

# FOCUS for Mainframe

Using Functions

Version 7.3

EDA, EDA/SQL, FIDEL, FOCCALC, FOCUS, FOCUS Fusion, FOCUS Vision, Hospital-Trac, Information Builders, the Information Builders logo, Parlay, PC/FOCUS, SmartMart, SmartMode, SNAPPack, TableTalk, WALDO, Web390, WebFOCUS and WorldMART are registered trademarks, and iWay and iWay Software are trademarks of Information Builders, Inc.

Due to the nature of this material, this document refers to numerous hardware and software products by their trademarks. 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 © 2003, 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 how to use functions to perform certain calculations and manipulations. It is intended for application developers. This manual is part of the FOCUS documentation set.

The documentation set consists of the following components:

- 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 Using Functions manual describes internal functions and user written subroutines.
- 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 users' documentation for FOCUS Version 7.3 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.

## How This Manual Is Organized

---

This manual includes the following chapters:

Chapter/Appendix		Contents
1	How to Use This Manual	Describes how to use the information in the <i>FOCUS Using Functions</i> manual.
2	Introducing Functions	Offers an introduction to functions and explains the different types of functions available.
3	Accessing and Calling a Function	Describes the considerations for supplying arguments in a function, explains how to use a function in a command, and how to access externally-stored functions.
4	Character Functions	Describes the available character functions, which enable you to manipulate alphanumeric fields and character strings.
5	Maintain-specific Character Functions	Describes Maintain-specific character functions which manipulate alphanumeric fields and character strings.
6	Data Source and Decoding Functions	Describes the available data source functions, which enable you to search for or retrieve data source records or values.
7	Date and Time Functions	Describes the available date and time functions, which enable you to manipulate date and time values.
8	Maintain-specific Date and Time Functions	Describes Maintain-specific date and time functions which manipulate date and time values.
9	Format Conversion Functions	Describes the available format conversion functions, which convert fields from one format to another.
10	Numeric Functions	Describes the available numeric functions, which enable you to perform calculations on numeric constants and fields.
11	System Functions	Describes the available system functions, which enable you to make calls to the operating system to obtain information about the operating environment or to use a system service.

Chapter/Appendix		Contents
<b>A</b>	Creating a Subroutine	Describes how to create and store site-specific functions.

## Summary of New Features

---

The FOCUS for Mainframe documentation describes the following new features and enhancements:

New Feature	Manual	Chapter
Increased ACROSS values (from 95)	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
IN-RANGES-OF	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
SET BYDISPLAY	<i>Creating Reports</i>	Chapter 4, <i>Sorting Tabular Reports</i>
Extensions to FORECAST	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Multivariate Regress	<i>Creating Reports</i>	Chapter 6, <i>Creating Temporary Fields</i>
Summary Prefix Operators	<i>Creating Reports</i>	Chapter 7, <i>Including Totals and Subtotals</i>
AnV (VARCHAR) support	<i>Creating Reports</i>	Chapter 8, <i>Using Expressions</i>
	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>
	<i>Using Functions</i>	Chapter 4, <i>Character Functions</i>
Increased IF-THEN-ELSE	<i>Creating Reports</i>	Chapter 8, <i>Using Expressions</i>
FOCFIRSTPAGE/ &FOCNEXTPAGE	<i>Creating Reports</i>	Chapter 9, <i>Customizing Tabular Reports</i>
Increased Number of sort headings/footings	<i>Creating Reports</i>	Chapter 9, <i>Customizing Tabular Reports</i>
Increased column title space	<i>Creating Reports</i>	Chapter 9, <i>Customizing Tabular Reports</i>
Multiple FOLD-LINE	<i>Creating Reports</i>	Chapter 9, <i>Customizing Tabular Reports</i>

Summary of New Features

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
NEWPAGE	<i>Creating Reports</i>	Chapter 9, <i>Customizing Tabular Reports</i>
TBLASTPAGE	<i>Creating Reports</i>	Chapter 10, <i>Styling Reports</i>
Stylesheet enhancements	<i>Creating Reports</i>	Chapter 10, <i>Styling Reports</i>
Multiple reports in one PDF file	<i>Creating Reports</i>	Chapter 10, <i>Styling Reports</i>
Decimal Alignment of Headings	<i>Creating Reports</i>	Chapter 10, <i>Styling Reports</i>
Cascading Stylesheets	<i>Creating Reports</i>	Chapter 11, <i>Cascading Stylesheets</i>
Excel 2000	<i>Creating Reports</i>	Chapter 12, <i>Saving and Reusing Report Output</i> Chapter 10, <i>Styling Reports</i>
SET HOLDFORMAT	<i>Creating Reports</i>	Chapter 12, <i>Saving and Reusing Report Output</i>
Excel 97	<i>Creating Reports</i>	Chapter 12, <i>Saving and Reusing Report Output</i> Chapter 10, <i>Styling Reports</i>
Holding Missing values	<i>Creating Reports</i>	Chapter 13, <i>Handling Records With Missing Field Values</i>
Compiled Defines	<i>Creating Reports</i>	Chapter 16, <i>Improving Report Processing</i>
FML Hierarchy	<i>Creating Reports</i>	Chapter 17, <i>Creating Financial Reports</i>
FORMULTIPLE	<i>Creating Reports</i>	Chapter 17, <i>Creating Financial Reports</i>
Indenting FML Reports	<i>Creating Reports</i>	Chapter 17, <i>Creating Financial Reports</i>
SET BLANKINDENT	<i>Creating Reports</i>	Chapter 17, <i>Creating Financial Reports</i>

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
FML Hierarchy	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>
Long qualified field names	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>
Minus edit format option	<i>Describing Data</i>	Chapter 4, <i>Describing an Individual Field</i>
SUFFIX=TAB	<i>Describing Data</i>	Chapter 5, <i>Describing a Sequential, VSAM, or ISAM Data Source</i>
MDI	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
GROUPS in FOCUS Files	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
DATASET for a segment in MFD	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
XFOCUS Database	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
Long Segment Names, Long Index Names	<i>Describing Data</i>	Chapter 6, <i>Describing a FOCUS Data Source</i>
SET HNODATA	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
SET HOLDMISS	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
SET NULL=ON	<i>Developing Applications</i>	Chapter 1, <i>Customizing Your Environment</i>
SET SAVEDMASTERS	<i>Developing Applications</i>	Chapter 5, <i>Enhancing Application Performance</i>
Wide lines	<i>Developing Applications</i>	Chapter 3, <i>Managing Flow of Control in an Application</i>
	<i>Overview and Operating Environments</i>	Chapter 5, <i>CMS Guide to Operations</i> Chapter 6, <i>OS/390 and MVS Guide to Operations</i>

<b>New Feature</b>	<b>Manual</b>	<b>Chapter</b>
&FOCUSER	<i>Developing Applications</i>	Chapter 3, <i>Managing Flow of Control in an Application</i>
Long Amper variables	<i>Developing Applications</i>	Chapter 3, <i>Managing Flow of Control in an Application</i>
MAINTAIN FILETYPE Extension	<i>Maintaining Databases</i>	Chapter 2, <i>Maintain Concepts</i>
Enhanced screening conditions for Maintain	<i>Maintaining Databases</i>	Chapter 7, <i>Command Reference</i>
COMBINE 63 files	<i>Maintaining Databases</i>	Chapter 9, <i>Modifying Data Sources With MODIFY</i>
FOCUS SETs from Maintain	<i>Maintaining Databases</i>	Chapter 9, <i>Modifying Data Sources With MODIFY</i>
Raised Number of Partitions for External Index	<i>Maintaining Databases</i>	Chapter 11, <i>Creating and Rebuilding Databases</i>
IEDIT	<i>Overview and Operating Environments</i>	Chapter 3, <i>Invoking Your System Editor With IEDIT</i>
Relative GDG Number +1	<i>Overview and Operating Environments</i>	Chapter 6, <i>OS/390 and MVS Guide to Operations</i>
SET USERFCHK and SET USERFNS	<i>Using Functions</i>	Chapter 3, <i>Accessing and Calling a Function</i>
FMLINFO	<i>Using Functions</i>	Chapter 10, <i>Numeric Functions</i>
Subroutine NORMSINV and NORMSDST	<i>Using Functions</i>	Chapter 10, <i>Numeric Functions</i>



## Documentation Conventions

---

This manual describes the following conventions apply throughout this manual:

Convention	Description
<b>THIS TYPEFACE</b> or <i>this typeface</i>	Denotes syntax that you must enter exactly as shown.
<i>this typeface</i>	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
<u>underscore</u>	Indicates a default setting.
<i>this typeface</i>	Represents a placeholder (or variable), a cross-reference, or an important term. It may also indicate a button, menu item, or dialog box option you can click or select.
<b>this typeface</b>	Highlights a file name or command.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices; type one of them, not the braces.
[ ]	Indicates a group of optional parameters. None are required, but you may select one of them. Type only the parameter in the brackets, not the brackets.
	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
...	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis points (...).
. . . . . .	Indicates that there are (or could be) intervening or additional commands.

