provides the same function at the command level, as does the PASS parameter of the SET command.

Syntax: *-PASS password*

where:

*password*

Is a password or a variable containing a password.

<b>Command:</b>	-PROMPT
<b>Function:</b>	<p>-PROMPT types a message to the terminal and reads the reply from the user. This reply assigns a value to the variable named.</p> <p>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.</p> <p>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.</p> <p><b>Note:</b> You cannot use format and list together.</p> <p>In MODIFY, PROMPT specifies additional data input needs.</p> <p>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.</p> <p>See -CRTFORM.</p>
<b>Syntax:</b>	<pre>-PROMPT &amp;name [ [.format] (.list) ] [.text]. ]</pre> <p>where:</p> <p><i>&amp;name</i> Is a user-defined variable.</p> <p><i>format</i> Optionally specifies alphanumeric or integer data type and length.</p> <p><i>text</i> Optionally specifies prompting text that appears on the screen. Must be delimited by periods.</p> <p><i>list</i> Optionally specifies a range of acceptable responses. Must be enclosed in parentheses.</p>

**Command:** -QUIT

**Syntax:** -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).

**Function:**

Forces an immediate exit from the procedure. Lines that have been stacked 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.

See also -RUN and -EXIT.

**Command:** -READ

**Function:** Reads data from non-FOCUS files. -READ can access data in either fixed or free form.  
See -WRITE.

**Syntax:** `-READ ddname[, ] [NOCLOSE] &name[.format.][, ] ...`

where:

*ddname*

Is the logical name of the file as defined to FOCUS using FILEDEF (or, for MVS, 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. Note that format must be delimited by periods. The format is ignored for comma-delimited files.

<b>Command:</b>	-REPEAT
<b>Function:</b>	<p>-REPEAT allows looping in a procedure.</p> <p>The parameters FROM, TO, and STEP can appear in any order.</p> <p>A loop ends when any of the following occurs:</p> <ul style="list-style-type: none"><li>• It is executed in its entirety.</li><li>• A -QUIT or -EXIT is issued.</li><li>• 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.</li></ul>
<b>Syntax:</b>	<pre>-REPEAT <i>label</i> <i>n</i> TIMES -REPEAT <i>label</i> WHILE <i>condition</i> -REPEAT <i>label</i> FOR <i>&amp;variable</i> [FROM <i>fromval</i>] [TO <i>toval</i>] [STEP <i>s</i>]</pre> <p>where:</p> <p><i>label</i></p> <p>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.</p> <p><i>n</i> TIMES</p> <p>Specifies the number of times to execute the loop. The value of <i>n</i> can be a local variable, a global variable, or a constant. If it is a variable, it is evaluated only once, so the only way to end the loop early is with -QUIT or -EXIT (you cannot change the number of times to execute the loop).</p> <p>WHILE <i>condition</i></p> <p>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.</p> <p>FOR <i>&amp;variable</i></p> <p>Is a variable that is tested at the start of each execution of the loop. It is compared with the value of <i>fromval</i> and <i>toval</i> (if supplied). The loop is executed only if <i>&amp;variable</i> is less than or equal to <i>toval</i> (STEP is positive), or greater than or equal to <i>toval</i> (STEP is negative).</p> <p>FROM <i>fromval</i></p> <p>Is a constant that is compared with <i>&amp;variable</i> at the start of each execution of the loop. The default value is 1.</p>



*TO toval*

Is a value against which *&variable* is tested. The default is 1,000,000.

*STEP s*

Is a constant used to increment *&variable* at the end of each execution of the loop. It may be positive or negative. The default value is 1.

**Command:**

-RUN

**Function:**

-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.

**Syntax:**

-RUN

**Command:**

-SET

**Function:**

-SET assigns a literal value to a variable, or a value that is computed in an arithmetic or logical expression.

Single quotation marks around a literal value are optional unless it contains embedded blanks or commas, in which case you must include them.

**Syntax:**

-SET *&[&]name=expression;*

where:

*&name*

Is the name of a variable whose value will be set.

*expression*

Is a valid expression. Expressions can occupy several lines, so end the command with a semicolon (;).

<b>Command:</b>	-TSO RUN
Function:	In TSO, loads and executes the specified user-written subroutine. <b>Note:</b> The prefix -TSO can be used only with RUN. -SET can also execute user-written programs.
Syntax:	<code>-TSO RUN <i>subroutine</i></code> where: <code><i>subroutine</i></code> Is the name of a FOCUS user-written subroutine.
<b>Command:</b>	-TYPE
Function:	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 will be displayed in their 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.
Syntax:	<code>-TYPE[+] <i>text</i></code> <code>-TYPE[0] <i>text</i></code> <code>-TYPE[1] <i>text</i></code> where: <code>-TYPE1</code> Sends the text after issuing a page eject. <code>-TYPE0</code> Sends the text after skipping a line. <code>-TYPE+</code> Sends the text but does not add a line feed. <code><i>text</i></code> Is a character string that fits on a line.

**Command:** -WINDOW

**Function:** 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.

**Syntax:** `-WINDOW windowfile windowname`  
`[PFKEY|NOPFKEY] [GETHOLD] [BLANK|NOBLANK]`  
`[CLEAR|NOCLEAR]`

**where:**

*windowfile*  
Identifies the file in which the windows are stored. In CMS, this is a file name. The file must have a file type of FMU.  
In MVS/TSO, this is a member name. The member must belong to a PDS allocated to ddname FMU.

*windowname*  
Identifies which window in the file will be 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.

**Command:**

-WRITE

**Function:**

Writes information to non-FOCUS files.

Note that all files that have been written should be closed upon any exit from the procedure using -QUIT, -EXIT, or -RUN.

In TABLE, WRITE is a synonym for SUM; functionally it is quite different from -WRITE.

See -READ.

**Syntax:**

-WRITE *ddname* [NOCLOSE] *text*

where:

*ddname*

Is the logical name of the file as defined to FOCUS using FILEDEF (or for MVS, 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:**        -“ “

**Function:**        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.

See -CRTFORM.

**Syntax:**         - " "

where:

" "

Enclose textual information, fields and spot markers.

## System 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.

Some 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.
- Procedure files must have the record format (RECFM) F and the logical record length (LRECL) 80.

The following are some general rules that apply in regard to supplying values for variables:

- The maximum length of a variable value is 79 characters.
- A physical FOCSTACK line with all variables expanded to their full values cannot exceed 80 characters. Since most variables are part of a line in a procedure, it is recommended that you use values that are less than 80 characters long.
- 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, CITY='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.

The following is a list of operating system default values, limits, and format specifications.

- The default value for the operating system return code value is 8.
- The maximum number of amper variables available in a procedure is 512, of which approximately 30 are reserved for use by FOCUS. This includes all local, global, system, statistical, special, and index variables.
- Literals must be surrounded by single quotation marks if they contain embedded blanks or commas. To produce a literal that includes 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 255.
- Integer formats are described by the letter I followed by the number of digits to be entered. The number can be from one to nine 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. The label may precede or follow GOTO in the procedure.
- A date given to the Dialogue Manager cannot be more than 20 characters long, including spaces.
- -INCLUDE files can be nested up to 4 levels deep.
- 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 5

# Defining a Word Substitution

### Topics:

- The LET Command
- Variable Substitution
- Null Substitution
- Multiple-line Substitution
- Recursive Substitution
- Using 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

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 your requests easier to understand.



## 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 your 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. You can substitute any phrase that you enter online unless you are entering a MODIFY 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 and 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.

Once defined, 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 that 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. A semicolon is not required after the last definition on a line.

**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 unnumbered or numbered. 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 coded 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 in MVS:

```
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

```
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.

Special considerations regarding number signs apply in the CMS environment.

### 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 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. Enter:

```
LET ECHO
```

This turns on the LET ECHO facility. When you enter a FOCUS statement, LET ECHO displays the statement as interpreted by FOCUS.

#### Note:

- 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 at the top of a FOCEXEC. END ECHO gets coded on the last line of the FOCEXEC.

To turn off the LET ECHO facility and return to the FOCUS command level, enter:

```
ENDECHO
```

**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 request, 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

```
LET !4 = EX DAILYRPT  
LET !6 = END  
LET !20 = IF RECDLIMIT EQ 10  
LET !21 = .EX MYREPORT
```

---

## CHAPTER 6

# Enhancing Application Performance

### Topics:

- FOCUS Facilities
- Loading a File
- Compiling a MODIFY Request
- Accessing a FOCUS Data Source (MVS Only)

This topic covers FOCUS facilities that are available to you across command environment boundaries. These facilities are easy to use and, in many cases, step-by-step instructions are provided.

## 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 <i>Loading a File</i> on page 6-2).
COMPILE	Translates MODIFY requests into compiled code ready for execution (see <i>Compiling a MODIFY Request</i> on page 6-7).
MINIO	<b>Note:</b> This facility is for MVS only.  Improves performance by reducing I/O operations when accessing FOCUS data sources (see <i>Accessing a FOCUS Data Source (MVS Only)</i> on page 6-8).

## 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).
- Compiled MODIFY requests (FOCCOMP).
- MODIFY requests (MODIFY).

Using memory-resident files decreases execution time because the files do not have to be read from disk. Use the UNLOAD command to remove the files from memory.

### Syntax

#### How to Load a File

```
LOAD filetype filename1... [filename2...]
```

where:

*filetype*

Specifies the type of file to be loaded (MASTER, access file, FOCEXEC, FOCCOMP, or MODIFY).

*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 the 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, access file, FOCEXEC, MODIFY, or FOCCOMP). 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 the 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 them 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. This is because 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
DATACOM	FOCDTCM
UDB	FOCSQL
IDMS	FOCIDMS
IMS (IMS=NEW only)	ACCESS
MODEL 204	FOCM204
ORACLE	FOCSQL
SQLDS	FOCSQL
S2K	FOCS2K
SUPRA	ACCESS
TERADATA	FOCSQL
TOTAL	FOCTOTAL

## Loading a Compiled MODIFY Request

When you load a compiled MODIFY request, FOCUS loads the FOCCOMP file from disk into memory, then reads and parses the Master File and binds the description to the FOCCOMP file. You may then run the request by issuing the RUN command. The RUN command causes FOCUS to search for a memory-resident FOCCOMP file. If FOCUS cannot find the file, it searches for a disk version in the normal way.

Loading FOCCOMP files not only eliminates the I/Os required to read large FOCCOMP files and their associated Master Files, but also causes another, more subtle effect. When you issue the RUN command to execute a FOCCOMP file from disk, virtual storage must be paged in to accommodate it. If the FOCCOMP file is large, it may require many pages (and a large virtual storage area) in a very short time. If you load the FOCCOMP file first, the initial surge of paging occurs only once at LOAD time. After that, each execution of the loaded file requires a lower paging rate.

### Syntax

#### How to Execute a Compiled Request

*RUN request*

where:

*request*

Is the name of the compiled request stored in memory.



## Loading a MODIFY Request

The LOAD MODIFY command is similar to the COMPILE command (described in the *Maintaining Databases* manual) except that instead of writing the compiled output to a FOCCOMP file on disk, FOCUS writes the output into memory as a pre-loaded, compiled MODIFY. FOCUS then reads the Master File associated with the MODIFY command from disk and translates it into an internal table that is tightly bound with the compiled MODIFY. Thus the command

```
>LOAD MODIFY NEWTAX
```

has substantially the same effect as

```
>COMPILE NEWTAX  
>LOAD FOCCOMP NEWTAX
```

except that the compiled code is never written to disk.

After you enter a LOAD MODIFY command, the resulting compiled MODIFY is indistinguishable from code loaded with LOAD FOCCOMP. Thus the UNLOAD MODIFY and ? LOAD MODIFY commands produce exactly the same results as the UNLOAD FOCCOMP and ? LOAD FOCCOMP commands. Note that the UNLOAD FOCCOMP and UNLOAD MODIFY commands unload the bound Master File as well.

When you issue the RUN command to invoke a MODIFY procedure, FOCUS looks for a memory-resident compiled procedure (created by a LOAD FOCCOMP or LOAD MODIFY command) of that name. If the procedure cannot be found, FOCUS then searches for a disk version of the FOCCOMP file in the normal way.

The benefits of the LOAD MODIFY command are that disk space is not used to store the FOCCOMP file, disk I/Os are reduced, the FOCEXEC cannot get out of step with the compiled version, and the paging rate is reduced as it is with FOCCOMP files.

## 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 will be displayed. 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			
CAR	MASTER	4200	BYTES
EXPERSON	MASTER	4200	BYTES
CARTEST	FOCEXEC	8400	BYTES

## Compiling a MODIFY Request

The COMPILE command translates a MODIFY request stored in a FOCEXEC into an executable code module. This module, like an object code module, cannot be edited by a user. However, it loads faster than the original request because the MODIFY commands have already been interpreted by FOCUS (the initialization time of a compiled MODIFY module can be four to ten times faster than the original request). Compiling a request can save a significant amount of time if the request is large and must be executed repeatedly. You compile the request once, and execute the module as many times as you need it.

Enter the COMPILE command at the FOCUS command level (the FOCUS prompt). To module, use the RUN command from the FOCUS command level.

**Syntax**

**How to Compile MODIFY Request**

COMPILE *focexec* [AS *module*]

where:

*focexec*

Is the name of the FOCEXEC where the request is stored.

*module*

Is the name of the module. The default is the FOCEXEC name. FOCEXEC names and module names are system dependent.

**Syntax**

**How to Execute a Module**

RUN *module*

where:

*module*

Is the name of the module.

You will see no difference in execution between the module and the original request, but it will load much faster.

## Reference

### Considerations for Compiling a MODIFY Request

The following are considerations for compiling a MODIFY request:

- The FOCEXEC procedure to be compiled may only contain one MODIFY request. It may not contain any other FOCUS, Dialogue Manager, or operating system commands.
- Before compiling a request or executing a module, allocate all input and output files such as transaction files and log files. These allocations must be in effect at run time.
- Before compilation, issue any SET, USE, COMBINE, or JOIN commands necessary to run the request.
- If the data source you are modifying is joined to another file (using the JOIN command) during compilation, it must be joined to the file at run time.
- If you are modifying a combined structure (using the COMBINE command), the structure must be combined both at compilation and at run time.
- FOCEXECs prompt for Dialogue Manager variable values at compilation time. These values cannot be changed at run time.
- If you are using FOCUS security to prevent unauthorized users from executing the request, the password you set at compilation time must be the same one set at run time.

## Accessing a FOCUS Data Source (MVS Only)

MINIO is a new I/O buffering technique that improves performance by reducing I/O operations when accessing FOCUS data sources under MVS. With MINIO set on, no block is ever read more than once, and therefore 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.

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 fifty percent are achievable with MINIO. The actual reduction will vary 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 update physically disorganized data sources both 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

? 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

### Topics:

- 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

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.

## 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 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.

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 *Technical Publications Catalog*.

## 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 your application accesses them.

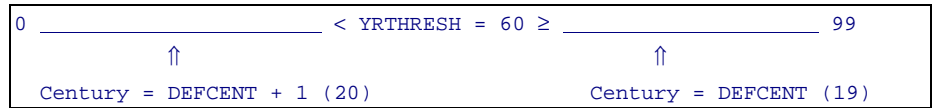
The sliding window technique recognizes that the earliest and latest values for a single date field in most business applications are within 100 years of one another. For example, a human resources application typically contains a field for the birth date of each active employee. The difference in the 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 your documentation on describing data.

## 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 or subroutine, supplied by Information Builders, that uses legacy dates, and the input date does not contain century digits.  
On input, the subroutine will use the window defined for an I6 legacy date field (with edit options). The output format may be I8 (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 an 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 MODIFY.

- 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.

## Reference

### Restrictions With MODIFY

The following results occur when you use the sliding window technique with a MODIFY request or FOCCOMP procedure:

- A MODIFY request compiled prior to Version 7.0 Release 6, when run with global SET DEFCENT and SET YRTHRESH settings, or with file-level or field-level settings, yields a FOC1886 error message. You must recompile the MODIFY request.
- A MODIFY request compiled in Version 7.0 Release 6, when run with global SET DEFCENT and SET YRTHRESH settings, or with file-level or field-level settings, yields a FOC1885 warning message.
- A FOCCOMP procedure, compiled with global SET DEFCENT and SET YRTHRESH settings, and run in releases prior to Version 7.0 Release 6, yields a FOC548 invalid version message. You must recompile the MODIFY request.
- A FOCCOMP procedure that contains DEFCENT/YRTHRESH or FDEFCENT/FYRTHRESH attributes in the associated Master File, and run in releases prior to Version 7.0 Release 6, yields a FOC306 description error message.

## 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 DEFCEMENT 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 your documentation on the SET command.

### Syntax

#### How to Define a Global Window With SET

To define a global window, issue two SET commands.

The first command is

```
SET DEFCEMENT = {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 FDEFCEMENT for the century. Two-digit years less than the threshold assume the value of FDEFCEMENT + 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 FDEFCEMENT 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 your documentation on creating reports.

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 their 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 your documentation on the SET command.

## 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 change 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 your documentation on describing data.

### Example

#### Defining a Dynamic Global Window With SET

In the following request, the COMPUTE command calls the subroutine 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 subroutine 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 subroutine 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 your documentation on creating reports.

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 their 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, their 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 [ALL]
```

where:

**ALL**

Returns values for every possible environment setting. Excluding it generates a shorter list of the most common settings.

### Example

#### Querying the Current Global Value of DEFCENT and YRTHRESH

Enter

```
? SET
```

to query the current global value of DEFCENT and YRTHRESH.

The following is a response to the query:

PARAMETER SETTINGS					
ALL	OFF	HDAY	.	PRINT	ONLINE
.	.	.	.	.	.
.	.	.	.	.	.
DEFCENT	20	PAGE-NUM	ON	TRACKIO	ON
.	.	.	.	.	.
.	.	.	.	.	.
FOCSTACK SIZE	8	PREFIX	.	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 your documentation on describing data.

## 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 DEFCECENT for the century. Two-digit years less than the threshold assume the value of DEFCECENT + 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 DEFCECENT 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 DEFCECENT/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 your database maintenance documentation.

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 your documentation on creating reports.

```

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 (FDFCENT + 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 DEFCECENT and YRTHRESH attributes to the Master File on the field declaration.

The syntax for the first attribute is

```
{DEFCECENT|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 DEFCECENT for the century. Two-digit years less than the threshold assume the value of DEFCECENT + 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 DEFCECENT 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 DEFCECENT 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 your database maintenance documentation.

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
  92022900022920000229000229200002290002000100020006000200002292000
  88022900022920000229000229200002290002000100020006000200002292000
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
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 the last example in *Defining a Global Window With SET* on page 7-8 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; their 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 subroutine call, the subroutine 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 your documentation on creating reports. 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 DEFCEM 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 DEFCEM for the century. Two-digit years less than the threshold assume the value of DEFCEM + 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 DEFCEM 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 subroutine 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 Subroutine Input in a DEFINE Command**

The following sample request illustrates a call to the subroutine 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 your documentation on creating reports.

```
DEFINE FILE EMPLOYEE
SIXTY_DAYS/I8YYMD = AYMD(HIRE_DATE, 60, 'I8YYMD');
END

TABLE FILE EMPLOYEE
PRINT HIRE_DATE SIXTY_DAYS
END
```

The subroutine 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 subroutine 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. 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 subroutine 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, their 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.

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 your database maintenance documentation.

The DEFCENT and YRTHRESH parameters must immediately follow the field format specification; their 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 subroutine call, the subroutine 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 Calculated Value in a Report

Use standard COMPUTE syntax, as described in your documentation on creating reports. 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 DEFCEMENT 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 DEFCEMENT for the century. Two-digit years less than the threshold assume the value of DEFCEMENT + 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 DEFCEMENT 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 subroutine 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 your database maintenance documentation; partial syntax is shown here.

On the line that specifies the name of the calculated value, include the DEFCEMENT and YRTHRESH parameters and values. The parameters must immediately follow the field format information.

```
MODIFY FILE filename
```

```
.  
.
.
```

```
COMPUTE
```

```
  fieldname[/format] [{DEFCEMENT|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 subroutine 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 DEFCEM and YRTHRESH on the COMPUTE command define a window from 1999 to 2098. Notice that both DEFCEM 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 (DEFCEM + 1).

The output is:

```
PAGE      1

      SALARY  LATEST_DAT_INC
      -----  -
$332,929.00  2082/08/01
```

**Example**

**Defining a Window for Subroutine Input in a COMPUTE Command**

The following sample request illustrates a call to the subroutine 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 your documentation on creating reports.

In the request, the input field is HIRE\_DATE. The subroutine 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 subroutine uses the FDFCENT 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 subroutine 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 your documentation on the SET command.
- You can display the current date with a 4-digit year using the Dialogue Manager system variables &YYMD, &MDYY, and &DMYY. The system variable &DATEfmt 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 your documentation on Dialogue Manager.

## 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 your documentation on the SET command.

## Date Functions and Subroutines

The date functions and subroutines supplied with your software work across centuries. Many of them facilitate date manipulation. For details on date functions and subroutines, see your documentation on creating reports.

## **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 your documentation on database maintenance.

## **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 your documentation on describing data.

## **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

### Topics:

- Integrating the Euro Currency
- Converting Currencies
- Preparing FOCUS to Process Currency Conversions
- Activating the Currency Data Source
- Querying the Currency Data Source in Effect
- Processing Currency Data

This topic describes how to create and use a currency data source to convert to and from the new euro currency.

## Integrating the Euro Currency

With the introduction of the euro currency, businesses need to maintain books in two currencies, add new fields to their data source designs, and perform new types of currency conversions. FOCUS can perform currency conversions according to the rules specified by the European Union. Before you can use FOCUS to process currency conversions, you must:

- Create a currency data source with the currency IDs and exchange rates you will use. See *Creating the Currency Data Source* on page 8-4.
- Identify fields in your data sources that represent currency data. See *Identifying Fields That Contain Currency Data* on page 8-6.
- Activate your currency data source. See *Activating the Currency Data Source* on page 8-8.

After you complete these preliminary steps, you can perform currency conversions. See *Processing Currency Data* on page 8-10.

**Note:** Operating system vendors are in the process of integrating the euro currency symbol into their environments. As the euro symbol becomes available, FOCUS will support it.

## Converting Currencies

Although the euro was introduced in 11 countries of the European Union on January 1, 1999, it will not immediately replace local currencies in those countries. During the transition period from 1999 to 2002, both traditional currencies and the euro will be used simultaneously for accounting purposes and non-cash transactions in each participating country. Euro cash will not be introduced until January 1, 2002, and by July 1, 2002 the traditional currencies will no longer be legal tender.

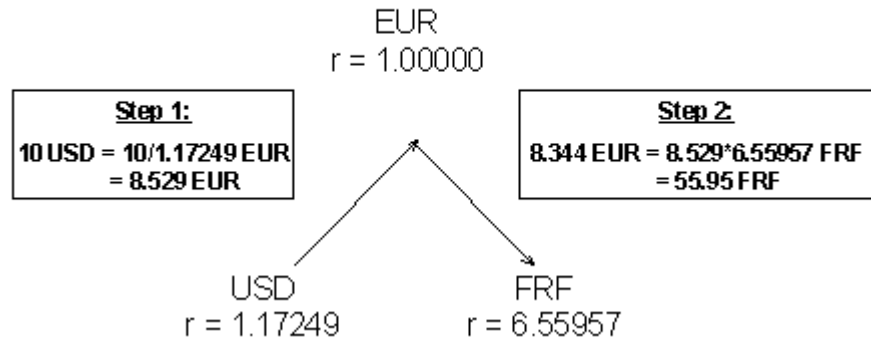
On the last day of 1998, the European Union set fixed exchange rates between the euro and the traditional national currency in each of the 11 adopting member nations. While the exchange rates within Euroland will remain fixed, exchange rates between the euro and non-euro countries will continue to vary freely and, in fact, several rates may be in use at one time (for example, actual and budgeted rates).



The European Union has established the following rules for currency conversion:

- The exchange rate must be specified as a decimal value,  $r$ , with six significant digits (not six decimal places). For example, 123.456 has six significant digits but not six decimal places. 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 at least three decimal places (FOCUS rounds to exactly three decimal places). Then convert from the euro to the second national unit, rounding the result to two decimal places. The following diagram illustrates this two-step conversion process known as *triangulation*:

Converting 10 US Dollars to French Francs



## Preparing FOCUS to Process Currency Conversions

Although 11 or more currencies in the European Union will be converting to the euro, more than 100 currencies have a recognized status worldwide. In addition, you may need to define custom currencies for some applications.

You identify your currency codes and rates by creating a currency data source. The currency data source can be any type of data source that FOCUS can access.

### Creating the Currency Data Source

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 U.S. dollars or BEF for Belgian francs. (For a partial list of recognized currency codes, see *Sample Currency Codes* on page 8-13.)
- 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; the currencies you can define are not limited to official currencies and, therefore, the currency data source can be fully customized for your applications.

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 FOCUS loads the currency data source into memory before you perform currency conversions.

## Syntax

### How to Specify Currency Codes and Rates in a Master File

The currency data source can be any type of data source accessible by FOCUS (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.

The syntax is

```
FIELD = CURRENCY_ID,   FORMAT = A3,           ACTUAL = A3 , $
FIELD = rate1,         FORMAT = {D12.6|numeric_format1}, ACTUAL = A12, $
.
.
.
FIELD = raten,         FORMAT = {D12.6|numeric_formatn}, ACTUAL = A12, $
```

where:

#### 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:** FOCUS automatically recognizes the code EUR; you should *not* store this code in your currency data source. See *Sample Currency Codes* on page 8-13 for a list of common currency codes.

#### rate1,...,raten

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 An

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).

## **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,    1.17249,    1.20000,$
BEF,    40.3399,    41.00000,$
```

## **Identifying Fields That Contain Currency Data**

Once 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} , $
```

- 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:

*filename*

Is the name of the file for which this field is defined.

*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.

*expression*

Is a valid expression.

### Example

#### Identifying a Currency-Denominated Field

Assume that the currency data source contains the currency ID value BEF (Belgian francs).

If the FINANCE data source contains a field named PRICE that is denominated in Belgian francs, the description of PRICE in the FINANCE Master File could be:

```
FIELD = PRICE, ALIAS=, FORMAT = P17.2, CURR=BEF,$
```

## Activating the Currency Data Source

Before you can perform currency conversions, you must bring the relevant currency data source into memory by issuing the SET EUROFILE command.

### Syntax

#### How to Activate Your Currency Data Source

Issue the following command at the FOCUS command prompt, in a FOCEXEC, or in any supported profile:

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

where:

*ddname*

Is the name of the Master File for the currency data source. There is no default value for EUROFILE. The ddname must refer to a data source known to FOCUS and accessible by FOCUS in read-only mode.

OFF

Deactivates the currency data source and removes it from memory.

During your FOCUS session, if you want to access a different currency data source, you can re-issue the SET EUROFILE command.

**Note:**

- You cannot append any additional SET parameters to the SET EUROFILE command line. For example, the PAUSE setting would be lost if you issued the following command:

```
SET EUROFILE=filename , PAUSE=OFF
```

- You cannot issue the SET EUROFILE command within a TABLE request.

## 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 the ? SET ALL query command or the ? EUROFILE query command.

### Syntax

#### How to Determine the Currency Data Source in Effect

If you want to determine which currency data source is in effect, issue the ? SET ALL command or the new EUROFILE query command:

```
? SET EUROFILE
```

### Example

#### Determining the Currency Data Source in Effect

Issuing the command

```
? EUROFILE
```

produces information similar to the following:

```
EUROFILE          GBP
```

### Reference

#### SET 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
```

**Note for Pooled Table users:** The SET EUROFILE command creates a pool boundary.

## 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 FOCUS. You do not need to add the EUR currency ID to your currency data source.

### Syntax

### How to Convert Currency Data

Use the following syntax for calling a currency conversion function.

- In a TABLE, GRAPH, or MODIFY 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*

Must be 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. The result will always be rounded to two decimal places, which will display if the format allows at least two decimal places.



*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 its currency's *rate1* value 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's *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's *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.

**Note:** Maintain does not support these currency conversion functions.

## Example      Converting Currencies

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);$
```

## Reference      Currency Calculation Processing and Messages

The result is always 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.

Issuing a TABLE 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
```

## Reference      Sample Currency Codes

The following rates were in effect on December 31, 1998. Euroland countries as of that date are marked with an asterisk (\*). Their rates are fixed and will not change; the rates for other countries can change over time:

Country	Currency Code	Rate
Austria*	ATS	13.7603
Belgium*	BEF	40.3399
Canada	CAD	1.7978
Denmark	DKK	7.46215
European Union	EUR	1
Finland*	FIM	5.94573
France*	FRF	6.55957
Germany*	DEM	1.95583
Greece	GRD	328.6
Ireland*	IEP	0.787564
Italy*	ITL	1936.27
Japan	JPY	133.149
Luxembourg*	LUF	40.3399
Netherlands*	NLG	2.20371
Norway	NOK	8.91039
Portugal*	PTE	200.482
Spain*	ESP	166.386
Sweden	SEK	9.52669
Switzerland	CHF	1.61093
UK	GBP	0.706739
USA	USD	1.17249

## Example      Converting U.S. Dollars to Euros, French Francs, and Belgian Francs

Assume PRICE is denominated in U.S. dollars and ACTUAL is the name of a rate in the currency data source. Using the currency conversion rates from *Sample Currency Codes* on page 8-13, the following FOCEXEC converts PRICE to euros, French francs, and Belgian francs:

```

-* CURRCODE IS THE CURRENCY DATA SOURCE
-* CURRDATA IS THE DATA SOURCE WITH CURRENCY-DENOMINATED FIELDS

-* THE FOLLOWING FILEDEFS ARE FOR RUNNING UNDER CMS
CMS FILEDEF CURRCODE DISK CURRCODE TEXT A
CMS FILEDEF CURRDATA DISK CURRDATA TEXT A

-* THE FOLLOWING ALLOCATIONS ARE FOR RUNNING UNDER MVS
-* DYNAM ALLOC FILE CURRCODE DA USER1.FOCEXEC.DATA(CURRCODE) SHR REU
-* DYNAM ALLOC FILE CURRDATA DA USER1.FOCEXEC.DATA(CURRDATA) SHR REU

SET EUROFILE = CURRCODE

DEFINE FILE CURRDATA
PRICEEUR/P17.2 CURR EUR = EUR(PRICE, ACTUAL);
END

TABLE FILE CURRDATA
PRINT PRICE PRICEEUR AND COMPUTE
PRICEFRF/P17.2 CURR FRF = FRF(PRICE, ACTUAL);
PRICEBEF/P17.2 CURR BEF = BEF(PRICE, ACTUAL);
END

```

This request generates the following report:

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

**Note:** 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. Therefore, PRICEEUR yields the following inaccurate result (see PRICEBEF above) and is not valid as the intermediate value in a currency conversion that requires triangulation:

```
COMPUTE PRICENEW/P17.2 CURR BEF = BEF(PRICEEUR, ACTUAL);
```

```
PRICENEW
```

```
-----
```

```
171.85
```

```
206.54
```

```
1376.40
```

```
344.10
```

---

## CHAPTER 9

# Designing Windows With Window Painter

### Topics:

- Introduction
- Window Files and Windows
- Integrating Windows and the FOCEXEC
- Tutorial: A Menu-Driven Application
- Window Painter Screens
- Transferring Window Files

This topic describes how to create FOCUS menus and windows that work with FOCEXECs.

## 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. You can implement horizontal menus and multi-input windows as part of your FOCUS application. Horizontal menus can also have pulldown 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 then 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 them to fit your application. You can design user-friendly menus and can display convenient and eye-catching instructions onscreen.

FOCUS Window Painter itself guides you step by step, using windows like those you will be creating.

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 you can write 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 MVS or CMS.

## 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 will 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 Chapter 4, *Managing Applications With Dialogue Manager*. For details of integrating a FOCEXEC with the Window facility using return and goto values see *Integrating Windows and the FOCEXEC* on page 9-21.



## 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.

- In CMS, window files have file type FMU, and are created and updated on the A disk automatically by Window Painter.
- In MVS, 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 will temporarily allocate 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

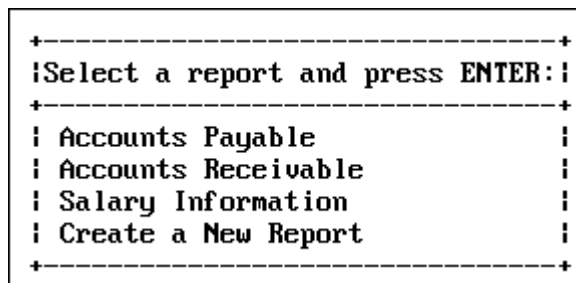
Window Painter creates 10 different types of windows, each with its own special uses:

- Vertical menus
- Horizontal menus
- Text input windows
- Text display windows
- File names windows
- Field names windows
- File contents windows
- Return value display windows
- Execution windows
- Multi-input windows

These windows are described in the following topics.

### Vertical Menu

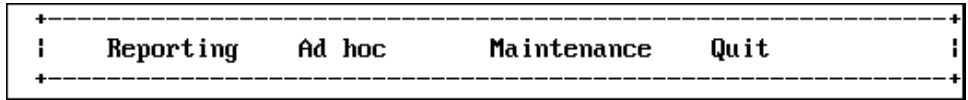
This is a *vertical* menu:



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 9-61.

## Horizontal Menu

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 pulldown 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.

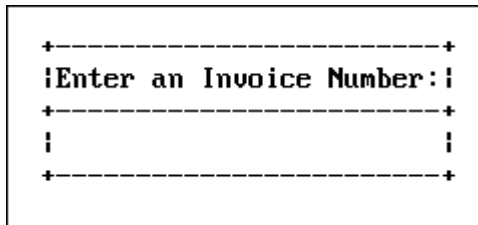
An application can display an associated pulldown menu for an item on a horizontal menu when the cursor is on that item. Choose the pulldown option from the Window Options menu as discussed in *Creating Windows* on page 9-14. An option to display descriptive text above or below the horizontal menu is 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 9-57.

## Text Display Windows

This is a *text display* window:

```

+-----+
|Instructions for printing:|
+-----+
|
| Press ALT-7 if you wish |
| to generate an OFFLINE |
| report.                 |
|
| Press ALT-8 if you wish |
| to generate an ONLINE  |
| report.                 |
|
+-----+

```

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:

```

+-----+
|Select the report you wish to generate and press ENTER:|
+-----+
| (MORE)-----|
| SALARY FOCEXEC B1 |
| ACCTS  FOCEXEC B1 |
| BILLS  FOCEXEC B1 |
| BUDGET FOCEXEC B1 |
| (MORE)-----|
+-----+

```

A file names window presents a list of names of up to 409 files (in CMS) or 1023 PDS members (in MVS). 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. Narrower windows can display a greater number of names.

## Field Names Windows

This is a *field names* window:

```
+-----+
|Select the field you wish to sort on and press ENTER:|
+-----+
| EMP_ID |
| LAST_NAME |
| FIRST_NAME |
| HIRE_DATE |
| DEPARTMENT |
| CURR_SAL |
| (MORE) |
+-----+
```

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 will be 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:

```

+-----+
|Select the record you want to display and press ENTER:|
+-----+
| STAMFORD      S  14B      |
| NEW YORK      U  14Z      |
| UNIONDALE    R  77F      |
| NEWARK        U   K1      |
+-----+

```

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 9-61.

- In CMS, the contents of any file (except as noted below) can be displayed. You will be prompted for the file name and file type.
- In MVS, the contents of any member of a PDS (except as noted below) can be displayed. Sequential files can also be displayed in TSO. You will be 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:

```

+-----+
|This is a sample Return Value Display window. |
+-----+
| TABLE FILE EMPLOYEE                          |
| PRINT EMP_ID FIRST_NAME LAST_NAME            |
| END                                           |
+-----+

```

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 9-61.

Only one return value display window may be displayed at a time on the screen. It will collect 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 9-61.

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 9-61.

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:

```

+-----+
| -* This is a sample Execution window. |
| TABLE FILE EMPLOYEE                 |
| PRINT EMP_ID BY LAST_NAME           |
| END                                   |
| -RUN                                  |
+-----+

Wind: EXECWIND Typ: Execution   PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add

```

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 enabling 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.

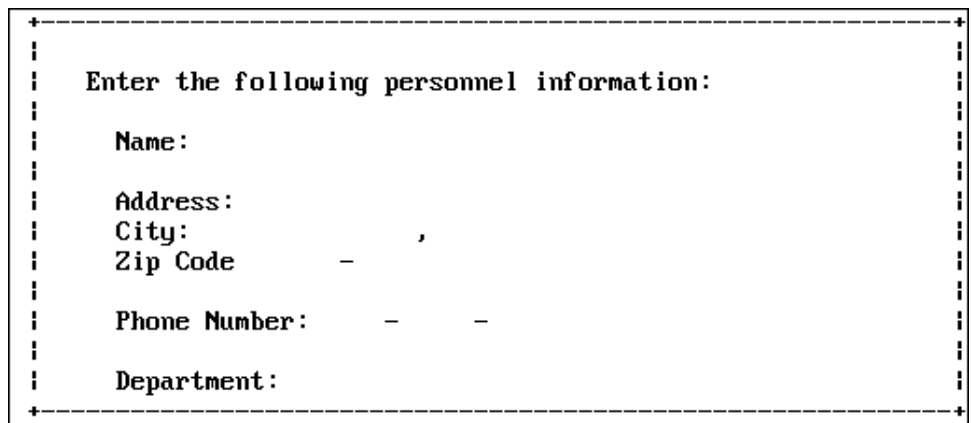


- Execution windows have no return values.
- Execution windows can contain up to 22 lines.
- Execution windows can use local variables.
- Goto values for execution windows should be assigned at line 1.
- Windows called from within execution windows preempt window goto values. For example, a -WINDOW command 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 will follow the logic determined by the windows themselves. This will substantially affect the application's transfer of control.
- Use -RUN for immediate execution; otherwise requests will be 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:



```
Enter the following personnel information:

Name:

Address:
City:
Zip Code      -
Phone Number:  -
Department:
```

A multi-input window prompts you for information that will be 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 9-61.

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 9-29 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 pulldown 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:

```
+-----+
| 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     |
|                +-----+
+-----+
```

You will be prompted to enter a name and brief description for the window, after which you will 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 will 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 the Enter key after each item; the item automatically appears on the horizontal menu.

The following is an example of a completed creation screen:

```

+-----+
| Vertical  Inputs   Lists   Execution Misc   End   |
+-----+

+-----+
| Vertical  |
| Inputs   |
| Lists    |
| Execution|
| Misc     |
| End      |
+-----+

Wind: HORO      Typ: Menu (horz)  PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add

```

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:

```

+-----+
|Exit this menu |
|Goto value     |
|Return value   |
|FOCEXEC name   | c      Maintenance  Quit  |
|Heading        |
|Description    |
|Show a window  |
|Unshow a window|
|Display list   |
|Hide list      |
|Popup          | (Off)|
|Help window    |
|Line break     |
|Multi select   | (Off)|
|Quit           | PF3 |
|Menu text      |
|Text line      | (x+1 |
|Pulldown       | (Off)|
|Conceal option |
|Switch window  |
+-----+

```

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:

```
          VERTICAL MENU TESTS
+-----+
| Vertical  Inputs   Lists   Execution Misc   End   |
+-----+
```

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 Pulldown option for a horizontal menu. With this option, you can assign a pulldown menu to be displayed for a horizontal menu item whenever the cursor is positioned on that item.

### Pulldown Menus

When you set the Pulldown option ON, you can display an associated pulldown 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 Pulldown option and press Enter. Note that when Pulldown is set ON, Menu Text is automatically set OFF.

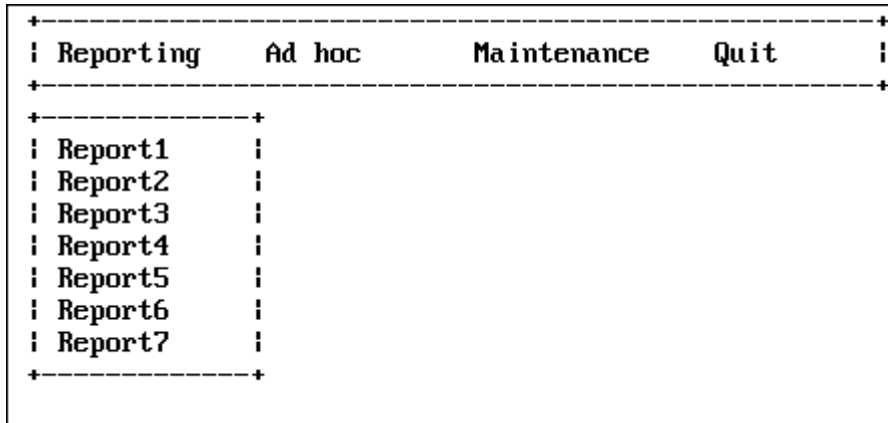
The associated pulldown menu must be a vertical menu. When creating the horizontal menu, you must assign a Goto value to point to the pulldown menu. To do so, position the cursor on the goto value, press Enter, and enter the name of the pulldown menu you want to display in the space provided:

```

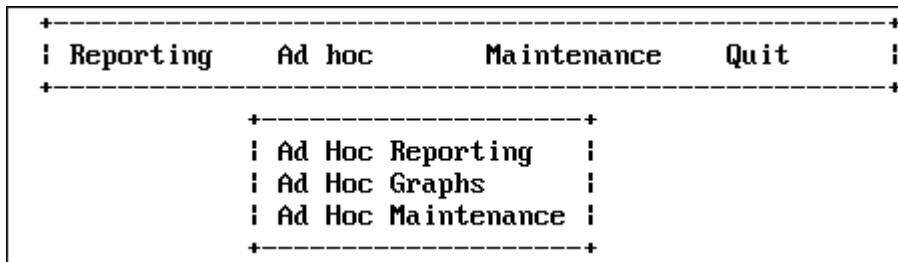
+-----+
| Reporting  +-----+
+-----+  |Enter name of next window to go to.|
          |Just 'Enter' for exit.                |
+-----+  +-----+
| Reporting  |  |rpts  |
| Ad hoc    |  +-----+
| Maintenance|
|   Quit    |
+-----+
```

You must create the vertical menu, rpts, as you would any other vertical menu. See *Tutorial: A Menu-Driven Application* on page 9-29 for examples.

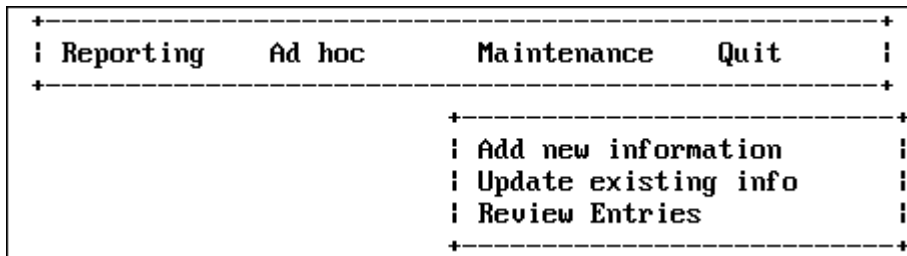
The following example shows a horizontal menu with the Reporting pulldown menu displayed:



The following screen shows the same menu with the Ad hoc pulldown menu displayed:



The following screen shows the same menu with the Maintenance pulldown 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 will then be prompted for a name, description and heading. Place the window on the screen and size it as desired.

```
+-----+
| 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     |
|               +-----+
|
```

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.

```

          F O C U S   W I N D O W   P A I N T E R
+-----+
| INSTRUCTIONS :  Move cursor to selection and hit ENTER |
|                Use PF3 or PF12 to undo a selection  |
|                Use PF1 for help                      |
+-----+
|
|                +-----+
|                |Enter a description:                 |
|                +-----+
|                |Sample file for Window Painter tutorial. |
|                +-----+
|

```

This is what the developer's screen looks like after several fields have been included in the multi-input window:

```

+-----+
|Enter the following personnel information:
+-----+
| Name: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
|
| Address: XXXXXXXXXXXXXXXXXXXXXXXX
|          XXXXXXXXXXXX , XX
| Zip Code:XXXXXX - XXXX
|
| Phone Number: XXX - XXX - XXXX
|
| Department: XXXXXXXXXXXXXXXX
+-----+

```

**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 you create with Window Painter are designed for you to use within an application FOCEXEC. This topic discusses how to integrate your windows into your FOCEXEC.

## Syntax

### The -WINDOW Command

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. In CMS, this is a file name. The file must have a file type of FMU or TRF. In MVS, this is a member name. The member must belong to a PDS allocated to ddname FMU.

*windowname*

Optional. Identifies which window in the file to display first. Can be set in Window Painter or in first window displayed.

*PFKEY*/*NOPFKEY*

Enables (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 will be 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 9-26.)

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 wish to 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 9-26).
- 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 wish to 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 whatever window you wish.
- If you wish 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, you may do so by returning control to the FOCEXEC and executing another -WINDOW command with the BLANK option in effect.

To back up a step during window execution, the user may press the PF12 or PF24 keys. This will 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 9-26.

## 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 almost anything you desire: 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 you wish 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 needing to select 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 itself 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**

For example, 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 you 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 you are creating your windows, you will 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 9-22, 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 will return to the previous window upon completion of the current window (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 their 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` will contain 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

If you wish 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 Chapter 1, *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 Chapter 2, *Querying Your Environment*.

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 will immediately return 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 will not be 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 file type FMU, and appear within single quotation marks.

*windowname*

Identifies the first window to be executed. If a window name is not specified, FOCUS will execute the default start window, or the first window created.

PFKEY/NOPFKEY

Tells FOCUS you will (will not) be testing 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 will be shown an automatically generated list of FOCEXECs from which to pick. A user who chooses to create a new request will be 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 will be 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:
<b>MAIN</b>	Report? Graph? Exit?	window EXECTYPE window EXECTYPE back to FOCEXEC
<b>EXECTYPE</b>	Existing Request? New Request?	window EXECNAME back to FOCEXEC
<b>EXECNAME</b>	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 will drive this application.

Begin by using the TED editor to create the FOCEXEC file SAMPLE. At the FOCUS prompt, type

```
TED SAMPLE
```

and press Enter. (In CMS, TED assigns the file type FOCEXEC unless you specify another file type. In MVS, you must specify ddname as follows:

```
FOCEXEC (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 they must begin in the first column. Dialogue Manager is discussed in Chapter 4, *Managing Applications With Dialogue Manager*.

You will 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 will use. (We will create it in this tutorial.) `MAIN` is the window where the procedure will begin.

Control will 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 `FOCEXEC` will branch to `-GENERATE`; if `XXIT`, to `-EXIT`. Each return value corresponds to a selection on the menu window `MAIN`.
3. This label begins the `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 will execute 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 (in CMS, for example, a file name and a file type)—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 command 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:

<b>BORDER</b>	A text display window used as a background display for the other windows.
<b>BANNER</b>	A text display window that introduces the application.
<b>MAIN</b>	A vertical menu from which the user can choose to create a graph or a report, or exit the application.
<b>EXECTYPE</b>	A vertical menu from which the user chooses to execute an existing procedure or create a new one.
<b>EXECNAME</b>	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 will be included in the window file named SAMPLE. You are going to start by building that window file.

- In CMS, when you use Window Painter to create a window file, the file is automatically created by the system on your A disk.
- In MVS, 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:

```

+-----+
| Reporting   Ad hoc       Maintenance   Quit       |
+-----+
                                     +-----+
                                     | Add new information |
                                     | Update existing info |
                                     | Review Entries      |
                                     +-----+

```

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 will look 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                       |
+-----+
|
|               +-----+
|               |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     |
|               +-----+
|
```

You will see a screen asking for a description of the window file.

Type

[Sample file for Window Painter tutorial](#)

and press Enter.

```

                                F O C U S   W I N D O W   P A I N T E R
+-----+
| INSTRUCTIONS :  Move cursor to selection and hit ENTER      |
|                Use PF3 or PF12 to undo a selection         |
|                Use PF1 for help                             |
+-----+
|
|                +-----+
|                |Enter a description:                        |
|                +-----+
|                |Sample file for Window Painter tutorial.   |
|                +-----+
|

```

## Creating the Text Display Window Named BORDER

Now you are ready to create the first window. The screen that appears on your display is the Window Painter Main Menu. Select

[Create a new window](#)

and press Enter.

```

                                F O C U S   W I N D O W   P A I N T E R
+-----+
| INSTRUCTIONS :  Move cursor to selection and hit ENTER      |
|                Use PF3 or PF12 to undo a selection         |
|                Use PF1 for help                             |
+-----+
|
|                +-----+
|                |Select one of the following:              |
|                +-----+
|                |Create a new window                       |
|                |Edit an existing window                   |
|                |Delete an existing window                 |
|                |Run the window file                       |
|                |Switch window files                       |
|                |Utilities                                 |
|                |End                                       |
|                |Quit without saving changes               |
|                +-----+
|

```



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 will create for the application. BORDER will supply 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.

```
File: SAMPLE          F O C U S   W I N D O W   P A I N T E R
+-----+
| 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.

```

File: SAMPLE          F O C U S   W I N D O W   P A I N T E R
+-----+
| INSTRUCTIONS :  Move cursor to selection and hit ENTER |
|                Use PF3 or PF12 to undo a selection   |
|                Use PF1 for help                       |
+-----+
                +-----+
                |Enter a window description:           |
                +-----+
                |This window borders all my screens.    |
                +-----+

```

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. You will now 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 will be 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.

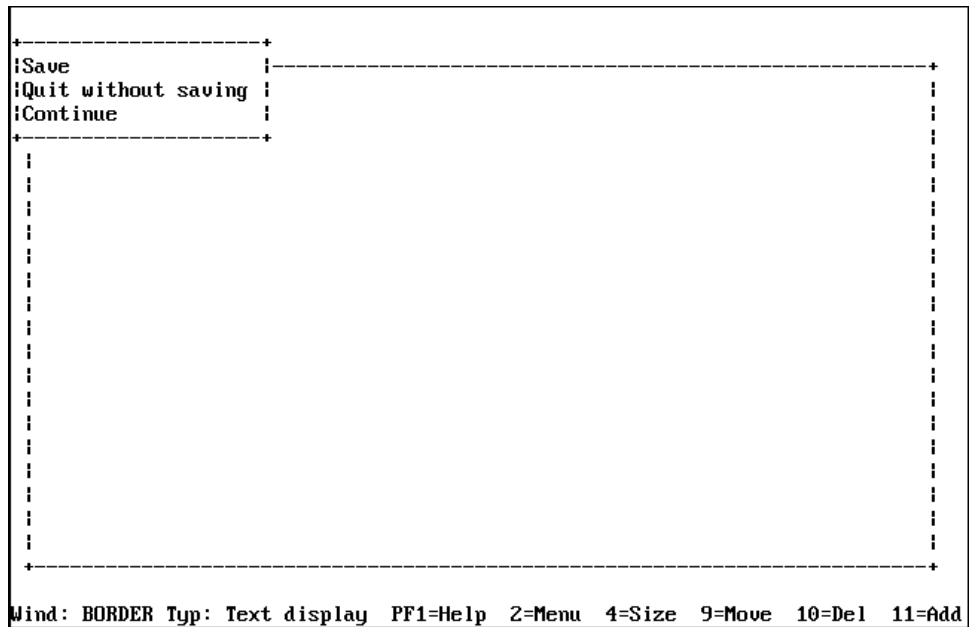
This window's border will form the background border for the other windows in this application.

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:

```
File: SAMPLE          F O C U S   W I N D O W   P A I N T E R
+-----+
|
|           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, you should save it. Press PF3.