## Related Publications

---

To view a current listing of our publications and to place an order, visit our World Wide Web site, <http://www.informationbuilders.com>. You can also contact the Publications Order Department at (800) 969-4636.

## 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 (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 database 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 Online, 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 (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 data sources, have you tried executing a query containing just the code to access the data source?
- 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 welcomes your opinions 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 Documentation Feedback 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.

---

---

# Contents

<b>1. How to Use This Manual</b> .....	<b>1-1</b>
Available Languages .....	1-2
Operating Systems .....	1-2
<b>2. Introducing Functions</b> .....	<b>2-1</b>
Using Functions .....	2-2
Types of Functions .....	2-3
Character Functions .....	2-3
Maintain-specific Character Functions .....	2-7
Data Source and Decoding Functions .....	2-10
Date and Time Functions .....	2-11
Maintain-specific Date and Time Functions .....	2-16
Format Conversion Functions .....	2-18
Numeric Functions .....	2-19
System Functions .....	2-21
<b>3. Accessing and Calling a Function</b> .....	<b>3-1</b>
Calling a Function .....	3-2
Supplying an Argument in a Function .....	3-4
Argument Types .....	3-4
Argument Formats .....	3-5
Argument Length .....	3-6
Number and Order of Arguments .....	3-6
Verifying Function Parameters .....	3-7
Calling a Function From a DEFINE, COMPUTE, or VALIDATE Command .....	3-10
Calling a Function From a Dialogue Manager Command .....	3-11
Assigning the Result of a Function to a Variable .....	3-11
Branching Based on a Function's Result .....	3-13
Calling a Function From an Operating System RUN Command .....	3-15
Calling a Function From Another Function .....	3-16
Calling a Function in WHERE or IF Criteria .....	3-17
Using a Calculation or Compound IF Command .....	3-18
Calling a Function in WHEN Criteria .....	3-19
Calling a Function From a RECAP Command .....	3-20
Storing and Accessing an External Function .....	3-22
Storing and Accessing a Function on OS/390 .....	3-22
Storing and Accessing a Function on UNIX .....	3-24
Storing and Accessing a Function on VM/CMS .....	3-24
Dynamic Language Environment Support .....	3-27

<b>4. Character Functions</b> .....	<b>4-1</b>
ARGLEN: Measuring the Length of a Character String .....	4-2
ASIS: Distinguishing Between a Space and a Zero .....	4-3
BITSON: Determining If a Bit Is On or Off .....	4-5
BITVAL: Evaluating a Bit String as a Binary Integer .....	4-6
BYTVAL: Translating a Character to a Decimal Value .....	4-8
CHKFMT: Checking the Format of a Character String .....	4-10
CTRAN: Translating One Character to Another .....	4-14
CTRFLD: Centering a Character String .....	4-19
EDIT: Extracting or Adding Characters .....	4-21
GETTOK: Extracting a Substring (Token) .....	4-22
LCWORD: Converting a Character String to Mixed Case .....	4-24
LJUST: Left-Justifying a Character String .....	4-26
LOCASE: Converting Text to Lowercase .....	4-28
OVLAY: Overlaying a Character String .....	4-29
PARAG: Dividing Text Into Smaller Lines .....	4-33
POSIT: Finding the Beginning of a Substring .....	4-35
RJUST: Right-Justifying a Character String .....	4-37
SOUNDEX: Comparing Character Strings Phonetically .....	4-39
SQUEEZ: Reducing Multiple Spaces to a Single Space .....	4-40
STRIP: Removing a Character From a String .....	4-42
SUBSTR: Extracting a Substring .....	4-44
TRIM: Removing Leading and Trailing Occurrences .....	4-46
UPCASE: Converting Text to Uppercase .....	4-49
Character Functions for AnV Fields .....	4-52
LENV: Returning the Length of an Alphanumeric Field .....	4-53
LOCASV: Creating a Variable Length Lowercase String .....	4-54
POSITV: Finding the Beginning of a Variable Length Substring .....	4-56
SUBSTV: Extracting a Variable Length Substring .....	4-57
TRIMV: Removing Characters From a String .....	4-60
UPCASV: Creating a Variable Length Uppercase String .....	4-61
<b>5. Maintain-specific Character Functions</b> .....	<b>5-1</b>
CHAR2INT: Translating a Character to an Integer Value .....	5-2
INT2CHAR: Translating an Integer Value to a Character .....	5-3
LCWORD and LCWORD2: Converting a Character String to Mixed Case .....	5-4
LENGTH: Determining the Length of a Character String .....	5-5
LJUST: Left-Justifying a Character String (Maintain) .....	5-6
LOWER: Converting a Character String to Lowercase .....	5-6
MASK: Extracting or Adding Characters .....	5-7
NLSCHR: Converting Characters From the Native English Code Page .....	5-8
OVLAY: Overlaying a Character String (Maintain) .....	5-9
POSIT: Finding the Beginning of a Substring (Maintain) .....	5-11

RJUST: Right-Justifying a Character String (Maintain)	5-13
SELECTS: Decoding a Value From a Stack	5-14
STRAN: Substituting One Substring for Another	5-16
STRCMP: Comparing Character Strings	5-18
STRICMP: Comparing Character Strings and Ignoring Case	5-20
STRNCMP: Comparing Character Substrings	5-21
STRTOKEN: Extracting a Substrings Based on Delimiters	5-22
SUBSTR: Extracting a Substring (Maintain)	5-24
TRIM: Removing Trailing Occurrences (Maintain)	5-25
TRIMLEN: Determining the Length of a String Excluding Trailing Spaces	5-26
UPCASE: Converting Text to Uppercase (Maintain)	5-27
<b>6. Data Source and Decoding Functions</b>	<b>6-1</b>
DECODE: Decoding Values	6-2
FIND: Verifying the Existence of a Value in an Indexed Field	6-6
LAST: Retrieving the Preceding Value	6-9
LOOKUP: Retrieving a Value From a Cross-referenced Data Source	6-11
Using the Extended LOOKUP Function	6-17
<b>7. Date and Time Functions</b>	<b>7-1</b>
Date and Time Function Terminology	7-2
Using Standard Date and Time Functions	7-2
Specifying Work Days	7-3
Enabling Leading Zeros For Date and Time Functions in Dialogue Manager	7-5
Using Date and Time Formats	7-7
Assigning Date-Time Values	7-9
DATEADD: Adding or Subtracting a Date Unit to or From a Date	7-13
DATECVT: Converting the Format of a Date	7-16
DATEDIF: Finding the Difference Between Two Dates	7-18
DATEMOV: Moving a Date to a Significant Point	7-21
HADD: Incrementing a Date-Time Value	7-25
HCNVRT: Converting a Date-Time Value to Alphanumeric Format	7-27
HDATE: Converting the Date Portion of a Date-Time Value to a Date Format	7-28
HDIFF: Finding the Number of Units Between Two Date-Time Values	7-30
HDTTM: Converting a Date Value to a Date-Time Value	7-32
HGETC: Storing the Current Date and Time in a Date-Time Field	7-33
HHMMSS: Retrieving the Current Time	7-34
HINPUT: Converting an Alphanumeric String to a Date-Time Value	7-36
H MIDNT: Setting the Time Portion of a Date-Time Value to Midnight	7-38
HNAME: Retrieving a Date-Time Component in Alphanumeric Format	7-39
HPART: Retrieving a Date-Time Component in Numeric Format	7-41
HSETPT: Inserting a Component Into a Date-Time Value	7-42
HTIME: Converting the Time Portion of a Date-Time Value to a Number	7-45