Press Enter to select Save. You will be returned 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.

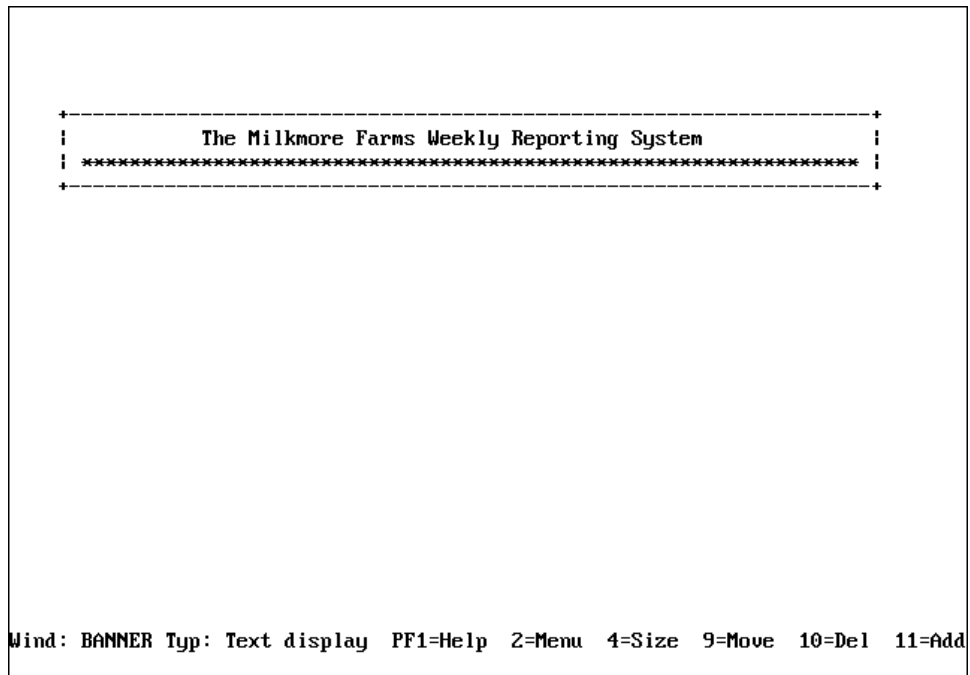
You will now enter text to be displayed in the window. Reposition the cursor on the first line within the window, ten spaces to the right of the window's left border, and type:

The Milkmore Farms Weekly Reporting System

Type a line of asterisks (\*) all the way across the window's second line. (Begin at the second column within the window, because the first column of every window is protected.)

You will now 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

Now you will 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 ten 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 will be created wide enough to contain the heading. Now position the cursor six rows below the window's bottom edge, and ten columns to the right of its right edge. Press PF4 and the window will be resized.

Type the following menu options as they appear below:

```

+-----+
|           Would you like to:           |
+-----+
| Create a report?                       |
| Create a graph?                       |
| Exit?                                  |
+-----+

Wind: MAIN Type: Menu (vert) PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add
```

Now you will 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.

```

+-----+
|Exit this menu  |
|Goto value    PF5|
|Return value  PF6|
|FOCEXEC name   |
|Heading       |
|Description    | +-----+
|Show a window  | |           Would you like to:           |
|Unshow a window| +-----+
|Display list   | | Create a report?                       |
|Hide list      | |                                     |
|Popup         (Off)| | Create a graph?                       |
|Help window    | |                                     |
|Line break     | | Exit?                               |
|Multi select (Off)| |                                     |
|Quit          PF3| +-----+
|ACE security rule|
|Switch window  |
+-----+

Wind: MAIN      Typ: Menu (vert)  PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add

```

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 will be prompted to either execute an existing report or create a new one. The window that displays that prompt will be called EXECTYPE, so the goto value of the first two menu options will be EXECTYPE.

Move the cursor to

`Goto value`

and press Enter.



In the space provided, type

EXECTYPE

and press Enter.

```
+-----+
|Enter name of next window to go to.| -----+
|Just 'Enter' for exit.           | you like to:  |
+-----+-----+
|EXECTYPE |           | Create a report? |
+-----+-----+           |
|           |           | Create a graph?  |
|           |           | Exit?             |
|           |           | -----+
|           |           |
+-----+-----+

Wind: MAIN      Typ: Menu (vert)  PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add
```

The return value collected by this window—&MAIN—will be 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.

```

+-----+-----+
|Enter return value for the line:| ld you like to: |
+-----+-----+
|RPT          | reate a report? |
+-----+-----+
|              | Create a graph?  |
|              |                  |
|              | Exit?           |
|              |                  |
+-----+-----+

Wind: MAIN      Typ: Menu (vert)  PF1=Help  Z=Menu  4=Size  9=Move 10=Del 11=Add

```

Exit the Window Options Menu by moving the cursor to

Exit this menu

and pressing Enter.

Now you will 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 will not assign a goto value. Since it exits to the FOCEXEC, there is no next 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 them to the list.

```
+-----+ +-----+
|Display list:| |Select one of these options:|
+-----+ +-----+
|             | |Add to the list      |
|             | |Delete from the list|
|             | |Quit                |
|             | +-----+
|             | |           Would you like to:           |
|             | +-----+
|             | | Create a report?                       |
|             | | Create a graph?                       |
|             | | Exit?                                  |
|             | +-----+
```

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 will be 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 will be 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

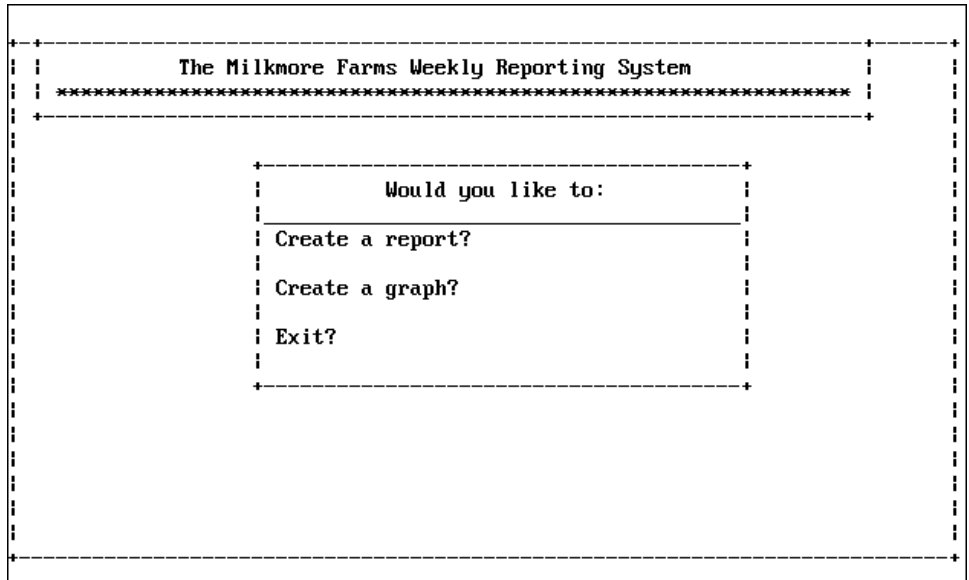
`Run the window file`

and press Enter.

Select

`MAIN`

as the starting screen. Press Enter, and you will see a screen like this:



Position the cursor on the “Create a report” line. When you press Enter to continue the display, you will see an error message because `EXECTYPE`—the goto value—has not been created yet. Ignore it, and press Enter to continue. You will see a screen displaying amper variables for this window and their 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 will create will also be 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.

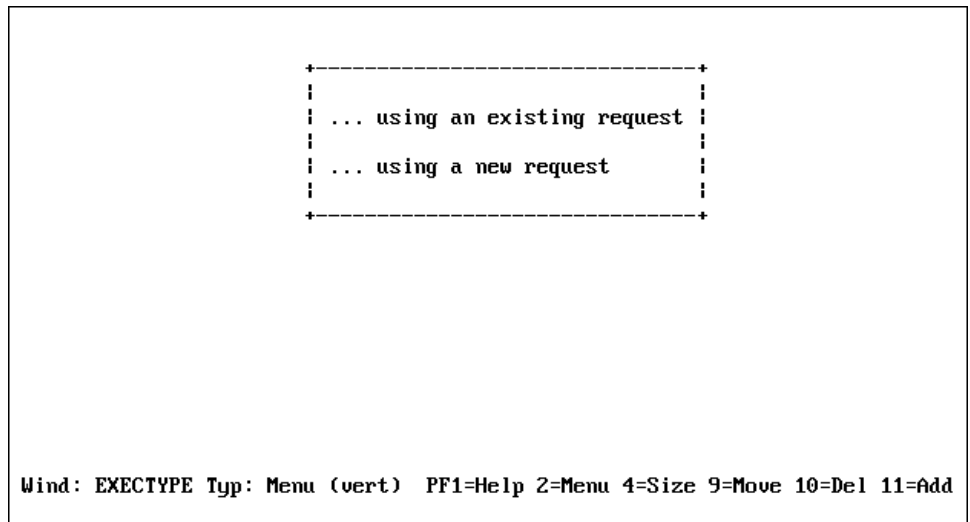
When prompted for a description, type

`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 GOTO value, a Return value and a FOCEXEC name. When prompted for a GOTO value press Enter again and you will be prompted for the Return value. Enter EXIST and press PF5 again and you are prompted for FOCEXEC name. Just press Enter.

If you select

`... using an existing request.`

from the EXECTYPE menu, the file names window EXECNAME will be displayed next. EXECNAME will contain a list of existing FOCEXEC files from which you may choose.

Move the cursor to the second menu item.

Now you need to consider the return and goto values for this option.

If you choose to create a new report or graph request, EXECNAME will not be displayed. Rather, control must pass back to the FOCEXEC, which will execute 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
```

Since you want control to pass to the FOCEXEC if this option is chosen, you will 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 it 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. You will see that (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 will be prompted for file-identification criteria. Type

\* FOCEXEC

and press Enter.

```
File: SAMPLE          F O C U S   W I N D O W   P A I N T E R
+-----+
| 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)|
|for & variable name containing the criteria: |
+-----+
|* FOCEXEC                                               |
+-----+
```

- In CMS, when the application is executed, this will select all files having the file type FOCEXEC.
- In MVS, when the application is executed, this will select 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 ten 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 will automatically be 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 will be plugged into the line

```
EX &EXECNAME
```

and the report or graph request will be executed.

But in order for this to happen, you must return control to the FOCEXEC. Therefore, you will assign no goto value to this window.

If you want 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 their actual return values set when the window is created or edited, the “return value” which is displayed and 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 will execute 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.

In CMS, this is a file name. The file must have a file type of FMU.

In MVS, this is a member name. The member must belong to ddname FMU.

If you do not specify file name, you will begin your Window Painter session at the Entry Menu, where you can choose a window file to use or can create a new window file. If you do specify file name, you will 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 will be asked if you want to create a new file. If not, the Window Painter Entry Menu will be displayed.

## Entry Menu

You can reach the Window Painter Entry Menu by typing

```
WINDOW [PAINT]
```

at the FOCUS prompt, and then pressing Enter.

The Entry Menu is the first screen you see:

```

File: SAMPLE          F O C U S   W I N D O W   P A I N T E R

+-----+
| 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                           |
|TEST      This is a test.                              |
|SAMPLE    Sample file for Window Painter tutorial.    |
+-----+

```

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 FOCXECs that include branching between window files. Refer to *Transferring Control in Window Applications* on page 9-22 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:

```

                                F O C U S   W I N D O W   P A I N T E R
+-----+
| INSTRUCTIONS :  Move cursor to selection and hit ENTER      |
|                                                         |
|               Use PF3 or PF12 to undo a selection          |
|               Use PF1 for help                             |
+-----+
                                +-----+
                                |Select one of the following: |
                                +-----+
                                |Create a new window           |
                                |Edit an existing window        |
                                |Delete an existing window     |
                                |Run the window file            |
                                |Switch window files            |
                                |Utilities                       |
                                |End                             |
                                |Quit without saving changes    |
                                +-----+

```

The following table summarizes the options on the Main Menu, along with illustrations of screens that appear when you select some of the options:

Menu Option	Description
<b>Create a new window</b>	Brings up the Window Creation Menu. You can select the type of window you want to create.
<b>Edit an existing window</b>	Brings up a list of windows in your current window file. You can select the one you want to edit.

```

File: SAMPLE          F O C U S   W I N D O W   P A I N T E R

+-----+
| INSTRUCTIONS :   Move cursor to selection and hit ENTER |
|                 Use PF3 or PF12 to undo a selection   |
|                 Use PF1 for help                       |
+-----+

+-----+
|Select window to edit:  |
+-----+
|BORDER  This window borders all my screens.  |
|BANNER  Banner for application MAIN menu.    |
|MAIN    User can report, graph, or exit.    |
|EXECTYPE Create a FOCEXEC or run an existing one. |
|EXECNAME Select an existing FOCEXEC from list. |
+-----+
    
```

Menu Option	Description
<b>Delete an existing window</b>	Brings up a list of windows in your current window file. You can select the one you want to delete.

```

File: SAMPLE          F O C U S   W I N D O W   P A I N T E R

+-----+
| INSTRUCTIONS :   Move cursor to selection and hit ENTER |
|                 Use PF3 or PF12 to undo a selection   |
|                 Use PF1 for help                       |
+-----+

+-----+
|Select window to delete:  |
+-----+
|BORDER  |
|BANNER  |
|MAIN    |
|EXECTYPE|
|EXECNAME|
+-----+
    
```

Menu Option	Description
<b>Run the window file</b>	<p>Brings up a list of windows in your current window file. You can select the one from which you want to start running the window file.</p> <p>After the window file is run, the windows' amper variable values are displayed. The display includes the first 20 characters of each value.</p> <p>This option shows you how your windows work without executing the FOCEXEC. Use this option to test your window file.</p>
<b>Switch Window files</b>	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.
<b>Utilities</b>	Brings up the Utilities Menu, which is discussed in <i>Utilities Menu</i> on page 9-72.
<b>End</b>	Returns you to native FOCUS. All work that you saved during the Window Painter session is kept.
<b>Quit without saving</b>	Returns you to native FOCUS. All work that you 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. You will see the following screen:

```

+-----+
| 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     |
|               +-----+
|

```

You will first need to select the type of window you will create. You will then be asked to enter an 8-character name and an optional 40-character description. These are for your use only; they 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: it indicates that any character or sequence of characters can occupy that position. In CMS, an asterisk used in file-identification criteria can be embedded (for example, \*DEPT FOCEXEC); the asterisk can be used in the file name, the file type, and the file mode. In MVS, 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 in CMS must specify the file name first, the file type second, and an optional file mode third. If the file mode is not specified, it defaults to an asterisk.
- File-identification criteria in MVS 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 will be prompted for the window name and heading. Once a window has been specified, you will see the Window Design screen.

For complete information about the types of windows you can create in Window Painter, see *Types of Windows You Can Create* on page 9-4.

The next screen displayed is the Window Design Screen, discussed in *Window Design Screen* on page 9-59. 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.

When you press Enter this time, the window appears, with its heading at the top. You can enlarge it, type text in it, and move it around the screen.

```
File: SAMPLE          FOCUS WINDOW PAINTER

                                     +-----+
                                     |This line is the window heading.|
                                     +-----+
                                     |                                     |
                                     +-----+

Wind: P258          Typ: Menu (vert)  PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add
```

The Window Design Screen lets you use the keyboard to manipulate the window you are creating.



The following chart summarizes Window Design Screen key functions in all window types.

<b>PF Key</b>	<b>Function</b>
<b>PF1</b>	Displays a window of help information.
<b>PF2</b>	Displays the Window Options menu. This menu is discussed in <i>Window Options Menu</i> on page 9-61.
<b>PF3</b>	Displays the exit menu. You can select: <ul style="list-style-type: none"><li>• Exiting from the Window Design Screen while saving your work.</li><li>• Quitting from the Screen without saving your work.</li><li>• Continuing your work.</li></ul>
<b>PF4</b>	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.
<b>PF5</b>	Gets the GOTO value, the Return value and the FOCEXEC name for the active window.
<b>PF6</b>	Sets the return value of the line that the cursor is on.
<b>PF7</b>	Scrolls the window up if the window contents extend beyond the top border.
<b>PF8</b>	Scrolls the window down if the window contents extend beyond the bottom border.
<b>PF9</b>	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.
<b>PF10</b>	Deletes the line of window contents identified by the current cursor position. If the window contents do not extend beyond the window borders, then the window itself will be reduced by one line.
<b>PF11</b>	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, then the window itself will increase by one line.

PF Key	Function
<b>PF12</b>	Provides the same function as the PF3 key.
<b>PF13 - PF24</b>	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. This reminds you that there are more lines of contents that are hidden beyond that border. You can view these lines by scrolling the window 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. When you 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:

Exit this menu	
Goto value	
Return value	
FOCEXEC name	
Heading	
Description	
Show a window	
Unshow a window	
Display list	
Hide list	
Popup	(Off)
Help window	
Line break	
Multi select	(Off)
Quit	PF3
Conceal option	
Switch window	

Would you like to:
Create a report?
Create a graph?
Exit?

Wind: MAIN      Typ: Menu (vert)    PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add

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
<b>Goto value</b>	<p>Selecting this option lets you specify the next window in the path from this selection field or window. You will be 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 you chose for it.)</p> <p>In menu windows, goto values are assigned to each menu item. In other windows, there is a single goto value for the entire window.</p> <p>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 and you will be 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.</p>

```

+-----+
|Enter name of next window to go to.| -----+
|Just 'Enter' for exit.             | you like to: |
+-----+ -----+
|EXECTYPE |           | Create a report? |
+-----+           |           |
|           |           | Create a graph?   |
|           |           |           |
|           |           | Exit?             |
|           |           |           |
+-----+           +-----+

```

Wind: MAIN      Typ: Menu (vert)    PF1=Help    Z=Menu    4=Size    9=Move    10=Del    11=Add

Menu Option	Description
<b>Return value</b>	<p>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, whose 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.</p> <p>To assign a return value, your cursor must be on the proper line when the Window Options Menu is brought up. Select Return value from the Window Options Menu and you will be 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. The return value is assigned.</p>

```

+-----+
|Enter return value for the line:| ld you like to: |
+-----+
|RPT          | reate a report?                          |
+-----+
          | Create a graph?                                |
          | |                                                |
          | Exit?                                        |
          | |                                                |
          +-----+

```

Wind: MAIN      Typ: Menu (vert)    PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add

<b>Menu Option</b>	<b>Description</b>
<b>FOCEXEC name</b>	Attaches a FOCEXEC to each menu selection of the window. The FOCEXEC is executed when the menu item is selected.
<b>Heading</b>	Changes the heading of any window you are working on. You can also add or remove a heading.
<b>Description</b>	Changes the description of any window you are working on.
<b>Show a window</b>	Used only during window editing, brings another window onto the screen for reference. You cannot edit the second window.
<b>Unshow a window</b>	Removes the shown window from the display.

Menu Option	Description
<b>Display list</b>	<p>Enables you to specify a list of up to 16 windows that will be visible when this window is displayed during execution.</p> <p>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.</p> <p>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 of them can be displayed on the screen at one time.</p>

```

+-----+
|Display list:|
+-----+
|BORDER |
|BANNER |
+-----+

+-----+
|Select one of these screens:|
+-----+
|EXECTYPE|
|EXECNAME|
+-----+

+-----+
|          Would you like to:          |
+-----+
| Create a report?                      |
|                                         |
| Create a graph?                       |
|                                         |
| Exit?                                  |
|                                         |
+-----+

Wind: MAIN      Typ: Menu (vert)  PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add

```

Menu Option	Description
<b>Hide list</b>	Allows you to specify windows that will 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 will be hidden except those in the display list. -- If you do not hide a window that was displayed, it will remain on the screen until another window that includes it on a hide list is displayed during execution.

```

+-----+
|Hide list: |
+-----+
|EXECNAME |
+-----+

+-----+
|Select one of these options:|
+-----+
|All |
|BORDER |
|BANNER |
+-----+

+-----+
| Would you like to: |
+-----+
| Create a report? |
| Create a graph? |
| Exit? |
+-----+

Wind: MAIN Typ: Menu (vert) PF1=Help 2=Menu 4=Size 9=Move 10=Del 11=Add
    
```

Menu Option	Description
<b>Popup (Off/On)</b>	<p>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.</p>
<b>Help window</b>	<p>Lets 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.</p> <p>Note that if the PFKEY option is specified in the -WINDOW command, you will have to explicitly set a PF key as the HELP key, as described in <i>Testing Function Key Values</i> on page 9-26.</p> <p>When selecting the Help window option, you will be asked to supply the name of the Help window file that contains the Help window. Next, you will be asked to supply the name of the Help window itself. The Help window can be an existing window, or one that you will create.</p> <p>If the Help window displays field names, it qualifies duplicates with the segment name.</p> <p>You can use any window type for a Help window. A text display window is easiest, except when you want to supply different help information for each item in a vertical menu, horizontal menu (that is, item-specific help).</p> <p>If you wish to assign item-specific help, use a file contents window that displays a file containing text in the following format:</p> <pre data-bbox="618 1142 1253 1216">=&gt;HELPPFILE =&gt; menu item this is the Help message you want the user to see.</pre> <p>where:</p> <pre data-bbox="618 1286 1262 1373">=&gt;     Is entered with an equal sign (=) and a greater-than sign (&gt;).</pre> <p>HELPPFILE Must be uppercase.</p>



<p><b>Help window</b> (continued)</p>	<p><i>menu item</i></p> <p>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.</p> <p>For example, if the first three lines of a vertical menu are</p> <pre>(1) Generate a sales report (2) Generate a stock report</pre> <p>and there are three blank spaces between the left border of the window and the beginning of the text, then the file containing help text could look like this:</p> <pre>=&gt;HELPPFILE =&gt;  (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. =&gt;  (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.</pre> <p>The lines immediately following the menu item text are displayed when the user positions the cursor on the menu item and presses PF1.</p> <p>In some cases you may wish to assign topic-specific help, but you may want the help text for some of the topics to be contained in a separate file. In these cases, 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.</p> <p>In CMS, use this file-identification format:</p> <pre>FILENAME= filename filetype [filemode]</pre> <p>In MVS, use this file-identification format:</p> <pre>FILENAME= membername ddname</pre> <p>To assign one set of instructions that can be used for multiple menu items, use the following syntax:</p> <pre>=&gt;DEFAULT This text appears when you have not written topic-specific help.</pre>
---	---

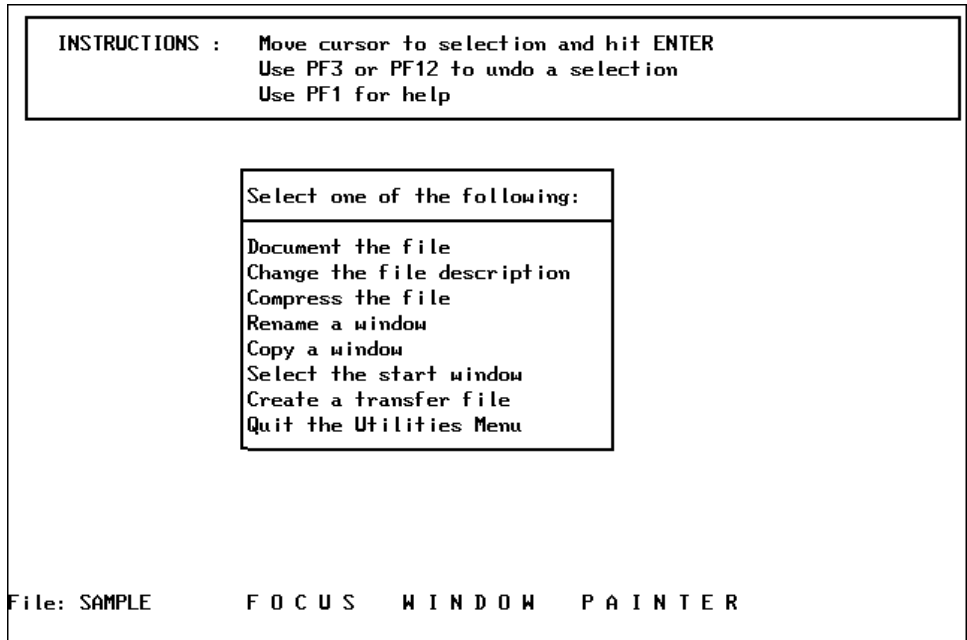
<p><b>Help window</b> (continued)</p>	<p>The DEFAULT text must be the last section in the Help file.</p> <p>Lines beginning with an * are comment lines that are not displayed.</p> <p>What follows is an example of a topic-specific Help file for the Main Menu used in the tutorial.</p> <pre>=&gt;HELPPFILE *Help file for tutorial/Main Menu =&gt; Create a report? Choose this option if you wish to create a new report. =&gt; Create a graph? Select this option if you wish to create pie charts, bar charts or other graphics. =&gt; Exit? If you wish to leave the application, choose this option.</pre>
<p><b>Line-break</b></p>	<p>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.</p> <p>When you select this option, you will see:</p> <pre>None New line before value New line after value Both</pre> <p>where:</p> <pre>None</pre> <p>Places return value directly after preceding value. If there is not enough room on this line, return value is placed on the next line.</p> <pre>New line before value</pre> <p>Places return value on the next line.</p> <pre>New line after value</pre> <p>Places return value on the same line as preceding value. Places next return value on next line.</p> <pre>Both</pre> <p>Places return value on a line by itself.</p>

<b>Multi-Select</b>	<p>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 eight windows can be stored at one time.</p> <p>When you select this option, you will see:</p> <pre>-Select Multi(On )</pre> <p>During execution, the user selects individual values by pressing PF9. After all selections have been made, the user presses Enter.</p> <p>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:</p> <pre>SET PF09=SELECT</pre> <p>You can also set a different PF key for selecting multiple items.</p> <p>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.</p> <p>The return value collected for a window using the Multi-Select option is the number of values selected by the user.</p> <p>To retrieve the individual values, issue a special WINDOW call, as follows:</p> <pre>-WINDOW windowfile windowname GETHOLD</pre> <p>where:</p> <pre>windowfile</pre> <p>Is the name of the window file.</p> <pre>windowname</pre> <p>Is the name of the Multi-Select window.</p> <pre>GETHOLD</pre> <p>Is the special parameter that retrieves one value at a time from the temporary file.</p> <p>The value is assigned to the variable &amp;windowname.</p>
---------------------	--

<b>Multi-Select</b> <i>(continued)</i>	<p>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.</p> <p>For each value to be retrieved, you will need a -WINDOW command with the GETHOLD option. Each value will be stored in &amp;windowname. If you wish to use this value, we recommend assigning it to another variable. For example, if the return value has the value 4, you would issue the special -WINDOW command four times; each time you would collect the value from &amp;windowname. Alternatively, you could perform a loop.</p> <p>Note that -WINDOW with the GETHOLD option will not transfer control from the FOCEXEC to the Window facility.</p>
<b>Quit</b>	Returns you to the Window Painter Entry Menu.
<b>Input fields</b>	Input fields pertain to Multi Input Windows. Selecting the field takes you to that field.
<b>Menu text</b>	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.
<b>Text line (x+1)</b>	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.
<b>Pulldown (off/on)</b>	If the setting is ON, placing the cursor on an item in a horizontal menu can display an associated pulldown menu. The default setting is OFF. Turn the setting ON by positioning the cursor on this option and pressing Enter. — The pulldown menu must be a vertical menu and must be assigned as the goto value for the horizontal menu item. Note that setting Pulldown ON automatically shuts off Menu Text.
<b>Switch window</b>	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 will be 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
<b>Document the file</b>	<p>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.</p> <p>In CMS, this option creates a file with file type TRF on your A disk.</p> <p>In MVS, 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 will temporarily allocate the PDS.</p> <p>This document contains detailed information about all the windows in the window file. It shows you the kinds of windows, their 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.</p> <p>The document is especially useful when creating a FOCEXEC, since it provides return and goto values in addition to other information.</p> <p><b>Note:</b> If you create another file with the same name, the file is not overwritten. It is appended.</p>

```

* 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
<b>Change the file description</b>	Changes the description of the current window.
<b>Compress the file</b>	This utility is provided to help you save space in memory. It allows space made available by deleted or edited windows to be reused.
<b>Rename a window</b>	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.
<b>Copy a window</b>	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.
<b>Select the start window</b>	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 will select the first window created to be the start window.