TODAY: Returning the Current Date .....	7-46
Using Legacy Date Functions .....	7-47
Using Old Versions of Legacy Date Functions .....	7-47
Using Dates With Two- and Four-Digit Years .....	7-48
AYM: Adding or Subtracting Months to or From Dates .....	7-50
AYMD: Adding or Subtracting Days to or From a Date .....	7-51
CHGDAT: Changing Format of a Date .....	7-53
DA Functions: Converting a Date to an Integer .....	7-56
DMY, MDY, YMD: Calculating the Difference Between Two Dates .....	7-58
DOWK and DOWKL: Finding the Day of the Week .....	7-60
DT Functions: Converting an Integer to a Date .....	7-61
GREGDT: Converting From Julian to Gregorian Format .....	7-63
JULDAT: Converting From Gregorian to Julian Format .....	7-65
YM: Calculating Elapsed Months .....	7-67
<b>8. Maintain-specific Date and Time Functions .....</b>	<b>8-1</b>
Maintain-specific Standard Date and Time Functions .....	8-2
HHMMSS: Retrieving the Current Time (Maintain) .....	8-2
Initial_HHMMSS: Returning the Time the Application Was Started .....	8-3
Initial_TODAY: Returning the Date the Application Was Started .....	8-3
TODAY: Retrieving the Current Date (Maintain) .....	8-3
TODAY2: Returning the Current Date .....	8-4
Maintain-specific Legacy Date Functions .....	8-5
ADD: Adding Days to a Date .....	8-5
DAY: Extracting the Day of the Month From a Date .....	8-6
JULIAN: Determining How Many Days Have Elapsed in the Year .....	8-6
MONTH: Extracting the Month From a Date .....	8-7
QUARTER: Determining the Quarter .....	8-8
SETMDY: Setting the Value to a Date .....	8-8
SUB: Subtracting a Value From a Date .....	8-10
WEEKDAY: Determining the Day of the Week for a Date .....	8-11
YEAR: Extracting the Year From a Date .....	8-12
<b>9. Format Conversion Functions .....</b>	<b>9-1</b>
ATODBL: Converting an Alphanumeric String to Double-Precision Format .....	9-2
EDIT: Converting the Format of a Field .....	9-5
FTOA: Converting a Number to Alphanumeric Format .....	9-7
HEXBYT: Converting a Decimal Integer to a Character .....	9-8
ITONUM: Converting a Large Binary Integer to Double-Precision Format .....	9-11
ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format .....	9-12
ITOZ: Converting a Number to Zoned Format .....	9-14
PCKOUT: Writing a Packed Number of Variable Length .....	9-16
UFMT: Converting an Alphanumeric String to Hexadecimal .....	9-18



<b>10. Numeric Functions</b> .....	<b>10-1</b>
ABS: Calculating Absolute Value .....	10-2
ASIS: Distinguishing Between a Blank and a Zero .....	10-3
BAR: Producing a Bar Chart .....	10-3
CHKPCK: Validating a Packed Field .....	10-6
DMOD, FMOD, and IMOD: Calculating the Remainder From a Division .....	10-9
EXP: Raising “e” to the Nth Power .....	10-11
EXPN: Evaluating a Number in Scientific Notation .....	10-13
FMLINFO: Returning FOR Values .....	10-14
INT: Finding the Greatest Integer .....	10-16
LOG: Calculating the Natural Logarithm .....	10-17
MAX and MIN: Finding the Maximum or Minimum Value .....	10-18
NORMSDST and NORMSINV: Calculating Cumulative Normal Distribution .....	10-19
PRDNOR and PRDUNI: Generating Reproducible Random Numbers .....	10-23
RDNORM and RDUNIF: Generating Random Numbers .....	10-26
SQRT: Calculating the Square Root .....	10-28
<b>11. System Functions</b> .....	<b>11-1</b>
FEXERR: Retrieving an Error Message .....	11-2
FGETENV: Retrieving the Value of an Environment Variable .....	11-3
FINDMEM: Finding a Member of a Partitioned Data Set .....	11-4
FPUTENV: Assigning a Value to an Environment Variable .....	11-6
GETPDS: Determining If a Member of a Partitioned Data Set Exists .....	11-7
GETUSER: Retrieving a User ID .....	11-12
HHMMSS: Retrieving the Current Time .....	11-13
MVSDYNAM: Passing a DYNAM Command to the Command Processor .....	11-14
TODAY: Returning the Current Date .....	11-17

<b>A. Creating a Subroutine</b> .....	<b>A-1</b>
Writing a Subroutine .....	A-2
Naming a Subroutine .....	A-3
Creating Arguments .....	A-3
Language Considerations .....	A-4
Programming a Subroutine .....	A-6
Compiling and Storing a Subroutine .....	A-11
Compiling and Storing a Subroutine on VM/CMS .....	A-11
Compiling and Storing a Subroutine on OS/390 .....	A-12
Compiling and Storing a Subroutine on UNIX .....	A-12
Testing the Subroutine .....	A-12
Using a Custom Subroutine: The MTHNAM Subroutine .....	A-13
Writing the MTHNAM Subroutine .....	A-13
Calling the MTHNAM Subroutine From a Request .....	A-19
Subroutines Written in REXX .....	A-20
Formats and REXX Subroutines .....	A-25
Compiling FUSREXX Macros in VM/CMS .....	A-31

---

---

## CHAPTER 1

# How to Use This Manual

### Topics:

- Available Languages
- Operating Systems

This manual describes the functions supplied with your Information Builders product. It is intended for application developers who call these functions from their programs to perform calculations or manipulate data. Other users who access corporate data to produce reports can call these functions.

This manual also explains how to create functions tailored to individual needs (called subroutines) for use with your Information Builders product.

In this manual, the description of each function identifies the language and operating system for which it is valid.

## Available Languages

---

A function is available in the reporting language, the Maintain language, or both:

- The reporting language includes all commands used to create a report. It is available to users of any Information Builders product.
- The Maintain language includes all commands used to maintain data sources with the Maintain product. It is available only to those who purchased Maintain.

Look in the description of an individual function for the available language, or in the categorized list of functions in Chapter 2, *Introducing Functions*.

## Operating Systems

---

A function can be available on the following operating systems: AS/400, HP, OpenVMS, z/OS OS/390, VM/ESA, and UNIX.

Look in the description of an individual function for the available operating system, or in the categorized list of functions in Chapter 2, *Introducing Functions*.

The description "All" applies to all operating systems named here.

---

---

## CHAPTER 2

# Introducing Functions

### Topics:

- Using Functions
- Types of Functions

The following topics offer an introduction to functions and explain the different types of functions available.

## Using Functions

---

Functions operate on one or more arguments and return a single value. The returned value can be stored in a field, assigned to a Dialogue Manager variable, used in a calculation or other processing, or used in a selection or validation test. Functions provide a convenient way to perform certain calculations and manipulations.

There are three types of functions:

- **Internal functions.** Built into the FOCUS language, requiring no extra work to access or use. The following reporting and Maintain functions are internal functions. You can not replace any of these internal functions with your own functions of the same name. All other functions are external.
  - ABS
  - ASIS
  - DMY, MDY, and YMD
  - DECODE
  - EDIT
  - FIND
  - LAST
  - LOG
  - LOOKUP
  - MAX and MIN
  - SQRT
  - All Maintain-specific functions
- **External functions.** Stored in an external library that must be accessed. When invoking these functions, an argument specifying the output field or format of the result is required. External functions are distributed with FOCUS. You can replace these functions with your own functions of the same name. However, in this case, you must set USERFNS=LOCAL.
- **Subroutines.** Written by the user and stored externally. For details, see Appendix A, *Creating a Subroutine*.

For information on how to use an internal or external function, see Chapter 3, *Accessing and Calling a Function*.

## Types of Functions

---

You can access any of the following types of functions:

- **Character functions.** Manipulate alphanumeric fields or character strings. For details, see *Character Functions* on page 2-3.
- **Maintain-specific character functions.** Manipulate alphanumeric fields or character strings. These functions are available only in Maintain. For details, see *Maintain-specific Character Functions* on page 2-7.
- **Data source and decoding functions.** Search for or retrieve data source records or values, and assign values. For details, see *Data Source and Decoding Functions* on page 2-10.
- **Date and time functions.** Manipulate dates and times. For details, see *Date and Time Functions* on page 2-11.
- **Maintain-specific date and time functions.** Manipulate dates and times. These functions are available only in Maintain. For details, see *Maintain-specific Date and Time Functions* on page 2-16.
- **Format conversion functions.** Convert fields from one format to another.  
For details, see *Format Conversion Functions* on page 2-18.
- **Numeric functions.** Perform calculations on numeric constants and fields. For details, see *Numeric Functions* on page 2-19.
- **System functions.** Call the operating system to obtain information about the operating environment or to use a system service. For details, see *System Functions* on page 2-21.

### Character Functions

The following functions manipulate alphanumeric fields or character strings. For details, see Chapter 4, *Character Functions*.

#### ARGLEN

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

Available Operating Systems: All

Available Languages: reporting, Maintain

#### ASIS

Distinguishes between a blank and a zero in Dialogue Manager.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX

Available Languages: reporting

**BITSON**

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

Available Operating Systems: All

Available Languages: reporting, Maintain

**BITVAL**

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

Available Operating Systems: All

Available Languages: reporting, Maintain

**BYTVAL**

Translates a character to its corresponding ASCII or EBCDIC decimal value.

Available Operating Systems: All

Available Languages: reporting, Maintain

**CHKFMT**

Checks a character string for incorrect characters or character types.

Available Operating Systems: All

Available Languages: reporting, Maintain

**CTRAN**

Translates a character within a character string to another character based on its decimal value.

Available Operating Systems: All

Available Languages: reporting, Maintain

**CTRFLD**

Centers a character string within a field.

Available Operating Systems: All

Available Languages: reporting, Maintain

**EDIT**

Extracts characters from or adds characters to a character string.

Available Operating Systems: All

Available Languages: reporting

**GETTOK**

Divides a character string into substrings, called tokens, where a specific character, called a delimiter, occurs in the string.

Available Operating Systems: All

Available Languages: reporting, Maintain



**LCWORD**

Converts the letters in a character string to mixed case.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**LJUST**

Left-justifies a character string within a field.

Available Operating Systems: All

Available Languages: reporting

**LOCASE**

Converts alphanumeric text to lowercase.

Available Operating Systems: All

Available Languages: reporting, Maintain

**OVLAY**

Overlays a base character string with a substring.

Available Operating Systems: All

Available Languages: reporting

**PARAG**

Divides a line of text into smaller lines by marking them with a delimiter.

Available Operating Systems: All

Available Languages: reporting, Maintain

**POSIT**

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

Available Operating Systems: All

Available Languages: reporting

**RJUST**

Right-justifies a character string.

Available Operating Systems: All

Available Languages: reporting

**SOUNDEX**

Searches for a character string phonetically without regard to spelling.

Available Operating Systems: All

Available Languages: reporting, Maintain

**SQUEEZ**

Reduces multiple contiguous spaces within a character string to a single space.

Available Operating Systems: All

Available Languages: reporting, Maintain

**STRIP**

Removes all occurrences of a specific character from a string.

Available Operating Systems: All

Available Languages: reporting, Maintain

**SUBSTR**

Extracts a substring based on where it begins and its length in the parent string.

Available Operating Systems: All

Available Languages: reporting

**TRIM**

Removes leading and/or trailing occurrences of a pattern within a character string.

Available Operating Systems: All

Available Languages: reporting

**UPCASE**

Converts a character string to uppercase.

Available Operating Systems: All

Available Languages: reporting

**Character Functions for AnV Fields**

**LENV**

Returns the actual length of an AnV field or the size of an An field.

Available Operating Systems: All

Available Languages: reporting

**LOCASV**

Converts alphanumeric text to lowercase in an AnV field.

Available Operating Systems: All

Available Languages: reporting

**POSITV**

Finds the starting position of a substring in an AnV field.

Available Operating Systems: All

Available Languages: reporting

### **SUBSTV**

Extracts a substring based on where it begins and its length in the parent string in an *AnV* field.

Available Operating Systems: All

Available Languages: reporting

### **TRIMV**

Removes leading and/or trailing occurrences of a pattern within a character string in an *AnV* field.

Available Operating Systems: All

Available Languages: reporting

### **UPCASV**

Converts a character string to uppercase in an *AnV* field.

Available Operating Systems: All

Available Languages: reporting

## **Maintain-specific Character Functions**

The following functions manipulate alphanumeric fields or character strings. They are available only in the Maintain language. For details, see Chapter 5, *Maintain-specific Character Functions*.

### **CHAR2INT**

Translates an ASCII or EBCDIC character to the integer value it represents, depending on the operating system.

Available Operating Systems: All

Available Languages: Maintain

### **INT2CHAR**

Translates an integer into the equivalent ASCII or EBCDIC character, depending on the operating system.

Available Operating Systems: All

Available Languages: Maintain

### **LCWORD and LCWORD2**

Converts the letters in a character string to mixed case.

Available Operating Systems: All

Available Languages: Maintain

**LENGTH**

Measures the length of a character string, including trailing blanks.

Available Operating Systems: All

Available Languages: Maintain

**LJUST**

Left-justifies a character string within a field.

Available Operating Systems: All

Available Languages: Maintain

**LOWER**

Converts a character string to lowercase.

Available Operating Systems: All

Available Languages: Maintain

**MASK**

Extracts characters from or adds characters to a character string.

Available Operating Systems: All

Available Languages: Maintain

**NLSCHR**

Converts a character from the native English code page to the running code page.

Available Operating Systems: All

Available Languages: Maintain

**OVLAY**

Overlays a base character string with a substring.

Available Operating Systems: All

Available Languages: Maintain

**POSIT**

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

Available Operating Systems: All

Available Languages: Maintain

**RJUST**

Right-justifies a character string.

Available Operating Systems: All

Available Languages: Maintain

**SELECTS**

Decodes a value from a stack.

Available Operating Systems: All

Available Languages: Maintain

**STRAN**

Substitutes a substring for another substring in a character string.

Available Operating Systems: All

Available Languages: Maintain

**STRCMP**

Compares two alphanumeric strings using the ASCII or EBCDIC collating sequence.

Available Operating Systems: All

Available Languages: Maintain

**STRICMP**

Compares two alphanumeric strings using the ASCII or EBCDIC collating sequence, but ignoring case differences.

Available Operating Systems: All

Available Languages: Maintain

**STRNCMP**

Compares a specified number of characters in two character strings starting at the beginning of the strings using the EBCDIC or ASCII collating sequence.

Available Operating Systems: All

Available Languages: Maintain

**SUBSTR**

Extracts a substring based on where it begins and its length in the parent string.

Available Operating Systems: All

Available Languages: Maintain

**TRIM**

Removes trailing occurrences of a pattern within a character string.

Available Operating Systems: All

Available Languages: Maintain

**TRIMLEN**

Determines the length of a character string excluding trailing spaces.

Available Operating Systems: All

Available Languages: Maintain

**UPCASE**

Converts a character string to uppercase.

Available Operating Systems: All

Available Languages: Maintain

**Data Source and Decoding Functions**

The following functions search for data source records, retrieve data source records or values, and assign values. For details, see Chapter 6, *Data Source and Decoding Functions*.

**DECODE**

Assigns values based on the coded value of an input field.

Available Operating Systems: All

Available Languages: reporting, Maintain

**FIND**

Determines if an incoming data value is in an indexed FOCUS data source field.

Available Operating Systems: All

Available Languages: reporting

**LAST**

Retrieves the preceding value for a field.

Available Operating Systems: All

Available Languages: reporting

**LOOKUP**

Retrieves a data value from a cross-referenced FOCUS data source in a MODIFY request.

Available Operating Systems: All

Available Languages: reporting

## Date and Time Functions

The following functions manipulate dates and times. For details see Chapter 7, *Date and Time Functions*.