Menu Option	Description
<b>Create a transfer file</b>	<p>Creates a file to be transferred for use with the Window facility in PC/FOCUS, TSO or another FOCUS environment.</p> <p>In CMS, this option creates a file with file type TRF on your A disk.</p> <p>In MVS, this option creates a member of the TRF PDS; that PDS must have already been allocated.</p>
<b>Quit the utilities menu</b>	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; this saves you the trouble of recreating the window file from scratch in 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 the XFER command.
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.

- In CMS, this Window Painter option automatically creates a transfer file with a file type of TRF on your A disk.
- In MVS, 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 are only going to use the transfer file during the current FOCUS session: Window Painter will temporarily allocate 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 will then be prompted for the name of the new transfer file. Enter any name that you wish; it can have the same name as the window file, or an entirely new name. In CMS the name that you enter is the file name; in MVS it is the member 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, as described below. 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 will be asked to select which window(s) you want to transfer. You can select

`All`

to transfer all of the windows in the current window file, or you can 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 then select the new window. Window Painter will merge 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 the XFER command. The XFER command is described in Chapter 6, *Enhancing Application Performance*.

## 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, on the other hand, the new environment supports features 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 their 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).

<b>Attribute</b>	<b>Description</b>
<b>FILENAME</b>	The name of the original window file.
<b>DESCRIPTION</b>	A comment field describing the file.
<b>WINDOWNAME</b>	The name of the window.
<b>TYPE</b>	The type of window: <ol style="list-style-type: none"> <li>1. Vertical menu</li> <li>2. Text input window</li> <li>3. Text display window</li> <li>4. Horizontal menu</li> <li>5. File names window</li> <li>6. Field names window</li> <li>7. File contents window</li> <li>8. Return value display window</li> <li>9. Execution window</li> <li>10. Multi-input window</li> </ol>
<b>COMMENT</b>	A comment field describing the window.
<b>TRANSLATE</b>	Type of input for text input windows (Type 2). <ol style="list-style-type: none"> <li>0 Allow mixed case input.</li> <li>1 Allow numeric input only.</li> <li>2 Translate input to uppercase.</li> </ol>
<b>ROW</b>	The row number of the upper left corner of the window.
<b>COLUMN</b>	The column number of the upper left corner of the window.
<b>HEIGHT</b>	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 will fit in the window frame, the PF7 and PF8 keys can scroll the window.
<b>TEXT LINE</b>	Position of menu text. Values are: +1, +2, -1, -2.
<b>WIDTH</b>	The width of the window frame, not including the border.

<b>Attribute</b>	<b>Description</b>
<b>INPUT FIELDS</b>	Fields for multi-input windows.
<b>WINDOW</b>	The number of lines in the actual window frame (not the number of lines of window data). This does not include borders.
<b>POPUP</b>	Sets the pop-up feature. 0 This will not be a pop-up window. 1 This will be a pop-up window.

*Figure 9-1. Transfer File Syntax: Window File Attributes*

<b>Attribute</b>	<b>Description</b>
<b>BORDER</b>	Sets the window border. 0 There will be no window border. 1 There will be a window border. 2 There will be a window border. Options 1 and 2 both result in a basic window border.
<b>HEADLEN</b>	Length of the window heading. If this value is 0, there will be no heading.
<b>RETURN</b>	Sets the line break feature for use with return value display windows. 0 Line break will not be used. 1 New line before this return value. 2 New line after this return value. 3 New line before and after this value.
<b>MULTI</b>	Sets the multi-select feature. 0 This will not be a multi-select window. 1 This will be a multi-select window.
<b>HEADING</b>	The text of the window heading.
<b>HELP</b>	The name of the help window for this window.
<b>HELPPFILE</b>	The name of the window file that contains the help window.

Attribute	Description
<b>DISPLAY</b>	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.
<b>HIDE</b>	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.

*Figure 9-2. Transfer File Syntax: Window Attributes*

Attribute	Description
<b>DATA</b>	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).
<b>GOTO</b>	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 will return to Dialogue Manager.
<b>VALUE</b>	<p>The return value supplied if this line is selected by the user. This value will be placed in the amper variable &amp;windowname, where windowname is the name of the window.</p> <p>For file names windows (TYPE = 5), this is the file selection criteria (including asterisks) of the file names to be displayed.</p> <p>For field names windows (TYPE = 6), this is the name of the Master File whose fields will be displayed.</p> <p>For file contents windows (TYPE = 7), this is the name of the file whose contents are to be displayed.</p>

*Figure 9-3. Transfer File Syntax: Window Line Attributes*

## 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 will occur. 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 will automatically be 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.** 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 will automatically translate 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 File Type or ddname
<b>.FEX</b>	FOCEXEC
<b>.MAS</b>	MASTER
<b>.ERR</b>	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 it will use 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 in MVS, 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 will temporarily allocate the PDS.

### Syntax

#### How to Compile a Transfer File

```
WINDOW COMPILE windowfile
```

where:

*windowfile*

Is the name of the transfer file.

In CMS, this must be the file name of a file with file type TRF.

The command will create a new file with the file name specified in the command, and a file type of FMU, on the A disk. Once it has been created, you can move the file to any disk you wish.

In MVS, this must be a member name of a member of a PDS allocated to ddname TRF.

The command will create 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 will search 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 will be used.

Note that if you compile a transfer file and later make changes to it, you will need to recompile the updated transfer file: otherwise, FOCUS will continue to use the older, unchanged compiled file.



---

## APPENDIX A

# Master Files and Diagrams

### Topics:

- Creating Sample Data Sources
- The EMPLOYEE Data Source
- The JOBFIL Data Source
- The EDUCFILE Data Source
- The SALES Data Source
- The PROD Data Source
- The CAR Data Source
- The LEDGER Data Source
- The FINANCE Data Source
- The REGION Data Source
- The COURSES Data Source
- The EMPDATA Data Source
- The EXPERSON Data Source
- The TRAINING Data Source
- The PAYHIST File
- The COMASTER File
- The VideoTrk and MOVIES Data Sources
- The VIDEOTR2 Data Source
- The Gotham Grinds Data Sources

This appendix contains data source descriptions and structure diagrams for the examples used throughout the documentation.

## Creating Sample Data Sources

You can create the sample data sources on your user ID by executing the procedures specified below. These FOCEXECs are supplied with FOCUS. If they are not available to you or if they produce error messages, contact your systems administrator.

To create these files, first make sure you have read access to the Master Files.

Data Source	Load Procedure Name
EMPLOYEE, EDUCFILE, and JOBFILE	Under CMS enter:  <code>EX EMPTEST</code>  Under MVS, enter:  <code>EX EMPTSO</code>  These FOCEXECs also test the data sources by generating sample reports. If you are using Hot Screen, remember to press either Enter or the PF3 key after each report. If the EMPLOYEE, EDUCFILE, and JOBFILE data sources already exist on your user ID, the FOCEXEC will replace the data sources with new copies. This FOCEXEC assumes that the high-level qualifier for the FOCUS data sources will be the same as the high-level qualifier for the MASTER PDS that was unloaded from the tape.
SALES PROD	<code>EX SALES</code> <code>EX PROD</code>
CAR	none (created automatically during installation)
LEDGER FINANCE REGION COURSES EXPERSON	<code>EX LEDGER</code> <code>EX FINANCE</code> <code>EX REGION</code> <code>EX COURSES</code> <code>EX EXPERSON</code>
EMPDATA TRAINING	<code>EX LOADEMP</code> <code>EX LOADTRAI</code>
PAYHIST	none (PAYHIST DATA is a sequential data source and is allocated during the installation process)
COMASTER	none (COMASTER is used for debugging other Master Files)
VideoTrk and MOVIES	<code>EX LOADVTRK</code>
VIDEOTR2	<code>EX LOADVID2</code>
Gotham Grinds	<code>EX LOADGG</code>

## The EMPLOYEE Data Source

The EMPLOYEE data source contains data about a company's employees. Its segments are:

- EMPINFO, which contains employee IDs, names, and positions.
- FUNDTRAN, which specifies employees' direct deposit accounts. This segment is unique.
- PAYINFO, which contains the employee's salary history.
- ADDRESS, which contains employees' home and bank addresses.
- SALINFO, which contains data on employees' monthly pay.
- DEDUCT, which contains data on monthly pay deductions.

The EMPLOYEE data source also contains cross-referenced segments belonging to the JOBFIL and EDUCFIL files, described later in this appendix. The segments are:

- JOBSEG (from JOBFIL), which describes the job positions held by each employee.
- SECSEG (from JOBFIL), which lists the skills required by each position.
- SKILLSEG (from JOBFIL), which specifies the security clearance needed for each job position.
- ATTNDSEG (from EDUCFIL), which lists the dates that employees attended in-house courses.
- COURSEG (from EDUCFIL), which lists the courses that the employees attended.

## The EMPLOYEE Master File

```

FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMP INFO, 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, $$
  FIELDNAME=CURR_SAL, ALIAS=CSAL, FORMAT=D12.2M, $
  FIELDNAME=CURR_JOBCODE, ALIAS=CJC, FORMAT=A3, $
  FIELDNAME=ED_HRS, ALIAS=OJT, FORMAT=F6.2, $
SEGNAME=FUNDTRAN, SEGTYPE=U, PARENT=EMP INFO
  FIELDNAME=BANK_NAME, ALIAS=BN, FORMAT=A20, $
  FIELDNAME=BANK_CODE, ALIAS=BC, FORMAT=I6S, $
  FIELDNAME=BANK_ACCT, ALIAS=BA, FORMAT=I9S, $$
  FIELDNAME=EFFECT_DATE, ALIAS=EDATE, FORMAT=I6YMD, $
SEGNAME=PAYINFO, SEGTYPE=SH1, PARENT=EMP INFO
  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=EMP INFO
  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=EMP INFO
  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=EMP INFO, CRFILE=EDUCFILE, CRKEY=EMP_ID,$
SEGNAME=COURSEG, SEGTYPE=KLU, PARENT=ATTNDSEG, CRFILE=EDUCFILE,$

```

# The EMPLOYEE Structure Diagram

STRUCTURE OF FOCUS FILE EMPLOYEE ON 09/15/00 AT 10.16.27

```

EMPINFO
01      S1
*****
*EMP_ID      **
*LAST_NAME   **
*FIRST_NAME  **
*HIRE_DATE   **
*            **
*****
I
+-----+-----+-----+-----+
I      I      I      I      I
I FUNDTRAN  I PAYINFO  I ADDRESS  I SALINFO  I ATTNDSG
02  I U      03  I SH1   07  I S1     08  I SH1   10  I KM
*****
*BANK_NAME * *DAT_INC ** *TYPE ** *PAY_DATE ** :DATE_ATTEND ::
*BANK_CODE * *PCT_INC ** *ADDRESS_LN1 ** *GROSS ** :EMP_ID ::K
*BANK_ACCT * *SALARY ** *ADDRESS_LN2 ** * ** : **
*EFFECT_DATE * *JOBCODE ** *ADDRESS_LN3 ** * ** : **
* * * ** * ** * ** : **
***** :*****:
***** :*****:
I I EDUCFILE
I I
I I
I JOBSEG I DEDUCT I COURSEG
04  I KU      09  I S1   11  I KLU
*****
:JOBCODE :K *DED_CODE ** :COURSE_CODE :
:JOB_DESC : *DED_AMT ** :COURSE_NAME :
: : * ** : :
: : * ** : :
: : * ** : :
***** :*****:
I JOBFILE EDUCFILE
I
+-----+-----+
I I
I SECSEG I SKILLSEG
05  I KLU  06  I KL
*****
:SEC_CLEAR : :SKILLS :
: : :SKILL_DESC :
: : : :
: : : :
: : : :
***** :*****:
JOBFILE :*****:
JOBFILE

```

## The JOBFIL Data Source

The JOBFIL data source contains information on a company's job positions. Its segments are:

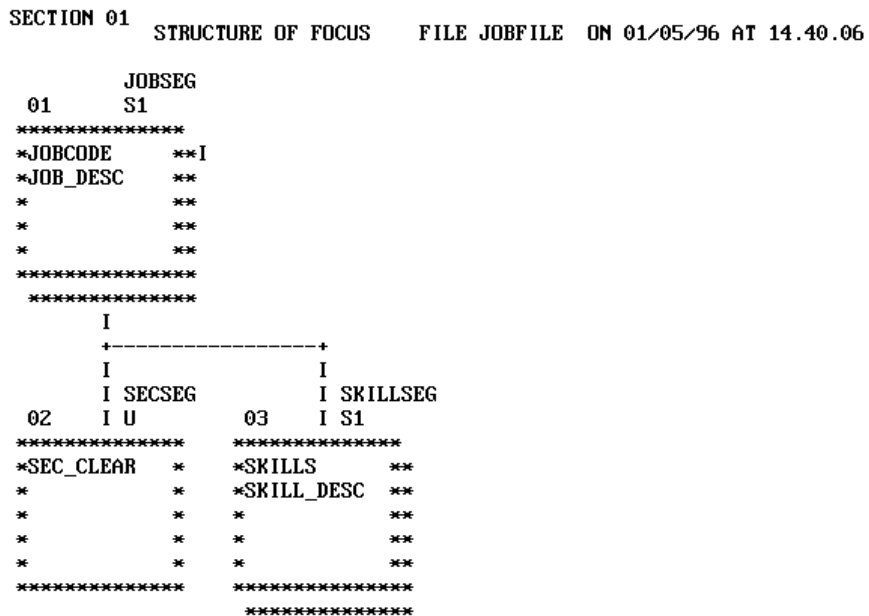
- JOBSEG describes what each position is. The field JOBCODE in this segment is indexed.
- SKILLSEG lists the skills required by each position.
- SECSEG specifies the security clearance needed, if any. This segment is unique.

## The JOBFIL Master File

```

FILENAME=JOBFIL ,SUFFIX=FOC
SEGNAME=JOBSEG ,SEGTYPE=S1
FIELD=JOBCODE ,ALIAS=JC ,USAGE=A3 ,INDEX=I,$
FIELD=JOB_DESC ,ALIAS=JD ,USAGE=A25 ,
SEGNAME=SKILLSEG ,SEGTYPE=S1 ,PARENT=JOBSEG
FIELD=SKILLS ,ALIAS= ,USAGE=A4 ,
FIELD=SKILL_DESC ,ALIAS=SD ,USAGE=A30 ,
SEGNAME=SECSEG ,SEGTYPE=U ,PARENT=JOBSEG
FIELD=SEC_CLEAR ,ALIAS=SC ,USAGE=A6 ,
    
```

## The JOBFIL Structure Diagram



## The EDUCFILE Data Source

The EDUCFILE data source contains data on a company's in-house courses. Its segments are:

- COURSESEG 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.

## The EDUCFILE Master File

```

FILENAME=EDUCFILE ,SUFFIX=FOC
SEGNAME=COURSESEG ,SEGTYPE=S1
  FIELD=COURSE_CODE ,ALIAS=CC           ,USAGE=A6           ,$
  FIELD=COURSE_NAME ,ALIAS=CD           ,USAGE=A30          ,$
SEGNAME=ATTNDSEG ,SEGTYPE=SH2 ,PARENT=COURSESEG
  FIELD=DATE_ATTEND ,ALIAS=DA           ,USAGE=I6YMD        ,$
  FIELD=EMP_ID      ,ALIAS=EID         ,USAGE=A9           ,INDEX=I,$

```

## The EDUCFILE Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE EDUCFILE ON 01/05/96 AT 14.45.44
          COURSESEG
01      S1
*****
*COURSE_CODE **
*COURSE_NAME **
*           **
*           **
*           **
*****
          I
          I
          I
          I ATTNDSEG
02      I SH2
*****
*DATE_ATTEND **
*EMP_ID      **I
*           **
*           **
*           **
*****
*****

```

## The SALES Data Source

The SALES data source records sales data for a dairy company (or a 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. Note the following about fields in this segment:
  - The PROD\_CODE field is indexed.
  - The RETURNS and DAMAGED fields have the MISSING=ON attribute.

## The 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=1, $
  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, $
```



# The SALES Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE SALES   ON 01/05/96 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 **
*          **
*****
*****

```

## The 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.

## The 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, $
```

## The PROD Structure Diagram

```
SECTION 01  
          STRUCTURE OF FOCUS   FILE PROD   ON 01/05/94 AT 14.57.38  
  
          PRODUCT  
01      S1  
*****  
*PROD_CODE **I  
*PROD_NAME **  
*PACKAGE **  
*UNIT_COST **  
* **  
*****  
*****
```

## The CAR Data Source

The CAR data source contains 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.
- WARRANT lists the type of warranty.
- EQUIP lists standard equipment.

The aliases in the CAR Master File are specified without the ALIAS keyword.

## The 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=WARRANT,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=WARRANTY,WARR,A40,$
SEGNAME=EQUIP,SEGTYPE=S1,PARENT=COMP
  FIELDNAME=STANDARD.EQUIP.A40.$

```

## The CAR Structure Diagram

```

SECTION 01                STRUCTURE OF FOCUS   FILE CAR   ON 01/05/96 AT 14.59.29

                                ORIGIN
                                S1
*****
*COUNTRY      **I
*
*
*
*****
                                I
                                I
                                I
                                I COMP
02          I S1
*****
*CAR          **
*
*
*
*****
                                I
                                I-----+-----+-----+-----+-----+
                                I          I          I
                                I CARREC    I WARRANT    I EQUIP
03          I S1          06          I S1          07          I S1
*****          *****          *****
*MODEL        **          *WARRANTY    **          *STANDARD    **
*
*
*
*****          *****          *****
*****          *****          *****
                                I
                                I
                                I
                                I BODY
04          I S1
*****
*BODYTYPE     **
*SEATS        **
*DEALER_COST  **
*RETAIL_COST  **
*
*****
                                I
                                I
                                I
                                I SPECS
05          I U
*****
*LENGTH       *
*WIDTH        *
*HEIGHT       *
*WEIGHT       *
*
*****

```

## The LEDGER Data Source

The LEDGER data source lists accounting information. 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.

### The LEDGER Master File

```
FILENAME=LEDGER, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
FIELDNAME=YEAR , , FORMAT=A4, $
FIELDNAME=ACCOUNT, , FORMAT=A4, $
FIELDNAME=AMOUNT , , FORMAT=I5C,$
```

### The LEDGER Structure Diagram

```
SECTION 01          STRUCTURE OF FOCUS   FILE LEDGER   ON 01/05/96 AT 15.07.56

          TOP
01        S2
*****
*YEAR          **
*ACCOUNT       **
*AMOUNT        **
*              **
*              **
*****
*****
```

## The FINANCE Data Source

The FINANCE data source contains financial information 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.

## The FINANCE Master File

```
FILENAME=FINANCE, SUFFIX=FOC,$
SEGNAME=TOP, SEGTYPE=S2,$
FIELDNAME=YEAR , , FORMAT=A4, $
FIELDNAME=ACCOUNT, , FORMAT=A4, $
FIELDNAME=AMOUNT , , FORMAT=D12C,$
```

## The FINANCE Structure Diagram

```
SECTION 01          STRUCTURE OF FOCUS      FILE FINANCE  ON 01/05/96 AT 15.17.08
      TOP
01      S2
*****
*YEAR      **
*ACCOUNT   **
*AMOUNT    **
*          **
*          **
*****
*****
```

## The REGION Data Source

The REGION data source lists account information for the east and west 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.

### The 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,$
```

### The REGION Structure Diagram

```
SECTION 01      STRUCTURE OF FOCUS  FILE REGION  ON 01/05/96 AT 15.18.48

          TOP
01      S1
*****
*ACCOUNT      **
*E_ACTUAL     **
*E_BUDGET     **
*W_ACTUAL     **
*             **
*****
*****
```

## The COURSES Data Source

The COURSES data source describes education courses. It consists of one segment, CRSESEG1. The field DESCRIPTION has a format of TEXT (TX).

### The 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,                $$
```

### The COURSES Structure Diagram

```
SECTION 01
          STRUCTURE OF FOCUS   FILE COURSES  ON 01/05/94 AT 15.20.59

          CRSESEG1
01      S1
*****
* COURSE_CODE **I
* COURSE_NAME **
* DURATION   **
* DESCRIPTION **I
*           **
*****
*****
```



## The EMPDATA Data Source

The EMPDATA data source contains organizational data about a company's employees. It consists of one segment, EMPDATA. Note the following:

- The PIN field is indexed.
- The AREA field is a temporary one.

## The 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=DIU, ALIAS=CDIU, 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.ZM, $
FIELDNAME=HIREDATE, ALIAS=HDAT, FORMAT=YMD, $
$
DEFINE AREA/A13=DECODE DIU (NE 'NORTH EASTERN' SE 'SOUTH EASTERN'
CE 'CENTRAL' WE 'WESTERN' CORP 'CORPORATE' ELSE 'INVALID AREA');$

```

## The EMPDATA Structure Diagram

```

SECTION 01 STRUCTURE OF FOCUS FILE EMPDATA ON 01/05/96 AT 14.49.09

      EMPDATA
01    S1
*****
*PIN          **I
*LASTNAME     **
*FIRSTNAME    **
*MIDINITIAL   **
*             **
*****
*****

```

## The EXPERSON Data Source

The EXPERSON data source contains personal data about individual employees. It consists of one segment, ONESEG.

### The 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    ,,$
    
```

### The EXPERSON Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE EXPERSON ON 01/05/96 AT 14.50.58

          ONESEG
          01      S1
          *****
          *SOC_SEC_NO **
          *FIRST_NAME **
          *LAST_NAME  **
          *AGE        **
          *           **
          *****
          *****
    
```

## The TRAINING Data Source

The TRAINING data source contains training course data for employees. It consists of one segment, TRAINING. Note the following:

- The PIN field is indexed.
- The EXPENSES, GRADE, and LOCATION fields have the MISSING=ON attribute.