### Standard Date and Time Functions

#### DATEADD

Adds a unit to or subtracts a unit from a date format.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

#### DATECVT

Converts date formats.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

#### DATEDIF

Returns the difference between two dates in units.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

#### DATEMOV

Moves a date to a significant point on the calendar.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

#### HADD

Increments a date-time field by a given number of units.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

#### HCVRT

Converts a date-time field to a character string.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

#### HDATE

Extracts the date portion of a date-time field, converts it to a date format, and returns the result in the format YYMD.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HDIFF**

Calculates the number of units between two date-time values.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HDTTM**

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HGETC**

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HHMMSS**

Retrieves the current time from the system.

Available Operating Systems: All

Available Languages: reporting

**HINPUT**

Converts an alphanumeric string to a date-time value.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HMIDNT**

Changes the time portion of a date-time field to midnight (all zeroes).

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HNAME**

Extracts a specified component from a date-time field and returns it in alphanumeric format.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HPART**

Extracts a specified component from a date-time field and returns it in numeric format.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain



**HSETPT**

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**HTIME**

Converts the time portion of a date-time field to the number of milliseconds or microseconds.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**TIMETOTS**

Converts a time to a timestamp.

Available Operating Systems: All

Available Languages: reporting, Maintain

**TODAY**

Retrieves the current date from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

## **Legacy Date Functions**

### **AYM**

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

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **AYMD**

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

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **CHGDAT**

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

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **DA**

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

*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.

Available Operating Systems: All

Available Languages: reporting, Maintain

### **DMY, MDY, and YMD**

Calculate the difference between two dates.

Available Operating Systems: All

Available Languages: reporting, Maintain

### **DOWK and DOWKL**

Find the day of the week that corresponds to a date.

Available Operating Systems: All

Available Languages: reporting, Maintain

**DT**

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

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.

Available Operating Systems: All

Available Languages: reporting, Maintain

**GREGDT**

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

Available Operating Systems: All

Available Languages: reporting, Maintain

**JULDAT**

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

Available Operating Systems: All

Available Languages: reporting, Maintain

**YM**

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

Available Operating Systems: All

Available Languages: reporting, Maintain

## **Maintain-specific Date and Time Functions**

The following functions manipulate dates and times. They are available only in the Maintain language. For details, see *Chapter 8, Maintain-specific Date and Time Functions*.

### **Maintain-specific Standard Date and Time Functions**

#### **HHMMSS**

Retrieves the current time from the system.

Available Operating Systems: All

Available Languages: Maintain

#### **Initial\_HHMMSS**

Retrieves the time that the Maintain module was started.

Available Operating Systems: All

Available Languages: Maintain

#### **Initial\_TODAY**

Retrieves the date that the Maintain module was started.

Available Operating Systems: All

Available Languages: Maintain

#### **TODAY**

Retrieves the current date from the system.

Available Operating Systems: All

Available Languages: Maintain

#### **TODAY2**

Retrieves the current date from the system.

Available Operating Systems: All

Available Languages: Maintain

## **Maintain-specific Legacy Date Functions**

### **ADD**

Adds a given number of days to a date.

Available Operating Systems: All

Available Languages: Maintain

### **DAY**

Extracts the day of the month from a date.

Available Operating Systems: All

Available Languages: Maintain

### **JULIAN**

Determines the number of days that have elapsed so far in the year up to a given date.

Available Operating Systems: All

Available Languages: Maintain

### **MONTH**

Extracts the month from a date.

Available Operating Systems: All

Available Languages: Maintain

### **QUARTER**

Determines the quarter of the year in which a date resides.

Available Operating Systems: All

Available Languages: Maintain

### **SETMDY**

Sets a value to a date.

Available Operating Systems: All

Available Languages: Maintain

### **SUB**

Subtracts a given number of days from a date.

Available Operating Systems: All

Available Languages: Maintain

### **WEEKDAY**

Determines the day of the week for a date.

Available Operating Systems: All

Available Languages: Maintain

**YEAR**

Extracts the year from a date.

Available Operating Systems: All

Available Languages: Maintain

**Format Conversion Functions**

The following functions convert fields from one format to another. For details, see Chapter 9, *Format Conversion Functions*.

**ATODBL**

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

Available Operating Systems: All

Available Languages: reporting, Maintain

**EDIT**

Converts an alphanumeric field that contains numeric characters to numeric format or converts a numeric field to alphanumeric format.

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting

**FTOA**

Converts a number in a numeric format to alphanumeric format.

Available Operating Systems: All

Available Languages: reporting, Maintain

**HEXBYT**

Obtains the ASCII or EBCDIC character equivalent of a decimal integer value.

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**ITONUM**

Converts a large binary integer in a non-FOCUS data source to double-precision format.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

**ITOPACK**

Converts a large binary integer in a non-FOCUS data source to packed-decimal format.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **ITOE**

Converts a number in numeric format to zoned format.

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **PCKOUT**

Writes a packed number of variable length to an extract file.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **UFMT**

Converts characters in alphanumeric field values to hexadecimal representation.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

## **Numeric Functions**

The following functions perform calculations on numeric constants or fields. For details, see Chapter 10, *Numeric Functions*.

### **ABS**

Returns the absolute value of a number.

Available Operating Systems: All

Available Languages: reporting, Maintain

### **ASIS**

Distinguishes between a blank and a zero in Dialogue Manager.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX

Available Languages: reporting

### **BAR**

Produces a horizontal bar chart.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **CHKPCK**

Validates the data in a field described as packed format.

Available Operating Systems: All

Available Languages: reporting, Maintain

**DMOD, FMOD, and IMOD**

Calculate the remainder from a division.

Available Operating Systems: All

Available Languages: reporting, Maintain

**EXP**

Raises the number "e" to a specified power.

Available Operating Systems: All

Available Languages: reporting, Maintain

**EXPN**

Evaluates a number expressed in scientific notation.

Available Operating Systems: AS/400, OS/390

Available Languages: reporting

**FMLINFO**

Returns the FOR value associated with each row in an FML report.

Available Operating Systems: All

Available Languages: reporting

**INT**

Returns the integer component of a number.

Available Operating Systems: All

Available Languages: reporting, Maintain

**LOG**

Returns the natural logarithm of a number.

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

**MAX and MIN**

Return the maximum or minimum value, respectively, from a list of values.

Available Operating Systems: All

Available Languages: reporting, Maintain

**NORMSDST and NORMSINV**

Perform calculations on a standard normal distribution curve.

Available Operating Systems: All

Available Languages: reporting



### **PRDNOR and PRDUNI**

Generate reproducible random numbers.

Available Operating Systems: All

Available Languages: reporting, Maintain

### **RDNORM, and RDUNIF**

Generate random numbers.

Available Operating Systems: All

Available Languages: reporting, Maintain

### **SQRT**

Calculates the square root of a number.

Available Operating Systems: All

Available Languages: reporting, Maintain

## **System Functions**

The following functions call the operating system to obtain information about the operating environment or to use a system service. For details, see Chapter 11, *System Functions*.

### **FEXERR**

Retrieves an Information Builders error message.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

### **FINDMEM**

Determines if a specific member of a partitioned data set (PDS) exists in batch processing.

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

### **GETPDS**

Determines if a specific member of a partitioned data set (PDS) exists, and if it does, returns the PDS name.

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

### **GETUSER**

Retrieves the ID of the connected user.

Available Operating Systems: All

Available Languages: reporting, Maintain

## *Types of Functions*

### **HHMMSS**

Retrieves the current time from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

### **MVSDYNAM**

Transfers a FOCUS DYNAM command to the DYNAM command processor.

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

### **TODAY**

Retrieves the current date from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

---

---

## CHAPTER 3

# Accessing and Calling a Function

### Topics:

- Calling a Function
- Supplying an Argument in a Function
- Calling a Function From a DEFINE, COMPUTE, or VALIDATE Command
- Calling a Function From a Dialogue Manager Command
- Calling a Function From Another Function
- Calling a Function in WHERE or IF Criteria
- Calling a Function in WHEN Criteria
- Calling a Function From a RECAP Command
- Storing and Accessing an External Function
- Dynamic Language Environment Support

The following topics describe the considerations for supplying arguments in a function, and explain how to use a function in a command and access functions stored externally.

## Calling a Function

---

You can call a function from a COMPUTE, DEFINE, or VALIDATE command; a Dialogue Manager command; a Financial Modeling Language (FML) command; or a Maintain command. A function is called with the function name, arguments, and, for external functions, an output field.

For details on external functions see *Types of Functions* in Chapter 2.

Some Maintain-specific functions require that the MNTUWS function library be retrieved when calling the function. For functions that require this, it is specified in the detailed information for that function. For details on retrieving the MNTUWS library, see *How to Access the Maintain MNTUWS Function Library* on page 3-4.

### **Syntax**

#### **How to Call a Function**

```
function(arg1, arg2, ... [outfield])
```

where:

*function*

Is the name of the function.

*arg1*, *arg2*, ...

Are the arguments.

*outfield*

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This argument is required only for external functions.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Syntax**    **How to Store Output in a Field**

```
COMPUTE field/fmt = function(input1, input2,... [outfield]);
```

or

```
DEFINE FILE file  
field/fmt = function(input1, input2,... [outfield]);
```

or

```
-SET &var = function(input1, input2,... [outfield]);
```

where:

**DEFINE**

Creates a virtual field that may be used in a request as though it is a real data source field.

**COMPUTE**

Calculates one or more temporary fields in a request. The field is calculated after all records have been selected, sorted, and summed.

*field*

Is the field that contains the result.

*file*

Is the file in which the virtual field is created.

*var*

Is the variable that contains the result.

*fmt*

Is the format of the field that contains the result.

*function*

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

*input1*, *input2*,...

Are the input arguments, which are data values or fields used in function processing. For more information about arguments, see *Supplying an Argument in a Function* on page 3-4.

*outfield*

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This argument is required only for external functions.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## **Syntax**      **How to Access the Maintain MNTUWS Function Library**

Place the following statement directly after the MAINTAIN command at the top of your procedure:

```
MODULE IMPORT (MNTUWS);
```

## **Supplying an Argument in a Function**

---

When supplying an argument in a function, you must understand which types of arguments are acceptable, the formats and lengths for these arguments, and the number and order of these arguments.

### **Argument Types**

The following are acceptable arguments for a function:

- Numeric constant, such as 6 or 15.
- Date constant, such as 022802.
- Date in alphanumeric, numeric, date, or AnV format format.
- Alphanumeric literal, such as STEVENS or NEW YORK NY. A literal must be enclosed in single quotation marks.
- Number in alphanumeric format.
- Field name, such as FIRST\_NAME or HIRE\_DATE. A field can be a data source field or temporary field. The field name can be up to 66 characters long or a qualified field name, unique truncation, or alias.
- Expression, such as a numeric, date, or alphanumeric expression. An expression can use arithmetic operators and the concatenation sign (). For example, the following are valid expressions:

```
CURR_SAL * 1.03
```

and

```
FN || LN
```

- Dialogue Manager variable, such as &CODE or &DDNAME.
- Format of the output value enclosed in single quotation marks.
- Another function.
- Label or other row or column reference (such as R or E), or name of another RECAP calculation, when the function is called in an FML RECAP command.

## Argument Formats

Depending on the function, an argument can be in alphanumeric, numeric, or date format. If you supply an argument in the wrong format, you will cause an error or the function will not return correct data. The following are the types of argument formats:

- **Alphanumeric argument.** An alphanumeric argument is stored internally as one character per byte. An alphanumeric argument can be a literal, an alphanumeric field, a number or date stored in alphanumeric format, an alphanumeric expression, or the format of an alphanumeric field. A literal is enclosed in single quotation marks, except when specified in operating systems that support Dialogue Manager RUN commands (for example, -MVS RUN).
- **Numeric argument.** A numeric argument is stored internally as a binary or packed number. A numeric argument includes integer (I), floating-point single-precision (F), floating-point double-precision (D), and packed decimal (P) formats. A numeric argument can be a numeric constant, field, or expression, or the format of a numeric field.

All numeric arguments are converted to floating-point double-precision format when used with a function, but results are returned in the format specified for the output field.

- **Date argument.** A date argument can be in either alphanumeric, numeric, or date format. The list of arguments for the individual function will specify what type of format the function accepts. A date argument can be a date in alphanumeric, numeric, or date format; a date field or expression; or the format of a date field.

If you supply an argument with a two-digit year, the function assigns a century based on the DATEFNS, YRTHRESH, and DEFCENT parameter settings.

## **Argument Length**

An argument is passed to a function by reference, meaning that the memory location of the argument is passed. No indication of the length of the argument is given.

You must supply the argument length for alphanumeric strings. Some functions require a length for the input and output arguments (for example, SUBSTR), and others use one length for both arguments (for example, UPCASE).

Be careful to ensure that all lengths are correct. Providing an incorrect length can cause incorrect results:

- If the specified length is shorter than the actual length, a subset of the string is used. For example, passing the argument 'ABCDEF' and specifying a length of 3 causes the function to process a string of 'ABC'.
- If the specified length is too long, whatever is in memory up to that length is included. For example, passing an argument of 'ABC' and specifying a length of 6 causes the function to process a string beginning with 'ABC' plus the three characters in the next three positions of memory. Depending on memory utilization, the extra three characters could be anything.

Some operating system routines are very sensitive to incorrectly specified lengths and read them into incorrectly formatted memory areas.

## **Number and Order of Arguments**

The number of arguments required varies according to each function. Functions supplied by Information Builders may require up to six arguments. User-written subroutines may require a maximum of 28 arguments including the output argument. If a function requires more than 28 arguments, you must use two or more calls to pass the arguments to the function.

Arguments must be specified in the order shown in the syntax of each function. The required order varies according to the function.



## Verifying Function Parameters

The USERFCHK setting controls the level of verification applied to DEFINE FUNCTION and Information-Builders-supplied function arguments. It does not affect verification of the number of parameters; the correct number must always be supplied.

USERFCHK is not supported from Maintain.

Functions typically expect parameters to be a specific type or have a length that depends on the value of another parameter. It is possible in some situations to enforce these rules by truncating the length of a parameter and, therefore, avoid generating an error at run-time.

The level of verification and possible conversion to a valid format performed depends on the specific function. The following two situations can usually be converted satisfactorily:

- If a numeric parameter specifies a maximum size for an alphanumeric parameter, but the alphanumeric string supplied is longer than the specified size, the string can be truncated.
- If a parameter supplied as a numeric literal specifies a value larger than the maximum size for a parameter, it can be reduced to the proper value.

### *Syntax*

#### How to Enable Parameter Verification

Parameter verification can be enabled only for DEFINE FUNCTIONS and functions supplied by Information Builders. If your site has a locally written function with the same name as an Information-Builders-supplied function, the USERFNS setting determines which function will be used:

```
SET USERFNS= {SYSTEM|LOCAL}
```

where:

##### SYSTEM

Gives precedence to functions supplied by Information Builders. SYSTEM is the default setting. This setting is required in order to enable parameter verification.

##### LOCAL

Gives precedence to locally written functions. Parameter verification is not performed with this setting in effect.

## **Syntax**    **How to Control Function Parameter Verification**

Issue the following command in FOCPARM, FOCPROF, on the command line, in a FOCEXEC, or in an ON TABLE command. Note that the USERFNS=SYSTEM setting must be in effect

```
SET USERFCHK = setting
```

where:

*setting*

Can be one of the following:

ON is the default value. Verifies parameters in requests, but does not verify parameters for functions used in Master File DEFINES. If a parameter has an incorrect length, an attempt is made to fix the problem. If such a problem cannot be fixed, an error message is generated and the evaluation of the affected expression is terminated.

Note that if a parameter provided is the incorrect type, verification fails and processing terminates.

Because parameters are not verified for functions specified in a Master File, no errors are reported for those functions until the DEFINE field is used in a subsequent request when, if a problem occurs, the following message is generated:

```
(FOC003) THE FIELDNAME IS NOT RECOGNIZED
```

OFF does not verify parameters except in the following cases:

- If a parameter that is too long would overwrite the memory area in which the computational code is stored, the size is automatically reduced without issuing a message.
- If an alphanumeric parameter is too short, it is padded with blanks to the correct length.

FULL is the same as ON, but also verifies parameters for functions used in Master File DEFINES.

Note that if a parameter provided is the incorrect type, verification fails and processing terminates.

ALERT verifies parameters in a request without halting execution when a problem is detected. It does not verify parameters for functions used in Master File DEFINES. If a parameter has an incorrect length and an attempt is made to fix the problem behind the scenes, the problem is corrected with no message. If such a problem cannot be fixed, a warning message is generated. Execution then continues as though the setting were OFF, but the results may be incorrect.