## The TRAINING Master File

```

FILENAME=TRAINING, SUFFIX=FOC
SEGNAME=TRAINING, SEGTYPE=SH3
FIELDNAME=PIN, ALIAS=ID, FORMAT=A9, INDEX=1, $
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,$

```

## The TRAINING Structure Diagram

```

SECTION 01
          STRUCTURE OF FOCUS   FILE TRAINING ON 12/12/94 AT 14.51.28

          TRAINING
01      SH3
*****
*PIN          **I
*COURSESTART **
*COURSECODE  **
*EXPENSES    **
*            **
*****
*****

```

## The 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.

## The 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=16YMTD, 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,$
```

## The PAYHIST Structure Diagram

```
SECTION 01          STRUCTURE OF FIX   FILE PAYHIST  ON 01/05/96 AT 14.51.59

          PAYSEG
01      S1
*****
*SOC_SEC_NO **
*DATE_OF_IN **
*AMT_OF_INC **
 *PCT_INC   **
 *          **
*****
*****
```

## The 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 lists file information.
- RECID lists segment information.
- FIELDID lists field information.
- DEFREC lists a description record.
- PASSREC lists read/write access.
- CRSEG lists cross-reference information for segments.
- ACCSEG lists DBA information.

## The COMASTER Master File

```

FILE=COMASTER, SUFFIX=COM,

SEGNAME=FILEID
FIELDNAME=FILENAME ,FILE ,A8 , ,,$
FIELDNAME=FILE SUFFIX ,SUFFIX ,A8 , ,,$

SEGNAME=RECID
FIELDNAME=SEGNAME ,SEGMENT ,A8 , ,,$
FIELDNAME=SEGTYPE ,SEGTYPE ,A4 , ,,$
FIELDNAME=SEGSIZE ,SEGSIZE ,14 , A4,$
FIELDNAME=PARENT ,PARENT ,A8 , ,,$
FIELDNAME=CRKEY ,UREY ,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, ,,$

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, ,,$

```

# The COMASTER Structure Diagram

SECTION 01

STRUCTURE OF EXTERNAL FILE COMASTER ON 12/12/94 AT 14.53.38

```

      FILEID
01      S1
*****
*FILENAME **
*FILE SUFFIX **
*
* **
* **
*****
      I
      +-----+
      I          I
      I RECID      I PASSREC
02      I N          07      I N
*****
*SEGNAME **      *READ/WRITE **
*SEGTYP E **      *
*SEGSIZE **      *
*PARENT **      *
*
* **      *
*****
      I
      +-----+
      I          I
      I FIELDID    I CRSEG
03      I N          06      I N
*****
*FIELDNAME **      *CRFILENAME **
*ALIAS **      *CRSEGNAME **
*FORMAT **      *ENCRYPT **
*ACTUAL **      *
*
* **      *
*****
      I
      I
      I
      I DEFREC
04      I N
*****
*DEFINITION **
*
* **
* **
* **
*****
      I
      I
      I
      I ACCSEG
05      I N
*****
*DBA **
*DBAFILE **
*USER **
*ACCESS **
*
* **
*****

```

## The VideoTrk and MOVIES Data Sources

The VideoTrk data source tracks customer, rental, and purchase information for a video rental business. It can be joined to the MOVIES data source. VideoTrk and MOVIES are used in examples that illustrate the use of the Maintain 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, $
```

### 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=LSTPR, ALIAS=LPRC, FORMAT=F6.2, $
    FIELDNAME=COPIES, ALIAS=NOC, FORMAT=I3, $
```



## VideoTrk Structure Diagram

```

SECTION 01
                STRUCTURE OF FOCUS      FILE VIDEOTRK ON 05/21/99 AT 12.25.19

                CUST
01             S1
*****
*CUSTID       **
*LASTNAME    **
*FIRSTNAME   **
*EXPDATE     **
*            **
*****
                I
                I
                I
                I TRANSDAT
02             I SH1
*****
*TRANSDATE   **
*            **
*            **
*            **
*            **
*****
                I
                +-----+
                I             I
                I SALES      I RENTALS
03             I S2          04     I S2
*****          *****
*PRODCODE    **   *MOVIECODE  **I
*TRANSCODE   **   *COPY       **
*QUANTITY    **   *RETURNDATE **
*TRANSTOT    **   *FEE        **
*            **   *            **
*****          *****
                *****

```

## MOVIES Structure Diagram

```
SECTION 01
                STRUCTURE OF FOCUS   FILE MOVIES   ON 05/21/99 AT 12.26.05

                MOVINFO
01             S1
*****
*MOVIECODE    **I
*TITLE        **
*CATEGORY     **
*DIRECTOR     **
*             **
*****
*****
```

## The VIDEOTR2 Data Source

The VIDEOTR2 data source tracks customer, rental, and purchase information for a video rental business. It is similar to VideoTrk but is a partitioned data source with both a Master and Access File and with a date-time field.

## The VIDEOTR2 Master File

```
FILENAME=VIDEOTR2,  SUFFIX=FOC,
ACCESS=VIDEOACX,  $
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=HYMYDI,  $
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,   $
  DEFINE DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
```

## The VIDEOTR2 Access File

On CMS,

```
MASTER VIDEOTR2
  DATANAME 'VIDPART1 FOCUS A'
  WHERE DATE EQ 1991;

  DATANAME 'VIDPART2 FOCUS A'
  WHERE DATE FROM 1996 TO 1998;

  DATANAME 'VIDPART3 FOCUS A'
  WHERE DATE FROM 1999 TO 2000;
```

On MVS, the data set names include your user ID as the high-level qualifier:

```
MASTER VIDEOTR2
  DATANAME userid.VIDPART1.FOCUS
  WHERE DATE EQ 1991;

  DATANAME userid.VIDPART2.FOCUS
  WHERE DATE FROM 1996 TO 1998;

  DATANAME userid.VIDPART2.FOCUS
  WHERE DATE FROM 1999 TO 2000;
```

## The VIDEOTR2 Structure Diagram

STRUCTURE OF FOCUS FILE VIDEOTR2 ON 09/27/00 AT 16.45.48

```

          CUST
01      S1
*****
*CUSTID      **
*LASTNAME    **
*FIRSTNAME   **
*EXPDATE     **
*            **
*****
          I
          I
          I
          I TRANSDAT
02      I SH1
*****
*TRANSDATE   **
*            **
*            **
*            **
*            **
*****
          I
          +-----+
          I          I
          I SALES    I RENTALS
03      I S2        04      I S2
*****          *****
*TRANSCODE   **  *MOVIECODE  ** I
*QUANTITY    **  *COPY        **
*TRANSTOT    **  *RETURNDATE **
*            **  *FEE         **
*            **  *            **
*****          *****
*            **  *            **

```

# The Gotham Grinds Data Sources

Gotham Grinds is a group of data sources that contain information about a specialty items company.

## The GGDEMOG Data Source

The GGDEMOG data source 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.

### The 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=AUGHHS298, 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=AUGHHI98, 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=P15T01998, ALIAS=E09, FORMAT=I09, TITLE='15 to 19',
DESC='Population 15 to 19 years old', $
FIELD=P20T02998, ALIAS=E10, FORMAT=I09, TITLE='20 to 29',
DESC='Population 20 to 29 years old', $
FIELD=P30T04998, ALIAS=E11, FORMAT=I09, TITLE='30 to 49',
DESC='Population 30 to 49 years old', $
FIELD=P50T06498, 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', $
```

## The GGDEMOG Structure Diagram

```
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=    1 ( REAL=    1 VIRTUAL=    0 )
NUMBER OF FIELDS=     12 INDEXES=    1 FILES=     1
TOTAL LENGTH OF ALL FIELDS= 101
SECTION 01
                STRUCTURE OF FOCUS      FILE GGDEMOG  ON 09/17/96 AT 12.18.05

                GGDEMOG
01             S1
*****
*ST             **I
*HH             **
*AVGHHSZ98     **
*MEDHHI98     **
*              **
*****
*****
```

## The GGORDER Data Source

The GGORDER data source contains order information for Gotham Grinds. It consists of two segments, ORDER01 and ORDER02, respectively.

## The 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 , $
```



## The GGPRODS Structure Diagram

```

NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=    1 ( REAL=    1 VIRTUAL=    0 )
NUMBER OF FIELDS=      7 INDEXES=    2 FILES=      1
TOTAL LENGTH OF ALL FIELDS=    63
SECTION 01
                STRUCTURE OF FOCUS   FILE GGPRODS   ON 09/17/96 AT 12.21.12

                GGPRODS
01             S1
*****
*PRODUCT_ID   **I
*VENDOR_CODE  **I
*PRODUCT_DES>***
*VENDOR_NAME  **
*              **
*****

```

## The GGSales Data Source

The GGSales data source contains sales information for Gotham Grinds. It consists of one segment, SALES01.

### The 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', $

```



## The GGSALES Structure Diagram

```

NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=   1 ( REAL=    1 VIRTUAL=    0 )
NUMBER OF FIELDS=    13 INDEXES=    5 FILES=    1
TOTAL LENGTH OF ALL FIELDS= 114
SECTION 01
                STRUCTURE OF FOCUS    FILE GGSALES    ON 09/17/96 AT 12.22.19

                GGSALES
01             S1
*****
*SEQ_NO       **
*CATEGORY     **I
*PCD          **I
*REGION       **I
*             **
*****
*****

```

## The GGSTORES Data Source

The GGSTORES data source contains information for each of Gotham Grinds' 12 stores in the United States. It consists of one segment, STORES01.

### The 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', $

```

## The GGSTORES Structure Diagram

```
NUMBER OF ERRORS=      0
NUMBER OF SEGMENTS=   1 ( REAL=   1 VIRTUAL=   0 )
NUMBER OF FIELDS=    7 INDEXES=   2 FILES=    1
TOTAL LENGTH OF ALL FIELDS= 108
```

SECTION 01

STRUCTURE OF FOCUS FILE GGSTORES ON 09/17/96 AT 12.23.09

```
GGSTORES
01      S1
*****
*STORE_CODE  **I
*STATE_      **I
*STORE_NAME  **
*ADDRESS1    **
*            **
*****
*****
```

---

## APPENDIX B

# Error Messages

### Topics:

- Accessing Error Files
- Displaying Messages Online

If you need 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.

## Accessing Error Files

For CMS, the ERRORS files are:

- FOT004 ERRORS
- FOG004 ERRORS
- FOM004 ERRORS
- FOS004 ERRORS
- FOA004 ERRORS
- FSQLXLT ERRORS
- FOCSTY ERRORS
- FOB004 ERRORS

For MVS, these files are the following members in the ERRORS PDS:

- FOT004
- FOG004
- FOM004
- FOS004
- FOA004
- FSQLXLT
- FOCSTY
- FOB004

## Displaying Messages Online

To display a message online, issue the following query command at the FOCUS command level

? *n*

where *n* is the message number.

The message number and text will display 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.

---

## APPENDIX C

# Creating Your Own Subroutines

### Topics:

- Process Overview
- Considerations for Writing Subroutines
- Compilation and Storage
- Testing the Subroutine
- Example of a Custom Subroutine: The MTHNAM Subroutine
- Subroutines Written in REXX

This topic discusses how to create your own private collection of subroutines to use with FOCUS.

## Process Overview

The process of creating a subroutine involves four steps:

1. Write the subroutine for FOCUS the same way you would for a program. Use any language that supports subroutine calls; among the most common languages are FORTRAN, COBOL, PL/I, Assembler, and C.
2. Store the subroutine in a separate file; do not include it in the main program.
3. Compile the subroutine. In MVS, link-edit it; in CMS, add the subroutine to a load library using the GENSUBLL command.
4. Test the subroutine; specify it in a FOCUS command, report request, or procedure.

For example, suppose you write a program named INTCOMP that calculates the amount of money in an account earning simple interest. The program reads a record, tests if the data is acceptable, and then calls a subroutine called SIMPLE that computes the amount of money. The program and the subroutine are stored together in the same file.

The program and the subroutine shown here are written in pseudocode (a method of representing computer code in a general way):

```
Begin program INTCOMP.  
Execute this loop until end-of-file.  
    Read next record, fields: PRINCPAL, DATE_PUT, YRRATE.  
    If PRINCPAL is negative or greater than 100,000,  
        reject record.  
    If DATE_PUT is before January 1, 1975, reject record.  
    If YRRATE is negative or greater than 20%, reject record.  
    Call subroutine SIMPLE (PRINCPAL, DATE_PUT, YRRATE, TOTAL).  
    Print PRINCPAL, YEARRATE, TOTAL.  
End of loop.  
End of program.
```

```
Subroutine SIMPLE (AMOUNT, DATE, RATE, RESULT).  
Retrieve today's date from the system.  
Let NO_DAYS = Days from DATE until today's date.  
Let DAY_RATE = RATE / 365 days in a year.  
Let RESULT = AMOUNT * (NO_DAYS * DAY_RATE + 1).  
End of subroutine.
```

If you move the SIMPLE subroutine into a file separate from the main program and compile it, you can call the subroutine from FOCUS. The following report request shows how much money employees would accrue if they invested their salaries in accounts paying 12%:

```
TABLE FILE EMPLOYEE  
PRINT LAST_NAME DAT_INC SALARY AND COMPUTE  
    INVESTED/D10.2 = SIMPLE (SALARY, DAT_INC, 0.12, INVESTED);  
BY EMP_ID  
END
```

**Note:** The subroutine is designed to return only the amount of the investment, not today's date. This is because a subroutine can return only a single value to FOCUS each time it is called.

## Considerations for Writing Subroutines

When you write a subroutine for FOCUS, there are requirements and limits that you need to consider. The topic provides information about:

- Naming conventions
- Argument considerations
- Programming considerations
- Language considerations
- A programming technique that uses entry points. Entry points enable you to use one algorithm to produce different results.
- A programming technique that allows multiple subroutine calls. Multiple calls enable the subroutine to process more than 28 arguments.

### Naming Conventions

The subroutine name may consist of up to eight characters, unless the language you are using to write the subroutine supports a shorter naming convention. Each character can be a letter or number. The first character of the name must be a letter (A-Z). Special symbols are not permitted.

### Argument Considerations

When you create your arguments, consider these points:

- **The argument maximum.** Subroutine calls in FOCUS may contain up to 28 arguments. However, you can bypass this restriction if you create a subroutine that accepts multiple calls, as described in *Programming Technique: Subroutines With More Than 28 Arguments* on page C-9.
- **Types of arguments.** Subroutine calls can serve as arguments in other subroutine calls or in FOCUS functions. For types of acceptable arguments and rules, see Chapter 3, *Using Functions and Subroutines*.
- **Input arguments.** FOCUS passes input arguments to subroutines using standard conventions. Register 1 points to the list of argument addresses. Each address is a full word.



- **Output arguments.** Subroutines may return only one output argument to the FOCUS request. Place this argument last in the subroutine argument list. You can choose any format for the output argument except in Dialogue Manager statements.
- **Internal processing.** When you specify values for arguments and FOCUS passes the arguments to a subroutine,
  - Alphanumeric arguments remain unchanged.
  - Numeric arguments are converted to 8-byte, double-precision data (except in -CMS RUN and -MVS RUN statements and amper variables, as discussed below).

Various languages represent double-precision fields as declarations:

Language	Declaration
Assembler	DS, D
C	Double
COBOL	COMP-2
FORTRAN	REAL*8
PL/I	DECIMAL FLOAT (16)

- **Dialogue Manager requirements.** If you are writing a subroutine specifically for Dialogue Manager, you may need to code your subroutine to perform conversion for these situations:
  - Operating system -RUN statements. FOCUS passes all arguments from -CMS RUN, -TSO RUN, and -MVS RUN statements as alphanumeric data. If your subroutine requires numeric arguments, you may choose to have your subroutine convert these arguments into numeric format. Otherwise, the user can use the ATODBL subroutine to convert the arguments into double-precision format before passing them to the subroutine. The ATODBL subroutine is described in Chapter 3, *Using Functions and Subroutines*.
  - Operating system -RUN statements and output argument format. If the subroutine is called from a -CMS or -TSO RUN statement, the output argument is stored in the output variable in numeric format. Since FOCUS cannot interpret data stored in Dialogue Manager variables in numeric format, the data is unreadable. To prevent this, have your subroutine convert the output value into a character string.

- -SET and output argument format. If the output argument is in numeric format, the -SET statement truncates the output value to an integer, converts it to a character string, and stores the value in a specified amper variable. To prevent this, have your subroutine convert the output value into a character string. This enables the numeric value to be passed to Dialogue Manager without being truncated to an integer.

## Programming Considerations

When you plan your programming requirements, consider these points:

- Write the subroutine as a proper subroutine, not as a function.
- If the subroutine initializes variables, it must initialize them each time it is executed (serial reusability).
- Since a single FOCUS request may execute a subroutine hundreds or even thousands of times, code the subroutine as efficiently as possible.
- If you create your own subroutines in text files or text libraries, the subroutine must be 31-bit addressable.

## Language Considerations

Language considerations include:

- **Available memory.**

If you write the subroutine in a language that brings libraries into memory (for example, FORTRAN and COBOL), the libraries reduce the amount of memory available to the subroutine.

- **FORTRAN input/output operations (I/O).**

In CMS, FOCUS does not support FORTRAN input/output operations. If a subroutine written in FORTRAN must read or write data, write the I/O portions in a separate subroutine in another language.

In MVS/TSO, FOCUS does support FORTRAN input/output operations.

- **PL/I notes:**

- Do not use the RETURNS attribute.
- Include the following attribute in the procedure (PROC) statement:

```
OPTIONS (COBOL)
```

- Declare alphanumeric arguments received from FOCUS requests as  
`CHARACTER (n)`  
 where *n* is the field length as defined by the FOCUS request. Do not use the VARYING attribute.
- Declare numeric arguments received from FOCUS requests as  
`DECIMAL FLOAT (16)`  
 or  
`BINARY FLOAT (53)`
- The format of the output argument to be returned to the FOCUS request depends on how the format is described in the DEFINE or COMPUTE commands:

FOCUS Format	PL/I Declaration
<i>An</i>	<code>CHARACTER (n)</code> (Do not use the VARYING attribute.)
I	<code>BINARY FIXED (31)</code>
F	<code>DECIMAL FLOAT (6)</code> or <code>BINARY FLOAT (21)</code>
D	<code>DECIMAL FLOAT (16)</code> or <code>BINARY FLOAT (53)</code>
P	<code>DECIMAL FIXED (15)</code> (for small packed numbers, 8 bytes)  <code>DECIMAL FIXED (31)</code> (for large packed numbers, 16 bytes)

- Declare variables that are not arguments with the STATIC attribute. This avoids dynamically allocating these variables every time the subroutine is executed.

- **C language notes:**
  - Do not return a value with the return statement.
  - Declare double-precision fields as 'double'.
  - The format of the output parameter to be returned to the FOCUS request depends on how the format is defined in the request, as shown by the chart below:

FOCUS Format	C Declaration
<i>An</i>	<code>char *xxx n</code>  ( <b>Note:</b> Alphabetical fields are not terminated with a null byte and, therefore, cannot be processed by many of the string manipulation subroutines in the run-time library.)
<i>I</i>	<code>long *xxx</code>
<i>F</i>	<code>float *xxx</code>
<i>D</i>	<code>double *xxx</code>
<i>P</i>	No equivalent in C.

## Programming Technique: Entry Points

Normally, subroutines are executed starting from their first statement. However, they can be executed starting from any place in their code if you designate that place as an *entry point*. (How you designate entry points depends on the language you are using.) Each entry point has a name.

To execute a subroutine at an entry point, specify the entry name in the subroutine call instead of the subroutine name. The general syntax is:

```
{subroutine|entrypoint} (input1, input2,...{'format'|outfield})
```

Entry points enable a subroutine to use one basic algorithm to produce different results. For example, the DOWK subroutine calculates the days of the week on which dates fall. When you specify the subroutine name DOWK, you obtain a 3-letter abbreviation of the day. If you specify the entry name DOWKL, you obtain the full name. The calculation, however, is the same.

## Entry Point Example

This example illustrates how entry points work. The FTOC subroutine, written in pseudocode below, converts Fahrenheit temperatures to Centigrade. The entry point FTOK (designated by the Entry statement) sets a flag that causes 273 to be subtracted from the Centigrade temperature (Kelvin temperature). The subroutine is:

```
Subroutine FTOC (FAREN, CENTI).
Let FLAG = 0.
Go to label X.
Entry FTOK (FAREN, CENTI).
Let FLAG = 1.
Label X.
Let CENTI = (5/9) * (FAREN - 32).
If FLAG = 1 then CENTI = CENTI - 273.
Return.
End of subroutine.
```

Here is a shorter way to write the subroutine. Notice that the *kelv* output argument listed for the entry point is different from the *centi* output argument listed at the beginning of the subroutine:

```
Subroutine FTOC (FAREN, CENTI).
Entry FTOK (FAREN, KELV).
Let CENTI = (5/9) * (FAREN - 32).
KELV = CENTI - 273.
Return.
End of Subroutine.
```

To obtain the Centigrade temperature, specify the subroutine name FTOC in the subroutine call. For example:

```
CENTIGRADE/D6.2 = FTOC (TEMPERATURE, CENTIGRADE);
```

To obtain the Kelvin temperature, specify the entry name FTOK in the subroutine call. For example:

```
KELVIN/D6.2 = FTOK (TEMPERATURE, KELVIN);
```

**Note:** In CMS, subroutines can be executed from their entry points only if the subroutines are stored in libraries. You must specify these libraries in the GLOBAL command, as described in *CMS: Compilation and Storage* on page C-13.

## Programming Technique: Subroutines With More Than 28 Arguments

Subroutine call syntax cannot specify more than 28 arguments, including the output argument. To process more than 28 arguments, you must write the subroutine so that the user can specify two or more call statements to pass the arguments to the subroutine.

We recommend the following technique for writing subroutines with multiple call statements:

1. Divide the subroutine into segments. Each segment will receive the arguments passed by one corresponding subroutine call.

The argument list in the beginning of your subroutine must represent the same number of arguments in the subroutine call, including a call number argument and an output argument.

You may process some of the arguments as dummy arguments if you have an unequal number of arguments. For example, if you divide 32 arguments among six segments, the each segment processes six arguments; the sixth segment processes two arguments and four dummy arguments.

2. Include a statement at the beginning of the subroutine that reads the call number (first argument) and branches to a corresponding segment. Each segment processes the arguments from one call. (For example, number 1 branches to the first segment, number 2 to the second segment, and so on.)
3. Have each segment store the arguments it receives in other variables (which can be processed by the last segment) or accumulate them in a running total.

End each segment with a statement returning control back to the FOCUS request (RETURN statement).

4. The last segment returns the final output value to the FOCUS request.

The following sample of pseudocode illustrates the four steps:

1. Subroutine *name* (*num*, *input1*, *input2*, *input3*, *input4*, *outfield*).
  2. If NUM is 1 then goto label ONE  
else goto label TWO.
- Label ONE.
3. Let *variable* = *input1* + *input2*.  
Return.
4. Label TWO  
LET *outfield* = *variable* + *input3* + *input4*  
Return  
End of subroutine

**Note:** You can also use the entry point technique, described in *Programming Technique: Entry Points* on page C-7, to write subroutines that process more than 28 arguments.

## Syntax

### How to Use Subroutines With Multiple Call Statements

To use a subroutine that requires more than 28 arguments, you must specify two or more call statements to pass the arguments to the subroutine.

The syntax for calling a subroutine with multiple call statements is

```
dummy = subroutine (1, group1, dummy);  
dummy = subroutine (2, group2, dummy);  
.  
.  
.  
outfield = subroutine (n, groupn, outfield);
```

where:

*dummy*

Is either the name of a dummy field or its format, enclosed in single quotation marks. It must have the same format as the *outfield* argument.

**Note:** Do not specify the *dummy* argument for the last call statement; use the *outfield* argument.

*subroutine*

Is the name of the subroutine, up to eight characters long, depending on your programming language.

*n*

Is a number that identifies each subroutine call. It must be the first argument in each subroutine call. The subroutine uses this call number to branch to segments of code.

*group1...*

Are lists of input arguments passed by each subroutine call. Each group contains the same number of arguments, but no more than 26 arguments.

$26 + \text{call number} + \text{output} = 28$

*outfield*

Is the output field that contains the value returned by the subroutine. It is the filename of the field that contains the output or the format of the output value, enclosed in single quotation marks, depending on the application. It is last argument in the last call.

**Note:**

- Each subroutine call contains the same number of arguments. This is because the argument list in each call must correspond to the argument list in the beginning of the subroutine. The last call may contain several dummy arguments.
- Subroutines may require additional arguments as determined by the programmer who created the subroutine.

*Example*

**Creating a Subroutine With 32 Input Arguments**

This example illustrates how to create a subroutine with 32 input arguments using the recommended technique. It also shows how the subroutine is specified in a DEFINE command.

The ADD32 subroutine, written in pseudocode, sums 32 numbers. It is divided into six segments, each of which adds six numbers from a subroutine call. (The total number of input arguments is 36 but the last four are dummy arguments.) The sixth segment adds two arguments to the SUM variable and returns the final output value. The sixth segment does not process any values supplied for the four dummy arguments.

The subroutine is:

```
Subroutine ADD32 (NUM, A, B, C, D, E, F, TOTAL).
If NUM is 1 then goto label ONE
else if NUM is 2 then goto label TWO
else if NUM is 3 then goto label THREE
else if NUM is 4 then goto label FOUR
else if NUM is 5 then goto label FIVE
else goto label SIX.

Label ONE.
Let SUM = A + B + C + D + E + F.
Return.

Label TWO
Let SUM = SUM + A + B + C + D + E + F
Return

Label THREE
Let SUM = SUM + A + B + C + D + E + F
Return

Label FOUR
Let SUM = SUM + A + B + C + D + E + F
Return

Label FIVE
Let SUM = SUM + A + B + C + D + E + F
Return

Label SIX
LET TOTAL = SUM + A + B
Return
End of subroutine
```



To use the ADD32 subroutine, list all six call statements; each call specifying six numbers. The last four numbers, represented by zeroes, are dummy arguments. In this example, the DEFINE command stores the total of the 32 numbers in the SUM32 field.

```
DEFINE FILE EMPLOYEE
DUMMY/D10 = ADD32 (1, 5, 7, 13, 9, 4, 2, DUMMY);
DUMMY/D10 = ADD32 (2, 5, 16, 2, 9, 28, 3, DUMMY);
DUMMY/D10 = ADD32 (3, 17, 12, 8, 4, 29, 6, DUMMY);
DUMMY/D10 = ADD32 (4, 28, 3, 22, 7, 18, 1, DUMMY);
DUMMY/D10 = ADD32 (5, 8, 19, 7, 25, 15, 4, DUMMY);
SUM32/D10 = ADD32 (6, 3, 27, 0, 0, 0, 0, SUM32);
END
```

## Compilation and Storage

Once you have written your subroutine, you need to compile and store it. This topic discusses compiling and storing your subroutine for CMS and MVS.

### CMS: Compilation and Storage

On CMS, compile the subroutine and use the GENSUBLL command to add the compiled object code to a load library (filetype LOADLIB). Enter:

```
GENSUBLL ?
```

to display for online information about the command. Do not store subroutine in the FUSELIB load library (FUSELIB LOADLIB), as it may be overwritten when your site installs the next release of FOCUS.

You may also compile the subroutine and store the compiled object code either as a text file (filetype TEXT), or as a member in a text library (filetype TXTLIB). Do not store it in the FUSELIB text library (FUSELIB TXTLIB), as it may be overwritten when your site installs the next release of FOCUS.

Individual text files are easier to maintain and control. Text libraries, on the other hand, enable you to build different entry points into the subroutine (as shown in *Programming Technique: Entry Points* on page C-7). Note that there are two CMS commands regarding text libraries:

- The TXTLIB command allows you to create, add to, and delete text libraries.
- The GLOBAL TXTLIB command allows users to specify text libraries to gain access to their subroutines.

If the subroutine is written in PL/I, append this line at the end of the text file

```
ENTRY subroutine
```

where:

```
subroutine
```

Is the name of the subroutine. You can do this using your system editor.

Make sure that any subroutines that your subroutine calls are also compiled and placed in text files or libraries.

## MVS: Compilation and Storage

On MVS, compile and link-edit the subroutine and store the module in a load library. If your subroutine calls other subroutines, compile and link-edit all the subroutines together in a single module.

If the subroutine is written in PL/I, include this link-editor control statement when link-editing the subroutine

```
ENTRY subroutine
```

where:

```
subroutine
```

Is the name of the subroutine.

Do not store the subroutine in the FUSELIB load library (FUSELIB.LOAD), as it may be overwritten when your site installs the next release of FOCUS.

## Testing the Subroutine

Once you have successfully compiled your subroutine, access it and test it. In order to access the subroutine, you need to issue the GLOBAL command for CMS or the ALLOCATE command for MVS.

If an error occurs during your testing, check to see if the error is in the FOCUS request or in the subroutine. If you are uncertain about its source, apply this test:

1. Write a dummy subroutine that has the same arguments but only returns a constant.
2. Execute the request with the dummy subroutine.

If the request executes the dummy subroutine normally, the error is in your subroutine. If the request still generates an error, the error is in the request.

If you intend to make your subroutine available to other users, be sure to document what your subroutine does, what the arguments are, what formats they have, and in what order they must appear in the FOCUS subroutine call.

## Example of a Custom Subroutine: The MTHNAM Subroutine

This topic illustrates how a subroutine can be written in FORTRAN, COBOL, PL/I, BAL Assembler, and C, and then executed in a FOCUS request. The subroutine, called MTHNAM, converts a number from 1 to 12 to the full name of the corresponding month (from January to December).

The subroutine performs the following:

1. The subroutine receives the input argument from the FOCUS request as a double-precision number.
2. It adds .000001 to the number. This compensates for rounding errors. (Rounding errors can occur since floating-point numbers are approximations and may be inaccurate in the last significant digit.)
3. It moves the number into an integer field.
4. If the number is less than 1 or greater than 12, it changes the number to 13.
5. It defines a 13-element array containing the names of the months. The last element is an error message.
6. It sets the index of the array equal to the number in the integer field. It then places the corresponding array element into the output argument. If the number is 13, the argument contains the error message.
7. It passes the output argument back to FOCUS.

## The MTHNAM Subroutine Written in FORTRAN

This is a FORTRAN version of the MTHNAM subroutine. The fields are:

MTH

Is the double-precision number passed by FOCUS.

MONTH

Is the name of the month passed back to FOCUS. Since the character string 'September' contains nine letters, MONTH is a 3-element array. The subroutine passes the three elements back to FOCUS; FOCUS concatenates them into one field.

A

Is a 2-dimensional, 13 by 3 array containing the names of the months. The last three elements contain the error message.

IMTH

Is the integer representing the month.

The program is:

```
SUBROUTINE MTHNAM (MTH,MONTH)
REAL*8      MTH
INTEGER*4   MONTH(3),A(13,3),IMTH
DATA
+   A( 1,1)/'JANU'/, A( 1,2)/'ARY  '/, A( 1,3)/'   '/,
+   A( 2,1)/'FEBR'/, A( 2,2)/'UARY'/, A( 2,3)/'   '/,
+   A( 3,1)/'MARC'/, A( 3,2)/'H   '/, A( 3,3)/'   '/,
+   A( 4,1)/'APRI'/, A( 4,2)/'L   '/, A( 4,3)/'   '/,
+   A( 5,1)/'MAY  '/, A( 5,2)/'   '/, A( 5,3)/'   '/,
+   A( 6,1)/'JUNE'/, A( 6,2)/'   '/, A( 6,3)/'   '/,
+   A( 7,1)/'JULY'/, A( 7,2)/'   '/, A( 7,3)/'   '/,
+   A( 8,1)/'AUGU'/, A( 8,2)/'ST  '/, A( 8,3)/'   '/,
+   A( 9,1)/'SEPT'/, A( 9,2)/'EMBE'/, A( 9,3)/'R   '/,
+   A(10,1)/'OCTO'/, A(10,2)/'BER '/, A(10,3)/'   '/,
+   A(11,1)/'NOVE'/, A(11,2)/'MBER'/, A(11,3)/'   '/,
+   A(12,1)/'DECE'/, A(12,2)/'MBER'/, A(12,3)/'   '/,
+   A(13,1)/'**ER'/, A(13,2)/'ROR*'/, A(13,3)/*   '/
IMTH=MTH+0.000001
IF (IMTH .LT. 1 .OR. IMTH .GT. 12) IMTH=13
DO 1 I=1,3
1 MONTH(I)=A(IMTH,I)
RETURN
END
```

## The MTHNAM Subroutine Written in COBOL

This is a COBOL version of the MTHNAM subroutine. The fields are:

**MONTH-TABLE**

Is a field containing the names of the months and the error message.

**MLINE**

Is a 13-element array that redefines the MONTH-TABLE field. Each element (called A) contains the name of a month; the last element contains the error message.

**A**

Is one element in the MLINE array.

**IX**

Is an integer field that indexes MLINE.

**IMTH**

Is the integer representing the month.

**MTH**

Is the double-precision number passed by FOCUS.

**MONTH**

Is the name of the month passed back to FOCUS.

The program is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MTHNAM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STORAGE SECTION.
    01 MONTH-TABLE.
        05 FILLER PIC X(9) VALUE 'JANUARY  '.
        05 FILLER PIC X(9) VALUE 'FEBRUARY '.
        05 FILLER PIC X(9) VALUE 'MARCH   '.
        05 FILLER PIC X(9) VALUE 'APRIL   '.
        05 FILLER PIC X(9) VALUE 'MAY     '.
        05 FILLER PIC X(9) VALUE 'JUNE    '.
        05 FILLER PIC X(9) VALUE 'JULY    '.
        05 FILLER PIC X(9) VALUE 'AUGUST  '.
        05 FILLER PIC X(9) VALUE 'SEPTEMBER'.
        05 FILLER PIC X(9) VALUE 'OCTOBER '.
        05 FILLER PIC X(9) VALUE 'NOVEMBER '.
        05 FILLER PIC X(9) VALUE 'DECEMBER '.
        05 FILLER PIC X(9) VALUE '**ERROR**'.
    01 MLIST REDEFINES MONTH-TABLE.
        05 MLINE OCCURS 13 TIMES INDEXED BY IX.
            10 A PIC X(9).
    01 IMTH PIC S9(5) COMP.
LINKAGE SECTION.
    01 MTH COMP-2.
    01 MONTH PIC X(9).
PROCEDURE DIVISION USING MTH, MONTH.
BEG-1.
    ADD 0.000001 TO MTH.
    MOVE MTH TO IMTH.
    IF IMTH < +1 OR > 12
        SET IX TO +13
    ELSE
        SET IX TO IMTH.
    MOVE A (IX) TO MONTH.
    GOBACK.
```

## The MTHNAM Subroutine Written in PL/I

This is a PL/I version of the MTHNAM subroutine. The fields are:

**MTHNUM**

Is the double-precision number passed by FOCUS.

**FULLMTH**

Is the name of the month passed back to FOCUS.

**MONTHNUM**

Is the integer representing the month.

**MONTH\_TABLE**

A 13-element array containing the names of the months. The last element contains the error message.

The program is:

```
MTHNAM:  PROC(MTHNUM,FULLMTH) OPTIONS(COBOL);
DECLARE  MTHNUM  DECIMAL FLOAT (16) ;
DECLARE  FULLMTH CHARACTER (9) ;
DECLARE  MONTHNUM FIXED BIN (15,0)  STATIC ;
DECLARE  MONTH_TABLE(13) CHARACTER (9)  STATIC
          INIT ('JANUARY',
               'FEBRUARY',
               'MARCH',
               'APRIL',
               'MAY',
               'JUNE',
               'JULY',
               'AUGUST',
               'SEPTEMBER',
               'OCTOBER',
               'NOVEMBER',
               'DECEMBER',
               '**ERROR**') ;

MONTHNUM = MTHNUM + 0.00001 ;
IF MONTHNUM < 1  MONTHNUM > 12 THEN
    MONTHNUM = 13 ;
FULLMTH = MONTH_TABLE(MONTHNUM) ;
RETURN;
END MTHNAM;
```

## The MTHNAM Subroutine Written in BAL Assembler

This is a BAL Assembler version of the MTHNAM subroutine.

```
START 0
STM 14,12,12(13)    save registers
BALR 12,0           load base reg
USING *,12

*
L 3,0(0,1)         load addr of first arg into R3
LD 4,=D'0.0'       clear out FPR4 and FPR5
LE 6,0(0,3)        FP number in FPR6
LPER 4,6           abs value in FPR4
AW 4,=D'0.00001'   add rounding constant
AW 4,DZERO         shift out fraction
STD 4,FPNUM        move to memory
L 2,FPNUM+4        integer part in R2
TM 0(3),B'10000000' check sign of original no
BNO POS           branch if positive
LCR 2,2           complement if negative

*
POS LR 3,2         copy month number into R3
C 2,=F'0'         is it zero or less?
BNP INVALID      yes. so invalid
C 2,=F'12'        is it greater than 12?
BNP VALID        no. so valid
INVALID LA 3,13(0,0) set R3 to point to item @13 (error)
*
VALID SR 2,2      clear out R2
M 2,=F'9'        multiply by shift in table

*
LA 6,MTH(3)      get addr of item in R6
L 4,4(0,1)       get addr of second arg in R4
MVC 0(9,4),0(6)  move in text

*
LM 14,12,12(13)  recover regs
BR 14           return

*
```



```
          DS   0D                alignment
FPNUM    DS   D                floating point number
DZERO    DC   X'4E00000000000000' shift constant
MTH      DC   CL9'dummyitem'    month table
          DC   CL9'JANUARY'
          DC   CL9'FEBRUARY'
          DC   CL9'MARCH'
          DC   CL9'APRIL'
          DC   CL9'MAY'
          DC   CL9'JUNE'
          DC   CL9'JULY'
          DC   CL9'AUGUST'
          DC   CL9'SEPTEMBER'
          DC   CL9'OCTOBER'
          DC   CL9'NOVEMBER'
          DC   CL9'DECEMBER'
          DC   CL9'**ERROR**'
        END   MTHNAM
```

## The MTHNAM Subroutine Written in C

This is a C language version of the MTHNAM subroutine.

```
void mthnam(double *,char *);
void mthnam(mth,month)
double *mth;
char *month;
{
char *nmonth[13] = {"January  ",
                   "February ",
                   "March    ",
                   "April   ",
                   "May     ",
                   "June    ",
                   "July    ",
                   "August  ",
                   "September",
                   "October ",
                   "November ",
                   "December ",
                   "***Error**"};

int imth, loop;
imth = *mth + .00001;
imth = (imth < 1 || imth > 12 ? 13 : imth);
for (loop=0;loop < 9;loop++)
    month[loop] = nmonth[imth-1][loop];
}
```

## The MTHNAM Subroutine Called by a FOCUS Request

The following example demonstrates how a FOCUS request uses the MTHNAM subroutine. The DEFINE command extracts the month portion of the pay date and executes the MTHNAM subroutine to convert it into the full name of the month. The name is stored in the PAY\_MONTH field. The report request prints the monthly pay of Alfred Stevens.

The request is as follows:

```
DEFINE FILE EMPLOYEE
MONTH_NUM/M = PAY_DATE;
PAY_MONTH/A12 = MTHNAM (MONTH_NUM, PAY_MONTH);
END
TABLE FILE EMPLOYEE
PRINT PAY_MONTH GROSS
BY EMP_ID BY FIRST NAME BY LAST_NAME
BY PAY_DATE
IF LN IS STEVENS
END
```

This request produces the following report:

PAGE 1

EMP_ID	FIRST NAME	LAST_NAME	PAY_DATE	PAY_MONTH	GROSS
071382660	ALFRED	STEVENS	81/11/30	NOVEMBER	\$833.33
			81/12/31	DECEMBER	\$833.33
			82/01/29	JANUARY	\$916.67
			82/02/26	FEBRUARY	\$916.67
			82/03/31	MARCH	\$916.67
			82/04/30	APRIL	\$916.67
			82/05/28	MAY	\$916.67
			82/06/30	JUNE	\$916.67
			82/07/30	JULY	\$916.67
			82/08/31	AUGUST	\$916.67

## Subroutines Written in REXX

A FOCUS request can call user-written subroutines coded in REXX. These routines, also called FUSREXX macros, provide a 4GL option to the languages supported for user-written subroutines.

### Using REXX Subroutines

REXX subroutines are supported in the VM/CMS and MVS environments:

- In CMS, a FUSREXX macro can contain either REXX source code or compiled REXX code created by running the source code through the REXX compiler. In addition, you can load either type of FUSREXX macro into memory using the EXECLOAD command. The compilation and load process reduces the CPU requirements and increases speed. Compilation also is a security tool, making private information difficult to read.
- In MVS, FOCUS supports source versions of REXX subroutines only.

Because of CPU requirements, the use of FUSREXX routines in large production jobs should be monitored carefully.

The following notes apply to the examples in this topic:

- REXX versions are not necessarily the same in all operating environments. Therefore, some of the examples may use REXX functions that are not available in your environment.
- The REXX code is listed, but not fully explained. See your REXX documentation for information about REXX instructions and functions.

### Syntax

#### How to Call a REXX User-Written Subroutine

In a DEFINE FILE command:

```
DEFINE FILE filename  
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);  
END
```

In a DEFINE attribute in the Master File:

```
DEFINE fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
```

In a COMPUTE command:

```
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
```

In a Dialogue Manager -SET command:

```
-SET &var = subname(inlen1, inparm1, ..., outlen, outparm);
```

where:

*fieldname*

Is the name of the field to receive the return value.

*An|In*

Is the format of the field to receive return value.

*subname*

Is the name of the REXX routine.

*inlen1, inparm1 ...*

Are the input parameters. Each parameter consists of a pair of values: a length and an alphanumeric parameter value. You can supply the name of an alphanumeric field, an alphanumeric literal, or an expression that resolves to an alphanumeric value. Up to 13 input parameter pairs are supported by FOCUS. Each parameter value can be up to 256 bytes long.

**Note:** Dialogue Manager converts input parameters that consist of numeric digits to decimal format, regardless of their original data type. Therefore, you cannot pass numeric input parameters to a REXX routine using `-SET`.

*outlen, outparm*

Is the output parameter pair, consisting of a length and a return value. In most cases, the return value should be alphanumeric, but integer return values are also supported. The return value can be the name of the field or Dialogue Manager variable to which the value is returned or its USAGE format enclosed in single quotation marks. The return value can be a minimum of one byte long and a maximum (for an alphanumeric value) of 256 bytes.

**Note:** If the value returned is integer, *outlen* must be 4 because FOCUS reserves four bytes for integer fields.

*&var*

Is the name of the Dialogue Manager variable to receive the return value.

REXX subroutines:

- Require input data to be character and should return character output. Integer return values are also supported, but the output length in the subroutine call must be four. FOCUS has a 256-byte limit on character variables. This limit also applies to FUSREXX routines. FUSREXX routines return variable length data. For this reason, you must supply the length of the input arguments and the maximum length of the output data.
- Do *not* require any input parameters, but *do* require one return parameter, which *must* return at least one byte of data. It is possible for a FUSREXX function to need no input, such as a function that returns USERID.

- Do not support floating-point numbers (REXX does not have native floating-point conversion routines). All numeric fields should be converted to character format with no commas using a FOCUS function such as EDIT before being passed to the FUSREXX routine. This prevents FOCUS from converting numbers to floating point before passing them to the FUSREXX routine.
- Are not supported in Dialogue Manager -CMS RUN commands.
- On VM/CMS, the FILETYPE of REXX user-written functions is FUSREXX; they can be stored on any accessed disk.
- On MVS, DDNAME FUSREXX must be allocated to a PDS, and that library will be searched before other MVS libraries.
- The search order for subroutines is:
  1. FUSREXX
  2. Standard CMS or MVS search order.

### Example

### Returning the Day of the Week

The FUSREXX routine DOW returns the day of the week an employee was hired. The routine passes one input parameter pair and one return field pair.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. DAY_OF_WEEK/A9 WITH AHDT= DOW(6,AHDT,9,DAY_OF_WEEK) ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAY_OF_WEEK
END
```

1. The input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The return field is up to nine bytes long and is named DAY\_OF\_WEEK.

The output is:

LAST_NAME	HIRE_DATE	DAY_OF_WEEK
-----	-----	-----
STEVENS	80/06/02	Monday
SMITH	81/07/01	Wednesday
JONES	82/05/01	Saturday
SMITH	82/01/04	Monday
BANNING	82/08/01	Sunday
IRVING	82/01/04	Monday
ROMANS	82/07/01	Thursday
MCCOY	81/07/01	Wednesday
BLACKWOOD	82/04/01	Thursday
MCKNIGHT	82/02/02	Tuesday
GREENSPAN	82/04/01	Thursday
CROSS	81/11/02	Monday

The FUSREXX macro is displayed below. The FUSREXX routine reads the input date, reformats it to MM/DD/YY format, and returns the day of the week using a REXX DATE call.

```
/* DOW routine. Return WEEKDAY from YMMDD format date */
Arg ymd .
Return Date('W',Translate('34/56/12',ymd,'123456'),'U')
```

## Example

### Returning Text Format

The REXX function called in this request returns the number of copies of each classic movie in text format. It passes one input parameter and one return field.

```
TABLE FILE MOVIES
PRINT TITLE AND COMPUTE
1. ACOPIES/A3 = EDIT(COPIES); AS 'COPIES'
AND COMPUTE
2. TXTCOPIES/A8 = NUMCNT(3,ACOPIES,8,TXTCOPIES);
WHERE CATEGORY EQ 'CLASSIC'
END
```

1. The input field is 3 bytes long. Data is passed in field ACOPIES. The COPIES field is converted to an alphanumeric field.
2. The return field is up to 8 bytes long and is named TXTCOPIES.

The output is:

TITLE	COPIES	TXTCOPIES
-----	-----	-----
EAST OF EDEN	001	One
CITIZEN KANE	003	Three
CYRANO DE BERGERAC	001	One
MARTY	001	One
MALTESE FALCON, THE	002	Two
GONE WITH THE WIND	003	Three
ON THE WATERFRONT	002	Two
MUTINY ON THE BOUNTY	002	Two
PHILADELPHIA STORY, THE	002	Two
CAT ON A HOT TIN ROOF	002	Two
CASABLANCA	002	Two

The FUSREXX macro is:

```
/* NUMCNT routine. Pass a number from 0 to 10 and return a character value
*/
Arg numbr .
data = 'Zero One Two Three Four Five Six Seven Eight Nine Ten'
numbr = numbr + 1 /* so 0 equals 1 element in array */
Return Word(data,numbr)
```

## Example

### Passing Multiple Arguments

The following example shows how to pass multiple arguments to a FUSREXX routine. It is an interest calculation using the present salary for the employee and the employee start date to calculate a present value. It passes four input parameters and one return field.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. ACSAL/A12 = EDIT(CURR_SAL) ;
3. DCSAL/D12.2 = CURR_SAL ;
4. PV/A12 = INTEREST(6,AHDT,6,'&YMD',3,'6.5',12,ACSal,12,PV) ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE DCSAL PV
END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The current salary is converted to an alphanumeric field for use in the interest calculation.
3. The current salary is converted to a double-precision field to include commas and a decimal point in the output.

4. The second input field is six bytes long. Data is passed as a FOCUS character variable &YMD in YYMMDD format.

The third input field is a character value of 6.5, which is 3 bytes long to account for the decimal point in the character string.

The fourth input field is 12 bytes long. This passes the character field ACSAL.

The return field is up to 12 bytes long and is named PV.

The output is:

LAST_NAME	FIRST_NAME	HIRE_DATE	DCSAL	PV
-----	-----	-----	-----	---
STEVENS	ALFRED	80/06/02	11,000.00	14055.14
SMITH	MARY	81/07/01	13,200.00	15939.99
JONES	DIANE	82/05/01	18,480.00	21315.54
SMITH	RICHARD	82/01/04	9,500.00	11155.60
BANNING	JOHN	82/08/01	29,700.00	33770.53
IRVING	JOAN	82/01/04	26,862.00	31543.35
ROMANS	ANTHONY	82/07/01	21,120.00	24131.19
MCCOY	JOHN	81/07/01	18,480.00	22315.99
BLACKWOOD	ROSEMARIE	82/04/01	21,780.00	25238.25
MCKNIGHT	ROGER	82/02/02	16,100.00	18822.66
GREENSPAN	MARY	82/04/01	9,000.00	10429.03
CROSS	BARBARA	81/11/02	27,062.00	32081.82

The FUSREXX macro is displayed below. The REXX format command is used to format the return value.

```
/* Simple INTEREST program. dates are yymmdd format */
Arg start_date,now_date,percent,open_balance, .

begin = Date('B',Translate('34/56/12',start_date,'123456'),'U')
stop   = Date('B',Translate('34/56/12',now_date,'123456'),'U')
valnow = open_balance * ((stop - begin) * (percent / 100)) / 365)

Return Format(valnow,9,2)
```



**Example**

**Accepting Multiple Tokens in Parameters**

FUSREXX routines can accept multiple tokens in a parameter. The following procedure passes employee information (pay date and monthly gross pay) as separate tokens in the first parameter. It passes three input parameters and one return field.

```
DEFINE FILE EMPLOYEE
1. COMPID/A256 = FN | ' ' | LN | ' ' | DPT | ' ' | EID ;
2. APD/A6 = EDIT(PAY_DATE) ;
3. APAY/A12 = EDIT(MO_PAY) ;
4. OK4RAISE/A1 = OK4RAISE(256,COMPID,6,APD,12,APAY,1,OK4RAISE) ;
END

TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME DEPARTMENT
IF OK4RAISE EQ '1'
END
```

1. The first input field is 256 bytes long. Data is passed in field COMPID. COMPID is the concatenation of several character fields passed as the first parameter. Each of the other parameters is a single argument.
2. The second input field is six bytes long. Data is passed in field APD. The pay date is converted to an alphanumeric field.
3. The third input field is 12 bytes long. Data is passed in field APAY. The monthly gross pay is converted to an alphanumeric field.
4. The return field is up to one byte long and is named OK4RAISE.

The output is:

EMP_ID	FIRST_NAME	LAST_NAME	DEPARTMENT
071382660	ALFRED	STEVENS	PRODUCTION

The FUSREXX macro is displayed below. Commas separate FUSREXX parameters. The ARG command specifies multiple variable names before the first comma and, therefore, separates the first FUSREXX parameter into separate REXX variables, using blanks as delimiters between the variables.

```
/* OK4RAISE routine. Parse separate tokens in the 1st parm, then more parms
*/

Arg fname lname dept empid, pay_date, gross_pay, .