Note that if a parameter provided is the incorrect type, verification fails and processing terminates.

**Example Verifying Parameters With Correctable Errors**

The following request uses SUBSTR to extract the substring that starts in position 6 and ends in position 14 of the TITLE field. The fifth argument specifies a substring length (500) that is too long (it should be no longer than 9):

```
SET USERFCHK = ON
TABLE FILE MOVIES
PRINT TITLE
COMPUTE
  NEWTITLE/A9 = SUBSTR(39, TITLE, 6 ,14, 500, NEWTITLE);
WHERE CATEGORY EQ 'CHILDREN'
END
```

When the request is executed with USERFCHK=ON or OFF, the incorrect length is corrected and the request continues processing:

TITLE	NEWTITLE
-----	-----
SMURFS, THE	S, THE
SHAGGY DOG, THE	Y DOG, TH
SCOOBY-DOO-A DOG IN THE RUFF	Y-DOO-A D
ALICE IN WONDERLAND	IN WONDE
SESAME STREET-BEDTIME STORIES AND SONGS	E STREET-
ROMPER ROOM-ASK MISS MOLLY	R ROOM-AS
SLEEPING BEAUTY	ING BEAUT
BAMBI	

**Example Verifying Parameters With Uncorrectable Errors**

The following request has an incorrect data type in the last argument to SUBSTR. This parameter should specify an alphanumeric field or format for the extracted substring:

```
SET USERFCHK = ON
TABLE FILE MOVIES
PRINT TITLE
COMPUTE
  NEWTITLE/F9 = SUBSTR(39, TITLE, 6 ,14, 500, 'F9');
WHERE CATEGORY EQ 'CHILDREN'
END
```

- When the request is executed with USERFCHK=ON, a message is produced and the request terminates:

```
ERROR AT OR NEAR LINE      5  IN PROCEDURE USERFC3 FOCEXEC
(FOC279) NUMERIC ARGUMENTS IN PLACE WHERE ALPHA ARE CALLED FOR
(FOC009) INCOMPLETE REQUEST STATEMENT
UNKNOWN FOCUS COMMAND  WHERE
      BYPASSING TO END OF COMMAND
```

- When the request is executed with USERFCHK=OFF, no verification is done and no message is produced. The request executes and produces incorrect results. In some environments, this type of error may cause abnormal termination of the application:

DIRECTOR	TITLE	NEWTITLE
-----	-----	-----
	SMURFS, THE	*****
BARTON C.	SHAGGY DOG, THE	*****
	SCOOBY-DOO-A DOG IN THE RUFF	*****
GEROMINI	ALICE IN WONDERLAND	1
	SESAME STREET-BEDTIME STORIES AND SONGS	-265774
	ROMPER ROOM-ASK MISS MOLLY	*****
DISNEY W.	SLEEPING BEAUTY	*****
DISNEY W.	BAMBI	0

## Calling a Function From a DEFINE, COMPUTE, or VALIDATE Command

---

You can call a function from a DEFINE command or Master File attribute, a COMPUTE command, or a VALIDATE command.

### **Syntax** How to Call a Function From a COMPUTE, DEFINE, or VALIDATE Command

```
DEFINE [FILE filename]
tempfield[/format] = function(input1, input2, input3, ... [outfield]);
COMPUTE
tempfield[/format] = function(input1, input2, input3, ... [outfield]);
VALIDATE
tempfield[/format] = function(input1, input2, input3, ... [outfield]);
```

where:

*filename*

Is the data source being used.

*tempfield*

Is the temporary field created by the DEFINE or COMPUTE command. This is the same field specified in *outfield*. If the function call supplies the format of the output value in *outfield*, the format of the temporary field must match the *outfield* argument.

*format*

Is the format of the temporary field. The format is required if it is the first time the field is created; otherwise, it is optional. The default value is D12.2.

*function*

Is the name of the function.

*input1, input2, input3...*

Are the arguments.

*outfield*

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This is required only for external functions.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## Calling a Function From a Dialogue Manager Command

---

You can call a function with Dialogue Manager in the following ways:

- From a -SET command, storing the result of a function in a variable. For details, see *Assigning the Result of a Function to a Variable* on page 3-11.
- From an -IF command. For details, see *Calling a Function in WHERE or IF Criteria* on page 3-17.
- From an operating system -RUN command. For details, see *Calling a Function From an Operating System RUN Command* on page 3-15.

Dialogue Manager converts a numeric argument to double-precision format. This occurs when the value of the argument is numeric; this is not affected by the format expected by the function. This means you must be careful when supplying arguments for a function in Dialogue Manager.

If the function expects an alphanumeric string and the input is a numeric string, incorrect results will occur because of conversion to floating-point 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.

## Assigning the Result of a Function to a Variable

You can store the result of a function in a variable with the -SET command.

A Dialogue Manager variable contains only alphanumeric data. If a function returns a numeric value to a Dialogue Manager variable, the value is truncated to an integer and converted to alphanumeric format before being stored in the variable.

## **Syntax**      **How to Assign the Result of a Function to a Variable**

```
-SET &variable = function(arg1, arg2[.LENGTH],..., 'format');
```

where:

*variable*

Is the variable to which the result will be assigned.

*function*

Is the function.

*arg1, arg2*

Are the function's arguments.

*.LENGTH*

Returns the length of the variable. If a function requires the length of a character string as an input argument, you can prompt for the character string and determine the length with the *.LENGTH* suffix.

*format*

Is the format of the result enclosed in single quotation marks. You cannot specify a Dialogue Manager variable for the output argument unless you use the *.EVAL* suffix; however, you can specify a variable for an input argument.

## **Example**      **Calling a Function From a -SET Command**

AYMD adds 14 days to the value of &INDATE. The &INDATE variable is previously set in the procedure in the six-digit year-month-day format.

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

The format of the output date is a six-digit integer (I6). Although the format 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.

## Branching Based on a Function's Result

You can branch based on a function's result by calling a function from a Dialogue Manager -IF command.

If a branching command spans more than one line, continue it on the next line by placing a dash (-) in the first column.

### **Syntax** How to Branch Based on a Function's Result

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

where:

*function*

Is the function.

*args*

Are the arguments.

*relation*

Is an operator that determines the relationship between the function and expression, for example, EQ or LE.

*expression*

Is a value, logical expression, or function. Do not enclose a literal in single quotation marks unless it contains a comma or embedded blank.

*label1, label2*

Are user-defined names up to 12 characters long. Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use a word that can be confused with a function, or an arithmetic or logical operation.

The *label* text can precede or follow the -IF criteria in the procedure.

`ELSE GOTO`

Passes control to *label2* when the -IF test fails.

### **Example**    **Branching Based on a Function's Result**

The result of the AYMD function provides a condition for a -IF test. One of two requests is executed, depending on the function's result :

```
-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;
7. -TYPE LONG PROJECT
   -*EX LONGPROJ
   -RUN
   -GOTO EXIT
8. -EARLY
   -TYPE SHORT PROJECT
   -*EX SHRTPROJ
   -RUN
   -GOTO EXIT
   -EXIT
```

The procedure processes as follows:

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



## Calling a Function From an Operating System RUN Command

You can call a function that contains only alphanumeric arguments from a Dialogue Manager -CMS RUN, -TSO RUN, or -MVS RUN command. This type of function performs a specific task but typically does not return a value.

If a function requires an argument in numeric format, you must first convert it to floating-point double-precision format using the ATODBL function because, unlike the -SET command, an operating system RUN command does not automatically convert a numeric argument to double-precision.

### Syntax

#### How to Call a Function From an Operating System -RUN Command

```
{-CMS|-TSO|-MVS} RUN function, input1, input2, ... [,&output]
```

where:

```
-CMS|-TSO|-MVS
```

Is the operating system.

```
function
```

Is the name of the function.

```
input1, input2,...
```

Are the arguments. Separate the function name and each argument with a comma. Do not enclose an alphanumeric literal in single quotation marks. If a function requires the length of a character string as an argument, you can prompt for the character string, then use the .LENGTH suffix to test the length.

```
&output
```

Is a Dialogue Manager variable. Include this argument if the function returns a value; otherwise, omit it. If you specify an output variable, you must pre-define its length using a -SET command.