If dept = 'PRODUCTION' & pay_date < '820000'
Then retvalue = '1'
Else retvalue = '0'

Return retvalue
```

FUSREXX routines *should* use the REXX RETURN function to return data to FOCUS. REXX EXIT is acceptable, but is generally used to end an EXEC, not a FUNCTION.

Correct	Not as Clear
/* Some FUSREXX function */	/* Another FUSREXX function */
Arg input	Arg input
some rexx process ...	some rexx process ...
Return data_to_Focus	Exit 0

## Example

### Returning an Integer Value

It is possible for REXX to return a value that is *not* character format. The following example shows how REXX returns an integer value. This example also shows how the format of the integer field is used as the last field in the return argument. It passes two input fields and one return field. The FUSREXX routine NUMDAYS returns the number of days between hire date and date of increase. Note that the return value for an integer is *always* four bytes long.

```

DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. ADI/A6 = EDIT(DAT_INC) ;
3. BETWEEN/I6 = NUMDAYS(6,AHDT,6,ADI,4,'I6') ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAT_INC BETWEEN
IF BETWEEN NE 0
END

```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The second input field is six bytes long. Data is passed in field ADI. The date of increase is converted to an alphanumeric field.
3. The return field is up to six bytes long and is named BETWEEN.

The output is:

LAST_NAME	HIRE_DATE	DAT_INC	BETWEEN
STEVENS	80/06/02	82/01/01	578
STEVENS	80/06/02	81/01/01	213
SMITH	81/07/01	82/01/01	184
JONES	82/05/01	82/06/01	31
SMITH	82/01/04	82/05/14	130
IRVING	82/01/04	82/05/14	130
MCCOY	81/07/01	82/01/01	184
MCKNIGHT	82/02/02	82/05/14	101
GREENSPAN	82/04/01	82/06/11	71
CROSS	81/11/02	82/04/09	158

The FUSREXX macro is displayed below. The return value is converted from REXX character to HEX and formatted to be four bytes long.

```
/* NUMDAYS routine. Return number of days between 2 dates in yymmdd format
*/
/* The value returned will be in hex format
*/

Arg first,second .

base1 = Date('B',Translate('34/56/12',first,'123456'),'U')
base2 = Date('B',Translate('34/56/12',second,'123456'),'U')

Return D2C(base2 - base1,4)
```

## Example

### Returning a Date Field From a FUSREXX Macro

FOCUS smart date fields contain the integer number of days since the base date 12/31/1900. REXX has a date function that can accept and return several types of date formats, including one called Base format ('B') that contains the number of days since the REXX base date 01/01/0001 (Jan. 1 of the Year 1).

Because input arguments must be alphanumeric, you cannot pass a smart date field to a REXX subroutine. Therefore, you can either:

- Pass the REXX routine an alphanumeric field with date display options and have it return a smart date value, if you account for the number of days difference between the FOCUS base date and the REXX base date and convert the result to integer.
- Pass the REXX routine a smart date value converted to alphanumeric format. With this technique, you must account for the difference in base dates for both the input and output.

The following example uses the technique of passing the subroutine an alphanumeric field with date display options. The FUSREXX macro called DATEREX1 takes two input arguments: an alphanumeric date in A8YYMD format and a number of days in character format. It returns a smart date in YYMD format that represents the input date plus the number of days. The FOCUS format A8YYMD corresponds to the REXX Standard format ('S').

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX1 routine. Add indate (format A8YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate,'S')+ days - 693959, 4)
```

The following request uses the DATEREX1 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE\_DATE is in I6YMD format, it must be converted to A8YYMD before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD = HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8YYMD= ADATE; NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX1(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The output is:

LAST_NAME	FIRST_NAME	HIRE_DATE	NEXT_DATE
-----	-----	-----	-----
BANNING	JOHN	82/08/01	1983/08/01
BLACKWOOD	ROSEMARIE	82/04/01	1983/04/01
CROSS	BARBARA	81/11/02	1982/11/02
GREENSPAN	MARY	82/04/01	1983/04/01
IRVING	JOAN	82/01/04	1983/01/04
JONES	DIANE	82/05/01	1983/05/01
MCCOY	JOHN	81/07/01	1982/07/01
MCKNIGHT	ROGER	82/02/02	1983/02/02
ROMANS	ANTHONY	82/07/01	1983/07/01
SMITH	MARY	81/07/01	1982/07/01
SMITH	RICHARD	82/01/04	1983/01/04
STEVENS	ALFRED	80/06/02	1981/06/02

The following example uses the technique of passing the subroutine a smart date converted to alphanumeric format. The FUSREXX macro called DATEREX2 takes two input arguments: an alphanumeric number of days that represents a smart date, and a number of days to add. It returns a smart date in YYMD format that represents the input date plus the number of days. Both the input date and output date are in REXX base date ('B') format.

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX2 routine. Add indate (original format YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate+693959,'B') + days - 693959, 4)
```

The following request uses the DATEREX2 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE\_DATE is in I6YMD format, it must be converted to an alphanumeric number of days before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD = HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8 = EDIT(ADATE); NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX2(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The report output is the same as that produced by the DATEREX1 macro.

## Compiling FUSREXX Macros in CMS

The SUM2 FUSREXX macro takes two amounts as input and returns the sum in integer format:

```
/* SUM2 routine. Add amount1 to amount2 and return as integer */
Arg amt1, amt2 .
Return D2C(amt1 + amt2,4)
```

To compile and compress this FUSREXX macro in CMS, issue the following command. Note that the file identifier must be in upper case:

```
rexcomp SUM2 FUSREXX A (condense
```

A FILELIST of SUM2 \* A lists the following files:

SUM2	CFUSREXX	A1	F	1024	2	1	1/31/00	12:07:19
SUM2	LISTING	A1	V	121	42	1	1/31/00	12:07:19
SUM2	FUSREXX	A1	F	80	3	1	1/31/00	12:04:19

The file SUM2 FUSREXX is the original source file. The file SUM2 CFUSREXX is the compiled version. To call the compiled version in a FOCUS request, you must rename it to have the file type FUSREXX. The file SUM2 LISTING details the results of the compilation.

To use the compiled version in a FOCUS request, issue the following commands. The EXECLOAD command, which loads the routine into memory and improves performance, is optional:

```
rename sum2 fusrexx a ssum2 fusrexx a
rename sum2 cfusrexx a sum2 fusrexx a
execload sum2 fusrexx a
```

Then, in FOCUS, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT CSAL AND COMPUTE
ASAL/A12 = EDIT(CSAL);
AMOUNT/A4 = '1000';
TOTSAL/I6 = SUM2(12, ASAL, 4, AMOUNT, 4, TOTSAL);
END
```

The output is:

CURR_SAL	ASAL	AMOUNT	TOTSAL
-----	----	-----	-----
\$11,000.00	000000011000	1000	12000
\$13,200.00	000000013200	1000	14200
\$18,480.00	000000018480	1000	19480
\$9,500.00	000000009500	1000	10500
\$29,700.00	000000029700	1000	30700
\$26,862.00	000000026862	1000	27862
\$21,120.00	000000021120	1000	22120
\$18,480.00	000000018480	1000	19480
\$21,780.00	000000021780	1000	22780
\$16,100.00	000000016100	1000	17100
\$9,000.00	000000009000	1000	10000
\$27,062.00	000000027062	1000	28062

# Index

## Symbols

- "... " command, 4-10, 4-96
- & (local variables), 4-6
- && (global variables), 4-6
- &ACCEPTS variable, 4-60
- &BASEIO variable, 4-60
- &CHNGD variable, 4-60
- &CURSOR variable, 4-62
- &CURSORAT variable, 4-62
- &DATE variable, 4-54
- &DATEfmt variable, 4-54, 7-31
- &DELTD variable, 4-60
- &DMY variable, 4-54
- &DMYY variable, 4-54, 4-58, 7-31
- &DUPLS variable, 4-60
- &ECHO variable, 4-42, 4-62
- &FOCCPU variable, 4-54
- &FOCDISORG variable, 2-11, 4-60
- &FOCERRNUM variable, 4-60
- &FOCEXTTRM variable, 4-54
- &FOCFIELDNAME variable, 4-54
- &FOCFOCEXEC variable, 4-55
- &FOCINCLUDE variable, 4-55
- &FOCMODE variable, 4-55
- &FOCPRINT variable, 4-55
- &FOCPUTLVL variable, 4-55
- &FOCQUALCHAR variable, 4-55
- &FOCREL variable, 4-55
- &FOCSBORDER variable, 4-55
- &FOCSYSTYP variable, 4-55
- &FOCTMPDSK variable, 4-56
- &FOCTRMSD variable, 4-56
- &FOCTRMSW variable, 4-56
- &FOCTRMTYP variable, 4-56
- &FOCTTIME variable, 4-56
- &FOCVTIME variable, 4-56
- &FORMAT variable, 4-60
- &HIPERFOCUS variable, 4-56
- &INPUT variable, 4-60
- &INVALID variable, 4-60
- &IORETURN variable, 4-56
- &LINES variable, 4-61
- &MDY variable, 4-56
- &MDYY variable, 4-56, 4-58, 7-31
- &NOMATCH variable, 4-61
- &PFKEY variable, 4-62, 9-26
- &QUIT variable, 4-44, 4-62
- &READS variable, 4-61
- &RECORDS variable, 4-61
- &REJECTS variable, 4-61
- &RETCODE variable, 4-24, 4-56
- &STACK variable, 4-43, 4-62
- &TOD variable, 4-56
- &TRANS variable, 4-61
- &WINDOWNAME variable, 4-62, 9-26
- &WINDOWVALUE variable, 4-62, 9-26

&YMD variable, 4-56  
&YYMD variable, 4-56, 4-58, 7-31  
(&& global variables), 2-25  
-\* command, 4-8, 4-9, 4-80  
.EVAL operator, 4-63  
.EXIST operator, 4-21  
.LENGTH operator, 4-22  
.TYPE operator, 4-23  
? && command, 2-25  
-? &[string] command, 4-10, 4-51  
? COMBINE command, 2-3  
-? command, 4-80  
? DEFINE query command, 2-4  
? EUROFILE command, 2-5, 8-9  
? F command, 2-5  
? FDT command, 2-6  
? FF command, 2-8  
? FILE command, 2-9  
? HOLD command, 2-12  
? JOIN command, 2-13  
? LANG command, 2-14  
? LET command, 2-14  
? LOAD command, 2-15, 6-6  
? n command, 2-15  
? PTF command, 2-16  
? RELEASE command, 2-17  
? SET ALL query command, 8-9  
? SET command, 2-17, 7-12  
? SET FOR query command, 2-17, 2-19  
? SET GRAPH command, 2-21

? SET NOT query command, 2-17, 2-19, 2-20  
-? SET SETCOMMAND &myvar, 4-10  
? STAT command, 2-22  
? STYLE command, 2-26  
? SU command, 2-24  
? USE command, 2-26

## A

ABS function, 3-36  
ACCBLN parameter, 1-3  
ACCEPT attribute, 1-24  
Access Files, 6-4  
    loading, 6-4  
    VIDEOTR2, A-27  
AGGR[RATIO] parameter, 1-3  
ALL parameter, 1-4  
ALLOWCVTERR parameter, 1-5  
alphanumeric format, 3-121, 3-123, 3-168  
    converting, 3-121, 3-168  
alphanumeric strings  
    converting from, 3-121  
amper variables, 4-49  
    EDIT function, 4-30  
    return values, 9-24  
applications, 9-51  
    executing, 9-51  
ARGLEN subroutine, 3-37  
arguments, 3-36, C-3  
    calculating square root, 3-163  
    calculating value, 3-36  
    in scientific notation, 3-96  
    in subroutines, 3-19, 3-20  
    length, 3-37  
    maximum value, 3-144  
    minimum value, 3-144  
    returning logarithm, 3-143



- 
- AS phrase, 1-6
  - ASIS function, 3-38
  - ASNAMES parameter, 1-6
  - Assembler language, C-4
  - ATODBL subroutine, 3-39
  - AUTOINDEX parameter, 1-6
  - AUTOPATH parameter, 1-7
  - AUTOSTRATEGY parameter, 1-7
  - AUTOTABLEF parameter, 1-7
  - AYM subroutine, 3-43
  - AYMD subroutine, 3-45
  - B**
  - BAL Assembler language, C-19
    - MTHNAM subroutine, C-19
  - bar charts, 3-47
  - BAR subroutine, 3-47
  - BINS parameter, 1-8
  - bit functions and subroutines, 3-3
    - BITSON, 3-49
    - BITVAL, 3-50
    - BYTVAL, 3-52
    - HEXBYT, 3-117
    - UFMT, 3-168
  - bits, 3-3
    - evaluating, 3-49
    - strings, 3-50
  - BITSON subroutine, 3-49
  - BITVAL subroutine, 3-50
  - BLKCALC parameter, 1-8
  - BOTTOMMARGIN parameter, 1-9
  - branching, 4-16
    - IF command, 3-24
  - BUSDAYS parameter, 1-8
    - BUSDAYS setting, 3-15
    - business day units, 3-15
  - BYPANEL parameter, 1-9
  - BYSCROLL parameter, 1-9
  - BYTVAL subroutine, 3-52
  - C**
  - C language, C-4
    - MTHNAM subroutine, C-20
  - Cache memory
    - Number of reads performed, 2-24
  - CACHE parameter, 1-10
    - calculated values, 7-26
      - sliding window, 7-26
  - CAR data source, A-11
  - CARTESIAN parameter, 1-11
  - CDN parameter, 1-11
  - character functions and subroutines, 3-4
    - ARGLEN, 3-37
    - ASIS, 3-38
    - BITSON, 3-49
    - BITVAL, 3-50
    - BYTVAL, 3-52
    - CHKFMT, 3-56
    - CTRAN, 3-62
    - CTRFLD, 3-68
    - EDIT, 3-93, 4-30
    - GETTOK, 3-106
    - LCWORD, 3-138
    - LJUST, 3-140
    - LOCASE, 3-142
    - OVLAY, 3-149
    - PARAG, 3-152
    - POSIT, 3-156
    - RJUST, 3-161
    - SUBSTR, 3-164
    - UPCASE, 3-170
  - character strings
    - converting, 3-138, 3-142, 3-170

- character strings (*continued*)
  - dividing, 3-106
  - extracting substring, 3-164
  - justifying, 3-140, 3-161
  - overlying substring, 3-149
- characters, 3-4
  - converting, 3-117
  - translating, 3-62
- CHECK FILE command, 7-10
- CHGDAT subroutine, 3-53
- CHKFMT subroutine, 3-56
- CHKPCK subroutine, 3-59
- CLOSE command, 4-9, 4-81
- CLOSE ddname command, 4-9
- CMS command, 4-9, 4-81
- CMS environment, C-12
  - compiling subroutines, C-12
  - FUSREXX macros, C-32
  - storing subroutines, C-12
- CMS RUN command, 4-9
  - subroutines, 3-26
- COBOL language, C-4
  - MTHNAM subroutine, C-16
- COLUMNSCROLL parameter, 1-12
- COMASTER data source, A-21
- COMBINE structures, 2-3
  - ? COMBINE command, 2-3
- command statistics, 2-22, 2-24
  - ? STAT command, 2-22
- commands, 1-2
  - CHECK FILE, 7-10
  - COMPILE, 6-7
  - COMPUTE, 7-26
  - DEFINE, 7-5, 7-20
  - Dialogue Manager, 4-9
  - EXEC, 4-7, 4-37, 4-41
  - GLOBAL, C-13
  - LET, 5-2
  - commands (*continued*)
    - LET CLEAR, 4-53
    - LET ECHO, 5-10
    - LOAD, 6-2
    - LOAD MODIFY, 6-6
    - MINIO, 6-8, 6-9, 6-11
    - query, 2-2
    - QUIT, 4-44
    - SET, 1-2
    - SET EUROFILE, 8-8
    - SET TESTDATE, 7-10
    - WINDOW COMPILE, 9-83
    - WINDOW PAINT, 9-52
- comments, 4-8
- COMPILE command, 6-7
- compound -IF tests, 4-19
- COMPUTE command, 7-26
  - sliding window, 7-26
  - subroutines in, 3-22
- COMPUTE parameter, 1-12
- concatenation, 4-65
- conversions
  - alphanumeric format, 3-168
  - character strings, 3-138, 3-142, 3-170
  - characters, 3-117
  - currency, 8-2
  - dates, 3-53, 3-77, 7-32
  - date-time fields, 3-113, 3-114
  - format, 3-39
  - from alphanumeric format, 3-91
  - from Gregorian format, 3-135
  - from Julian format, 3-109
  - from numeric format, 3-89, 3-100
  - numeric format, 3-130, 3-132, 3-134
  - strings, 3-121
  - to alphanumeric format, 3-100
  - to date format, 3-89, 3-114
  - to EBCDIC, 3-52
  - to Gregorian format, 3-109
  - to Julian format, 3-135
  - to numeric format, 3-91
  - to zone format, 3-134

- COUNTWIDTH parameter, 1-12
- COURSES data source, A-16
- cross-century dates, 7-1, 7-31
- CRTCLEAR command, 4-9, 4-81
- CRTFORM command, 4-10, 4-49, 4-75, 4-82
- CTTRAN subroutine, 3-62
- CTRFLD subroutine, 3-68
- currencies, 8-2
  - converting, 8-2
  - euro, 8-2
- CURRENCY attribute, 8-6
- currency conversions, 8-2, 8-10
  - currency data source, 8-4
- currency data source, 8-4
  - ? EUROFILE command, 2-5
  - activating, 8-8
  - creating, 8-4
  - CURRENCY attribute, 8-6
  - displaying, 2-5
  - querying, 8-9
- custom subroutines, C-14
- ## D
- data source functions and subroutines, 3-6
  - LAST, 3-137
- data source statistics, 2-9
  - ? FILE command, 2-9
- data sources
  - currency data source, 8-4
  - MINIO command, 6-8
  - statistics, 2-9
  - USE command, 2-26
- date conversions, 3-53, 3-77
- date formats, 7-5
  - with sliding window, 7-5
- date functions and subroutines, 3-6, 7-31
  - AYM, 3-43
  - AYMD, 3-45
  - CHGDAT, 3-53
  - DATEADD, 3-72
  - DATECVT, 3-77
  - DATEDIF, 3-78
  - DATEMOV, 3-80
  - DMY, 3-86
  - DOWK, 3-88
  - DOWKL, 3-88
  - DTDY, 3-89
  - DTDYM, 3-89
  - DTMDY, 3-89
  - DTMYD, 3-89
  - DTYDM, 3-89
  - DTYMD, 3-89
  - GREGDT, 3-109
  - HADD, 3-112
  - HCVRT, 3-113
  - HDATE, 3-114
  - HDIFF, 3-115
  - HDTTM, 3-116
  - HGETC, 3-119
  - HHMMSS, 3-120
  - HINPUT, 3-121
  - HMIDNT, 3-122
  - HNAME, 3-123
  - HPART, 3-125
  - HSETPT, 3-126
  - HTIME, 3-127
  - JULDAT, 3-135
  - legacy versions, 3-14
  - MDY, 3-86
  - settings, 3-14
  - TODAY, 3-166
  - valid date input, 3-9
  - YM, 3-174
  - YMD, 3-86
- DATEADD function/subroutine, 3-72
- DATECVT function, 3-77
- DATEDIF function, 3-78
- DATEDISPLAY parameter, 1-13

- DATEFNS parameter, 1-13, 3-14
- DATEFORMAT parameter, 1-14
- DATEMOV function, 3-80
- dates, 3-6, 7-2
  - calculating difference, 3-174
  - converting, 3-116, 7-32
  - date display format, 7-31
  - date display options, 4-58, 7-31
  - difference between, 3-86
  - functions and subroutines, 3-78, 3-166
  - holidays, 3-16
  - incrementing, 3-112
  - natural date literals, 4-33
  - setting, 4-33
  - system, 7-31
  - time stamp, 7-32
  - validating, 7-6
- date-time fields, 3-119, 3-122, 3-123, 3-125, 3-126
  - converting, 3-113, 3-127
  - converting to, 3-116
- date-time functions
  - component names, 3-8
- DATETIME parameter, 1-14
- date-time values, 3-121
- DECODE function, 3-83, 4-29
- decoding functions and subroutines
  - DECODE, 3-83, 4-29
- default century, 7-32
- DEFAULTS command, 4-10, 4-68, 4-70, 4-83
- DEFCENT parameter, 1-15, 7-2, 7-3, 7-5, 7-7
  - COMPUTE command, 7-26
  - DEFINE command, 7-20
  - querying, 7-12
  - with COMPUTE, 7-27
- DEFINE command, 7-5
  - sliding window, 7-5, 7-20
  - subroutines in, 3-22
- Dialogue Manager, 4-2
  - ASIS function, 3-38
  - canceling a procedure, 4-15
  - commands, 4-9
  - executing stacked commands, 4-13
  - exiting from FOCUS, 4-15
  - facilities, 4-36
  - leading zeros, 3-17
  - sending a message to the user, 4-11
  - stacked commands, 4-12
  - testing logic, 4-43
  - variables, 4-6
- Dialogue Manager commands, 4-9
  - \* , 4-8, 4-9, 4-80
  - ? , 4-80
  - ? &[string], 4-10, 4-51
  - ? SET COMMAND &myvar, 4-10
  - CLOSE, 4-9, 4-81
  - CLOSE ddname, 4-9
  - CMS, 4-9, 4-81
  - CMS RUN, 4-9
  - CRTCLEAR, 4-9, 4-81
  - CRTFORM, 4-10, 4-49, 4-75, 4-82
  - DEFAULTS, 4-10, 4-70, 4-83
  - EXIT, 4-10, 4-13, 4-83
  - GOTO, 4-10, 4-16, 4-84
  - HTMLFORM, 4-10, 4-84
  - IF, 4-10, 4-85
  - INCLUDE, 4-10, 4-37, 4-38, 4-86
  - label, 4-10, 4-86
  - MVS RUN, 4-10, 4-87
  - PASS, 4-7, 4-10, 4-87
  - PROMPT, 4-10, 4-49, 4-73, 4-88
  - QUIT, 4-10, 4-15, 4-89
  - QUIT FOCUS, 4-15
  - READ, 4-10, 4-72, 4-90
  - REPEAT, 4-10, 4-25, 4-91
  - RUN, 4-10, 4-12, 4-92
  - SET, 3-23, 4-10, 4-27, 4-28, 4-32, 4-44, 4-71, 4-92
  - TSO RUN, 4-10, 4-93
  - TYPE, 4-10, 4-11, 4-93
  - WINDOW, 4-10, 4-49, 4-75, 4-94, 9-3
  - WRITE, 4-10, 4-45, 4-95
- disorganized files, 2-11

DISPLAY parameter, 1-15

DMOD subroutine, 3-128

DMY function, 3-86

DOWK subroutine, 3-88

DOWKL subroutine, 3-88

DT subroutines, 3-89

DTDMY subroutine, 3-89

DTDYM subroutine, 3-89

DTMDY subroutine, 3-89

DTMYD subroutine, 3-89

DTSTRICT parameter, 1-16

DTYDM subroutine, 3-89

DTYMD subroutine, 3-89

dynamic window, 7-4, 7-10

## E

EDIT function, 3-91, 3-93, 4-30

EDUCFILE data source, A-7

EMPDATA data source, A-17

EMPLOYEE data source, A-3

EMPTYREPORT parameter, 1-16

Entry Menu, 9-53

error messages

? n command, 2-15

displaying explanations, 2-15

retrieving, 3-96

ESTRECORDS parameter, 1-17

euro currency, 8-2

converting, 8-2, 8-10

EUROFILE parameter, 1-18, 8-8

EXEC command, 4-7, 4-37, 4-41

execution windows, 9-11

-EXIT command, 4-10, 4-12, 4-13, 4-83

EXP subroutine, 3-95

EXPERSON data source, A-18

EXPN function, 3-96

EXTAGGR parameter, 1-18

external subroutines, 3-30

EXTHOLD parameter, 1-18

EXTSORT parameter, 1-19

EXTTERM parameter, 1-19

## F

FDEFCENT attribute, 7-3, 7-13

FEXERR subroutine, 3-96

field names windows, 9-8

field-level sliding window, 7-13

FIELDNAME parameter, 1-19

fields, 2-5

? F command, 2-5

? FF command, 2-8

file contents windows, 9-9

file directory table, 2-6

? FDT command, 2-6

file names windows, 9-7

creating, 9-49

file-level sliding window, 7-13

FILENAME parameter, 1-20

files, 6-2

disorganization, 2-11

loading, 6-2

writing to, 4-45

FILTER parameter, 1-20

FINANCE data source, A-14

Financial Modeling Language (FML), 3-28

- FINDMEM subroutine, 3-97
  - FIXRETRIEVE parameter, 1-21
  - FMOD subroutine, 3-128
  - FOC144 parameter, 1-21
  - FOC2GIGDB parameter, 1-22
  - FOCALLOC parameter, 1-22
  - FOCCOMP file, 6-5
    - with sliding window, 7-6
  - FOCSTACK parameter, 1-22
  - FOCUS facilities, 6-2
    - accessing FOCUS data sources, 6-8
    - compiling MODIFY requests, 6-7
    - loading files, 6-2
  - format conversion functions and subroutines, 3-11
    - ASIS, 3-38
    - ATODBL, 3-39
    - CHKPCK, 3-59
    - EDIT, 3-91
    - FTOA, 3-100
    - ITONUM, 3-130
    - ITOPACK, 3-132
    - ITOOZ, 3-134
    - PCKOUT, 3-154
    - UFMT, 3-168
  - format conversions, 3-39
  - format specifications, 4-76
  - FORTTRAN language, C-4
    - MTHNAM subroutine, C-15
  - FTOA subroutine, 3-100
  - function keys, 5-12
    - LET substitution, 5-12
    - testing values, 9-26
  - functions and subroutines, 3-1, 3-21
    - bit, 3-3
    - character, 3-4
    - data source, 3-6
    - date, 3-6
    - date-time, 3-119
    - functions and subroutines (*continued*)
      - differences between, 3-2
      - format conversion, 3-11
      - numeric, 3-12
      - system, 3-13
      - types, 3-3
      - with sliding window, 7-5, 7-23, 7-29
  - FUSELIB LOAD library, 3-30
  - FUSREXX macros, C-32
    - compiling in CMS, C-32
  - FYRTHRESH attribute, 7-3, 7-13
- ## G
- GETPDS subroutine, 3-101
  - GETTOK subroutine, 3-106
  - GETUSER subroutine, 3-108
  - GGDEMOG data source, A-29
  - GGORDER data source, A-30
  - GGPRODS data source, A-31
  - GGSALES data source, A-32
  - GGSTORES data source, A-33
  - GLOBAL command, C-13
  - global sliding window, 7-7
  - global variables, 4-6, 4-49, 4-53
    - ? && command, 2-25
  - Gotham Grinds data sources, A-29
    - GGDEMOG, A-29
    - GGORDER, A-30
    - GGPRODS, A-31
    - GGSALES, A-32
    - GGSTORES, A-33
  - GOTO command, 4-10, 4-16, 4-84
  - goto values, 9-3, 9-25, 9-62
  - graph parameters, 2-21
    - ? SET GRAPH command, 2-21
  - GREGDT subroutine, 3-109

**H**

HADD subroutine, 3-112  
 HCNVRT subroutine, 3-113  
 HDATE subroutine, 3-114  
 HDAY parameter, 1-23, 3-16  
 HDIFF subroutine, 3-115  
 HDTTM subroutine, 3-116  
 HEXBYT subroutine, 3-117  
 HGETC subroutine, 3-119  
 HHMMSS function/subroutine, 3-120  
 HINPUT subroutine, 3-121  
 HLISUDUMP parameter, 1-23  
 HLISUTRACE parameter, 1-23  
 HMIDNT subroutine, 3-122  
 HNAME subroutine, 3-123  
 HOLD fields, 2-12  
   ? HOLD command, 2-12  
 HOLDATTR parameter, 1-24  
 HOLDLIST parameter, 1-24  
 HOLDSTAT parameter, 1-25  
 holidays, 3-16  
 horizontal menus, 9-6  
   creating, 9-14  
 HOTMENU parameter, 1-25  
 HPART subroutine, 3-125  
 HSETPT subroutine, 3-126  
 HTIME subroutine, 3-127  
 -HTMLFORM command, 4-10, 4-84

**I**

IBMLE parameter, 1-26

IF command, 3-24  
   subroutines in, 3-24  
 -IF command, 4-10, 4-16, 4-85  
   -IF tests, 4-19  
   subroutines in, 3-24  
 IF selection tests, 3-23  
   subroutines in, 3-23  
 -IF tests, 4-19  
   compound, 4-19  
   operators and functions, 4-20  
   screening values, 4-20  
 IMMEDIATE parameter, 1-26  
 IMOD subroutine, 3-128  
 implied prompting, 4-49, 4-75  
 IMS parameter, 1-26  
 -INCLUDE command, 4-10, 4-37, 4-38, 4-86  
 INDEX parameter, 1-27  
 INT function, 3-129  
 ITONUM subroutine, 3-130  
 ITOPACK subroutine, 3-132  
 ITOZ subroutine, 3-134

**J**

JOBFILE data source, A-6  
 JOINOPT parameter, 1-27  
 joins, 2-13  
   ? JOIN command, 2-13  
 JULDAT subroutine, 3-135

**L**

-label command, 4-10, 4-86  
 LANG parameter, 1-28

- languages, C-5
  - considerations, C-5
  - environment support, 3-32
- LAST function, 3-137
- LCWORD function/subroutine, 3-138
- leading zeros, 3-17
- LEADZERO parameter, 1-29, 3-17
- LEDGER data source, A-13
- LEFTMARGIN parameter, 1-30
- legacy dates, 7-5
  - with sliding window, 7-5
- LET CLEAR command, 4-53
- LET command, 5-2
  - checking substitutions, 5-10
  - clearing substitutions, 5-11
  - long form, 5-2
  - multiple-line substitutions, 5-8
  - null substitutions, 5-7
  - recursive substitutions, 5-8
  - saving substitutions, 5-12
  - short form, 5-2
  - variable substitutions, 5-5
- LET ECHO command, 5-10
- LET substitutions
  - function key, 5-12
- LINES parameter, 1-30
- LJUST function/subroutine, 3-140
- LOAD command, 6-2
- LOAD MODIFY command, 6-6
- load procedures, A-2
- loading files, 2-15, 6-2
  - ? LOAD command, 2-15
  - Access Files, 6-4
  - compiled MODIFY requests, 6-5
  - displaying, 6-6
  - FOCCOMP file, 6-5

- loading files (*continued*)
  - LOAD, 6-2
  - Master Files, 6-4
  - MODIFY requests, 6-6
  - procedures, 6-4
- local variables, 4-6, 4-49, 4-52
- LOCASE subroutine, 3-142
- LOG function, 3-143
- looping, 4-25
  - controlling, 4-32
  - ending, 4-27

## M

- Main Menu, 9-54
- Master Files
  - defining sliding window, 7-13
  - displaying field information, 2-8
  - loading, 6-4
  - samples, A-2
- MASTER parameter, 1-31
- MAX function, 3-144
- MAXLRECL parameter, 1-31
- MDY function, 3-86
- memory
  - cache, 2-24
- menus, 9-4
  - creating, 9-2
  - horizontal, 9-6
  - pull-down, 9-16
  - vertical, 9-5
- merge
  - routine invocations, 2-24
- MESSAGE parameter, 1-31
- MIN function, 3-144
- MINIO command, 6-8, 6-9
  - restrictions, 6-11
  - usage, 6-9



- 
- MINIO parameter, 1-32
  - MOD subroutines, 3-128
  - MODIFY, 6-6
    - compiling, 6-7
    - loading, 6-6
  - MOVIES data source, A-24
  - MTHNAM subroutine, C-14
    - BAL Assembler language, C-19
    - C language, C-20
    - COBOL language, C-16
    - FOCUS requests, C-21
    - FORTRAN language, C-15
    - PL/I language, C-18
  - multi-input windows, 9-12
    - creating, 9-18
  - MULTIPATH parameter, 1-33
  - multiple-line substitutions, 5-8
  - MVS, C-13
    - accessing subroutines, 3-30
    - compiling subroutines, C-13
    - storing subroutines, C-13
  - MVS RUN command, 4-10, 4-87
    - subroutines, 3-26
  - MVSDYNAM subroutine, 3-145
- N**
- naming conventions, C-3
  - National Language Support (NLS), 2-14
    - ? LANG command, 2-14
  - natural date literals, 4-33
  - nesting procedures, 4-40
    - INCLUDE command, 4-40
  - NODATA parameter, 1-33
  - null substitutions, 5-7
  - numeric format, 3-130
    - converting, 3-130, 3-132, 3-134
    - converting to, 3-125
  - numeric functions and subroutines, 3-12
    - ABS, 3-36
    - ASIS, 3-38
    - BAR, 3-47
    - DMOD, 3-128
    - EXP, 3-95
    - EXPN, 3-96
    - FMOD, 3-128
    - IMOD, 3-128
    - INT, 3-129
    - LOG, 3-143
    - MAX, 3-144
    - MIN, 3-144
    - PRDNOR, 3-158
    - PRDUNI, 3-158
    - RDUNIF, 3-158
    - RENORM, 3-158
    - SQRT, 3-163
- O**
- ONLINE-FMT parameter, 1-34
  - ORIENTATION parameter, 1-34
  - OVLAY function/subroutine, 3-149
- P**
- packed decimal fields
    - validating, 3-59
  - PAGE-NUM parameter, 1-35
  - PAGESIZE parameter, 1-35
  - PANEL parameter, 1-38
  - PAPER parameter, 1-39
  - PARAG subroutine, 3-152
  - parameter settings, 2-17
    - ? SET command, 2-17

- parameters, 1-3
    - ALLOWCVTERR, 7-31
    - BUSDAYS, 3-15
    - DATEDISPLAY, 7-31
    - DATEFNS, 3-14
    - DEFCENT, 7-2, 7-7
    - EUROFILE, 8-8
    - HDAY, 3-16
    - LEADZERO, 3-17
    - UNITS, 1-55
    - YRTHRESH, 7-2, 7-7
  - partitioned data sets, 3-97, 3-101
  - PASS command, 4-7, 4-10, 4-87
  - PASS parameter, 1-39
  - passwords, 4-7
  - PAUSE parameter, 1-40
  - PAYHIST data source, A-20
  - PCKOUT subroutine, 3-154
  - PF keys, 9-26
  - PFnn command, 9-26
  - PFnn parameter, 1-40
  - PL/I language, C-4
    - MTHNAM subroutine, C-18
  - POSIT function/subroutine, 3-156
  - positional variables, 4-69
  - PRDNOR subroutines, 3-158
  - PRDUNI subroutines, 3-158
  - PREFIX parameter, 1-41
  - PRINT parameter, 1-41
  - PRINTPLUS parameter, 1-42
  - procedures, 4-2, 4-36, 4-44
    - comments, 4-8
    - creating, 4-6
    - CRTFORM command, 4-49, 4-75
    - debugging, 4-42
    - EXEC command, 4-37, 4-41
    - procedures (*continued*)
      - executing, 4-7, 4-12
      - incorporating multiple, 4-37
      - load, A-2
      - loading, 6-4
      - open-ended, 4-41
      - PROMPT command, 4-49
      - prompting, 4-49
      - SAMPLE, 9-31
      - sample in Window Painter, 9-51
      - security, 4-7, 4-44
      - storing, 4-6
      - TED, 4-6
      - testing, 4-42
      - variables, 4-49
      - WINDOW command, 4-49, 4-75
  - PROD data source, A-10
  - programming, C-5
    - considerations, C-5
    - entry points, C-7
    - subroutines, C-9
  - PROMPT command, 4-10, 4-49, 4-73, 4-88
  - prompting, 4-49
    - CRTFORM command, 4-49, 4-75
    - implied, 4-49, 4-75
    - PROMPT command, 4-49, 4-73
    - WINDOW command, 4-49, 4-75
  - PTFs, 2-16
    - displaying, 2-16
  - pull-down menus, 9-16
    - creating, 9-16
- ## Q
- QUALCHAR parameter, 1-43
  - QUALTITLES parameter, 1-43
  - query commands, 2-2
    - ? &&, 2-25
    - ? &[string], 4-51
    - ? COMBINE, 2-3
    - ? DEFINE, 2-4
    - ? EUROFILE, 2-5, 8-9

query commands (*continued*)

- ? F, 2-5
  - ? FDT, 2-6
  - ? FF, 2-8
  - ? FILE, 2-9
  - ? HOLD, 2-12
  - ? JOIN, 2-13
  - ? LANG, 2-14
  - ? LET, 2-14
  - ? LOAD, 2-15, 6-6
  - ? n, 2-15
  - ? PTF, 2-16
  - ? RELEASE, 2-17
  - ? SET, 2-17, 7-12
  - ? SET FOR, 2-17
  - ? SET GRAPH, 2-21
  - ? SET NOT, 2-17
  - ? STAT, 2-22
  - ? STYLE, 2-26
  - ? SU, 2-24
  - ? USE, 2-26
  - testing status, 4-24
- QUIT command, 4-44
- QUIT command, 4-10, 4-12, 4-15, 4-89
- QUIT FOCUS command, 4-15

**R**

- RDNORM subroutines, 3-158
- RDUNIF subroutines, 3-158
- READ command, 4-10, 4-68, 4-72, 4-90
- REBUILDMSG parameter, 1-44
- RECAP command, 3-28
- subroutines in, 3-28
- RECAP-COUNT parameter, 1-44
- RECORDLIMIT parameter, 1-44
- recursive substitutions, 5-8
- REGION data source, A-15
- REPEAT command, 4-10, 4-25, 4-91

- return value display windows, 9-9
- return values, 9-24
- REXX subroutines, C-22
- RIGHTMARGIN parameter, 1-45
- RJUST function/subroutine, 3-161
- RPAGESET parameter, 1-45
- RUN command, 3-26, 4-10, 4-12, 4-92
- subroutines, 3-26

**S**

- SALES data source, A-8
- sample data sources
- CAR, A-11
  - COMASTER, A-21
  - COURSES, A-16
  - creating, A-2
  - EDUCFILE, A-7
  - EMPDATA, A-17
  - EMPLOYEE, A-3
  - EXPERSON, A-18
  - FINANCE, A-14
  - Gotham Grinds data sources, A-29
  - JOBFILE, A-6
  - LEDGER, A-13
  - MOVIES, A-24
  - PAYHIST, A-20
  - PROD, A-10
  - REGION, A-15
  - SALES, A-8
  - TRAINING, A-19
  - VIDEOTR2, A-26
  - VideoTrk, A-24
- SAVEMATRIX parameter, 1-45
- SBORDER parameter, 1-46
- SCREEN parameter, 1-46
- screens, 9-51
- Entry Menu, 9-53
  - Main Menu, 9-54
  - Utilities Menu, 9-72

screens (*continued*)

- Window Creation Menu, 9-57
- Window Design Screen, 9-59
- Window Options Menu, 9-61

SET command, 1-2

- parameters, 1-3

-SET command, 4-10, 4-27, 4-28, 4-32, 4-33, 4-44, 4-68, 4-71, 4-92

- creating amper variables, 3-23
- looping, 4-32

SET EUROFILE command, 8-8

SET parameters

- ACCBLN, 1-3
- AGGR[RATIO], 1-3
- ALL, 1-4
- ALLOWCVTERR, 1-5
- ASNAMES, 1-6
- AUTOINDEX, 1-6
- AUTOPATH, 1-7
- AUTOSTRATEGY, 1-7
- AUTOTABLEF, 1-7
- BINS, 1-8
- BLKCALC, 1-8
- BOTTOMMARGIN, 1-9
- BUSDAYS, 1-8
- BYPANEL, 1-9
- BYSCROLL, 1-9
- CACHE, 1-10
- CARTESIAN, 1-11
- CDN, 1-11
- COLUMNSCROLL, 1-12
- COMPUTE, 1-12
- COUNTWIDTH, 1-12
- DATEDISPLAY, 1-13
- DATEFNS, 1-13
- DATEFORMAT, 1-14
- DATETIME, 1-14
- DEFCENT, 1-15
- DISPLAY, 1-15
- DTSTRICT, 1-16
- EMPTYREPORT, 1-16
- ESTRECORDS, 1-17
- EUROFILE, 1-18
- EXTAGGR, 1-18

SET parameters (*continued*)

- EXTHOLD, 1-18
- EXTSORT, 1-19
- EXTTERM, 1-19
- FIELDNAME, 1-19
- FILENAME, 1-20
- FILTER, 1-20
- FIXRETRIEVE, 1-21
- FOC144, 1-21
- FOC2GIGDB, 1-22
- FOCALLOC, 1-22
- FOCSTACK, 1-22
- HDAY, 1-23
- HLISUDUMP, 1-23
- HLISUTRACE, 1-23
- HOLDATTR, 1-24
- HOLDLIST, 1-24
- HOLDSTAT, 1-25
- HOTMENU, 1-25
- IBMLE, 1-26
- IMMEDTYPE, 1-26
- IMS, 1-26
- INDEX, 1-27
- JOINOPT, 1-27
- LANG, 1-28
- LEADZERO, 1-29
- LEFTMARGIN, 1-30
- LINES, 1-30
- MASTER, 1-31
- MAXLRECL, 1-31
- MESSAGE, 1-31
- MINIO, 1-32
- MULTIPATH, 1-33
- NODATA, 1-33
- ONLINE-FMT, 1-34
- ORIENTATION, 1-34
- PAGE-NUM, 1-35
- PAGESIZE, 1-35
- PANEL, 1-38
- PAPER, 1-39
- PASS, 1-39
- PAUSE, 1-40
- PFnn, 1-40
- PREFIX, 1-41
- PRINT, 1-41
- PRINTPLUS, 1-42

SET parameters (*continued*)

QUALCHAR, 1-43  
 QUALTITLES, 1-43  
 REBUILDMSG, 1-44  
 RECAP-COUNT, 1-44  
 RECORDLIMIT, 1-44  
 RIGHTMARGIN, 1-45  
 RPAGESET, 1-45  
 SAVEMATRIX, 1-45  
 SBORDER, 1-46  
 SCREEN, 1-46  
 SHADOW, 1-47  
 SHIFT, 1-47  
 SORTLIB, 1-48  
 SPACES, 1-48  
 SQLTOPTIF, 1-49  
 SQUEEZE, 1-49  
 STYLE[SHEET], 1-50  
 SUMPREFIX, 1-51  
 SUSI, 1-51  
 SUTABSIZE, 1-51  
 TEMP[DISK], 1-51  
 TERM, 1-52  
 TESTDATE, 1-52  
 TEXTFIELD, 1-53  
 TITLE, 1-53  
 TOPMARGIN, 1-54  
 TRACKIO, 1-54  
 TRMOUT, 1-54  
 UNITS, 1-55  
 USER, 1-55  
 WEEKFIRST, 1-57  
 WIDTH, 1-56, 1-57  
 XRETRIEVAL, 1-58  
 YRTHRESH, 1-58

SET TESTDATE command, 7-10

SHADOW parameter, 1-47

SHIFT parameter, 1-47

sliding window, 7-2

  calculated value, 7-26

  date format, 7-5

  date options, 7-31

  DEFCENT parameter, 7-2, 7-5

  defining, 7-3

sliding window (*continued*)

  defining in Master File, 7-13

  defining with SET, 7-7, 7-10

  dynamic window, 7-4, 7-10

  field-level, 7-13

  file-level, 7-13

  global, 7-7

  legacy date, 7-5

  YRTHRESH parameter, 7-2, 7-5

sort

  external, 2-23, 2-24

  status of, 2-23

SORTLIB parameter, 1-48

SPACES parameter, 1-48

special variables, 4-62

SQL date, 7-5

SQLTOPTTF parameter, 1-49

SQRT function, 3-163

SQUEEZE parameter, 1-49

startup profile, 4-36

statistical variables, 4-6, 4-49, 4-60

strings, 3-56, 3-156, 3-170

  alphanumeric, 3-93

  centering in field, 3-68

  character, 3-106

  justifying, 3-161

  substrings, 3-164

structure diagrams, A-2

STYLE[SHEET] parameter, 1-50

stylesheets, 2-26

  ? STYLE command, 2-26

SU machine, 2-24

subroutines, C-2

  accessing external, 3-30

  accessing in MVS, 3-30

  accessing on VM/CMS, 3-34

  arguments, 3-19, 3-20, C-3

  calling, 4-33

subroutines (*continued*)  
  command calls, 3-18  
  compiling, C-12  
  compiling in CMS, C-12  
  compiling in MVS, C-13  
  creating, C-2  
  custom, C-14  
  date, 3-9  
  external, 3-30  
  IF command, 3-24  
  -IF command, 3-24  
  in COMPUTE command, 3-22  
  in DEFINE command, 3-22  
  in functions, 3-21  
  in IF selection tests, 3-23  
  in VALIDATE command, 3-22  
  in WHERE selection tests, 3-23  
  language, C-5  
  MTHNAM, C-14  
  programming, C-5, C-7, C-9  
  RECAP command, 3-28  
  REXX, C-22  
  -RUN commands, 3-26  
  storing, C-12  
  storing external, 3-30  
  storing in CMS, C-12  
  storing in MVS, C-13  
  storing on VM/CMS, 3-34  
  testing, C-13  
  WHEN criteria, 3-27  
  with sliding window, 7-5, 7-23  
  writing, C-3

substitutions, 2-14  
  ? LET command, 2-14  
  checking, 5-10  
  clearing, 5-11  
  LET command, 5-2  
  multiple-line, 5-8  
  null, 5-7  
  recursive, 5-8  
  saving, 5-12  
  variable, 5-5  
  variables, 4-49

SUBSTR function/subroutine, 3-164

substrings, 3-149, 3-164

SUMPREFIX parameter, 1-51

SUSI parameter, 1-51

SUTABSIZE parameter, 1-51

system date, 7-31

system defaults, 4-97

system functions and subroutines, 3-13  
  FEXERR, 3-96  
  FINDMEM, 3-97  
  GETPDS, 3-101  
  GETUSER, 3-108  
  HHMMSS, 3-120  
  MVSDYNAM, 3-145  
  TODAY, 3-166

system variables, 4-6, 4-49, 4-54

## T

TED (text editor), 4-6

TEMP[DISK] parameter, 1-51

temporary fields  
  ? DEFINE command, 2-4  
  displaying, 2-4

TERM parameter, 1-52

TESTDATE parameter, 1-52

text display windows, 9-7  
  creating, 9-35, 9-39

text input windows, 9-6

TEXTFIELD parameter, 1-53

TITLE parameter, 1-53

TODAY function/subroutine, 3-166

TOPMARGIN parameter, 1-54

TRACKIO parameter, 1-54

TRAINING data source, A-19

TRMOUT parameter, 1-54

- TSO RUN command, 4-10, 4-93
  - subroutines, 3-26
- TYPE command, 4-10, 4-11, 4-93

## U

- UFMT subroutine, 3-168
- UNITS parameter, 1-55
- UPCASE subroutine, 3-170
- user IDs, 3-108
  - retrieving, 3-108
- USER parameter, 1-55
- Utilities Menu, 9-72

## V

- VALIDATE command, 3-22
  - subroutines in, 3-22
- values, 4-49
  - decoding, 3-83
  - default, 4-97
  - goto, 9-25
  - maximum, 3-144
  - minimum, 3-144
  - variables, 4-49
  - verifying input values, 4-76
- variable operations
  - altering commands, 4-63
  - changing value, 4-29
  - computing, 4-28
  - concatenating, 4-65
  - evaluating, 4-63
  - format conditions, 4-76
  - in procedures, 4-49
  - prompting, 4-79
  - querying, 4-51
  - substitution, 4-49
  - supplying values, 4-66, 4-68
  - valid values, 4-77
- variable substitutions, 5-5

- variable types, 4-6
  - amper variables, 4-30, 4-49, 4-51
  - global, 4-6, 4-49, 4-53
  - local, 4-6, 4-49, 4-52
  - positional, 4-69
  - special, 4-62
  - statistical, 4-6, 4-49, 4-60
  - system, 4-6, 4-49, 4-54
- variable values, 4-68
  - DEFAULTS command, 4-70
  - READ command, 4-72
  - SET sommand, 4-71
  - supplying from external files, 4-72
- vertical menus, 9-5
  - creating, 9-41, 9-47
- VIDEOTR2 Access File, A-27
- VIDEOTR2 data source, A-26
- VideoTrk data source, A-24
- virtual fields, 7-14
  - sliding window, 7-20

## W

- WEEKFIRST parameter, 1-57
- WHERE selection tests, 3-23
  - subroutines in, 3-23, 3-27
- WIDTH parameter, 1-56, 1-57
- window applications
  - transferring control, 9-22
- WINDOW command, 4-10, 4-49, 4-75, 4-94, 9-3
- WINDOW COMPILE command, 9-83
- Window Creation Menu, 9-57
- Window Design Screen, 9-59
- Window facility, 9-22
- window files, 9-3, 9-4
  - creating, 9-33

Window Options Menu, 9-61

- Display list, 9-65
- Heading, 9-64
- Help window, 9-67, 9-71
- Hide list, 9-66
- Line break, 9-69
- Multi-select, 9-70
- Popup, 9-67
- Quit, 9-71
- Show a window, 9-64
- Unshow a window, 9-64

WINDOW PAINT command, 9-52

Window Painter, 9-2

- goto values, 9-3
- invoking, 9-52
- main menu, 9-35
- screens, 9-51
- tutorial, 9-29
- window files, 9-3

Window Painter tutorial, 9-29

- SAMPLE, 9-31
- window file, 9-33

windows, 9-4

- &WINDOWNAME variable, 9-26
- &WINDOWVALUE variable, 9-26
- creating, 9-2, 9-14
- executing, 9-28
- execution, 9-11

windows (*continued*)

- field names, 9-8
- file contents, 9-9
- file names, 9-7
- horizontal, 9-6, 9-14
- multi-input, 9-12, 9-18
- procedures, 9-21
- return value display, 9-9
- returning to caller, 9-25
- text display, 9-7
- text input, 9-6
- types, 9-5
- vertical menu, 9-5

-WRITE command, 4-10, 4-45, 4-95

## X

XRETRIEVAL parameter, 1-58

## Y

YM subroutine, 3-174

YMD function, 3-86

YRTHRESH parameter, 1-58, 7-2, 7-3, 7-5, 7-7

- COMPUTE command, 7-26
- DEFINE command, 7-20
- querying, 7-12
- with COMPUTE, 7-27



---

# Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. Send comments to

Corporate Publications  
Attn: Manager of Documentation Services  
Information Builders  
Two Penn Plaza  
New York, NY 10121-2898

or FAX this page to (212) 967-0460, or call **Sara Elam** at (212) 736-4433, x**3207**.

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

Telephone: \_\_\_\_\_ Date: \_\_\_\_\_

Comments:

---

# Reader Comments