For example, if the function returns a value that is eight bytes long, define the variable with eight characters enclosed in single quotation marks before the function call:

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

### Example

#### Calling a Function From an Operating System -RUN Command

The following calls the CHGDAT function from a -CMS RUN command:

```
-SET &RESULT = '12345678901234567';
-CMS RUN CHGDAT, YYMD., MXDYY, &YYMD, &RESULT
-TYPE &RESULT
```

## Calling a Function From Another Function

---

A function can be an argument for another function.

### **Syntax** How to Call a Function From Another Function

```
field = function(arguments,] function2[arguments2,] arguments);
```

where:

*field*

Is the field that contains the result of the function.

*function*

Is a function.

*arguments*

Are arguments for *function*.

*function2*

Is the function that is an argument for *function*.

*arguments2*

Are arguments for *function2*.

### **Example** Calling a Function From Another Function

In the following, the AYMD function is an argument for the YMD function:

```
-SET &DIFF = YMD(&YYMD, AYMD(&YYMD, 4, 'I8'));
```

## Calling a Function in WHERE or IF Criteria

---

You can call a function in WHERE or IF criteria. When you do this, the output value of the function is compared against a test value.

### **Syntax** How to Call a Function in WHERE Criteria

WHERE *function relation expression*

where:

*function*

Is a function.

*relation*

Is an operator that determines the relationship between the function and expression, for example, EQ or LE.

*expression*

Is a constant, field, or function. A literal must be enclosed in single quotation marks.

### **Syntax** How to Call a Function in IF Criteria

WHERE *function relation value*

where:

*function*

Is a function.

*relation*

Is an operator that determines the relationship between the function and expression, for example, EQ or LE.

*value*

Is a constant. In a DEFINE or COMPUTE command, the value must be enclosed in single quotation marks.

### Example Calling a Function in WHERE Criteria

The SUBSTR function extracts the first two characters of LAST\_NAME as a substring, and the request prints an employee's name and salary if the substring is MC.

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME CURR_SAL
WHERE SUBSTR(15, LAST_NAME, 1, 2, 2, 'A2') IS 'MC';
END
```

The output is:

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

### Using a Calculation or Compound IF Command

You must specify the format of the output value in a calculation or compound IF command. There are two ways to do this:

- Pre-define the format within a separate command. In the following example, the AMOUNT field is pre-defined with the format D8.2 and the function returns a value to the output field AMOUNT. The IF command tests the value of AMOUNT and stores the result in the calculated value, AMOUNT\_FLAG.

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

- Supply the format as the last argument in the function call. In the following example, the command tests the returned value directly. This is possible because the function defines the format of the returned value (D8.2).

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

## Calling a Function in WHEN Criteria

---

You can call a function in WHEN criteria as part of a Boolean expression.

### **Syntax** How to Call a Function in WHEN Criteria

```
WHEN({function|value} relation {function|value});
```

or

```
WHEN NOT(function)
```

where:

*function*

Is a function.

*value*

Is a value or logical expression.

*relation*

Is an operator that determines the relationship between the value and function, for example, LE or GT.

### **Example** Calling a Function in WHEN Criteria

This request checks the values in LAST\_NAME against the result of the CHKFMT function. When a match occurs, the request prints a sort footing.

```
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 output is:

```
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
```

## Calling a Function From a RECAP Command

---

You can call a function in an FML RECAP command.

### **Syntax** How to Call a Function From a RECAP Command

```
RECAP name[(n) | (n,m) | (n,m,i)] [/format1] =  
function(input1, ..., ['format2']);
```

where:

*name*

Is the name of the calculation.

*n*

Displays the value in the column number specified by *n*. If you omit the column number, the value appears in all columns.

*n,m*

Displays the value in all columns beginning with the column number specified by *n* and ending with the column number specified by *m*.

*n,m,i*

Displays the value in the columns beginning with the column number specified by *n* and ending with the column number specified by *m* by the interval specified by *i*. For example, if *n* is 1, *m* is 5, and *i* is 2, the value displays in columns 1, 3, and 5.

*format1*

Is the format of the calculation. The default value is the format of the report column.

*function*

Is the function.

*input1,...*

Are the input arguments, which can include numeric constants, alphanumeric literals, row and column references (R notation, E notation, or labels), and names of other RECAP calculations.

*format2*

Is the format of the output value enclosed in single quotation marks. If the calculation's format is larger than the column width, the value displays in that column as asterisks.

### **Example** Calling a Function in a RECAP Command

This request sums the AMOUNT field for account 1010 using the label CASH, account 1020 using the label DEMAND, and account 1030 using the label TIME. The MAX function displays the maximum value of these accounts.

```
TABLE FILE LEDGER
SUM AMOUNT FOR ACCOUNT
1010 AS 'CASH ON HAND'      LABEL CASH   OVER
1020 AS 'DEMAND DEPOSITS'  LABEL DEMAND OVER
1030 AS 'TIME DEPOSITS'    LABEL TIME   OVER
BAR                          OVER
RECAP MAXCASH = MAX(CASH, DEMAND, TIME); AS 'MAX CASH'
END
```

The output is:

	AMOUNT
	-----
CASH ON HAND	8,784
DEMAND DEPOSITS	4,494
TIME DEPOSITS	7,961
	-----
MAX CASH	8,784

## Storing and Accessing an External Function

---

Internal functions are built in and do not require additional work to access. External functions are stored in load libraries from which they must be retrieved. The way these external functions are accessed is determined by your platform. These techniques may not have to be used every time a function is accessed. Access to a load library may be set only once at the time of installation.

You can also access private user-written subroutines. If you have a private collection of subroutines (that is, you created your own or use customized subroutines), do not store them in the function library. Store them separately to avoid overwriting them whenever your site installs a new release. For details on creating a subroutine, see Appendix A, *Creating a Subroutine*.

### Storing and Accessing a Function on OS/390

On OS/390, load libraries are partitioned data sets containing link-edited modules. These libraries are stored as EDALIB.LOAD or FUSELIB.LOAD. In addition, your site may have private subroutine collections stored in separate load libraries. If so, you need to allocate those libraries.

#### **Procedure** How to Allocate a Load Library in OS/390 Batch

To use a function stored as a load library, allocate the load library to ddname USERLIB in your JCL or CLIST.

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

#### **Example** Allocating the Load Library BIGLIB.LOAD in OS/390 Batch (JCL)

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

#### **Procedure** How to Allocate a Load Library in TSO

Allocate the load library to ddname USERLIB using the ALLOCATE command. You can issue the ALLOCATE command:

- In TSO before entering a FOCUS session.
- Before executing a request in a FOCUS session.
- In your PROFILE FOCEXEC.

If you are in a FOCUS session, you can also use the DYNAM ALLOCATE command.



**Syntax**    **How to Allocate a Load Library**

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

or

```
DYNAM ALLOC FILE USERLIB DA lib SHR
```

where:

**MVS|TSO**

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

**USERLIB**

Is the ddname to which you allocate a load library.

*lib1 lib2 lib3...*

Are the names of the load libraries, concatenated to ddname USERLIB.

**Example**    **Allocating the FUSELIB.LOAD Load Library**

```
TSO ALLOC FILE(USERLIB) DSN('MVS.FUSELIB.LOAD') SHR
```

or

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

**Example**    **Concatenating a Load Library to USERLIB in TSO**

Suppose a report request calls two functions: BENEFIT stored in library SUBLIB.LOAD, and EXCHANGE stored in library BIGLIB.LOAD. To concatenate the BIGLIB and SUBLIB load libraries in the allocation for ddname USERLIB, issue the following commands:

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

The load libraries are searched in the order in which they are specified in the ALLOCATE command.

**Example**    **Concatenating a Load Library to SETPLIB in Batch (JCL)**

Concatenate the load library to the ddname STEPLIB in your JCL:

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

## Storing and Accessing a Function on UNIX

No extra work is required.

## Storing and Accessing a Function on VM/CMS

On VM/CMS, supplied functions are stored as one of the following:

- Load library FUSELIB LOADLIB.
- Text library FUSELIB TXTLIB. A text library is composed of multiple text files called members. Functions can be stored as members of one or more text libraries. The file type for text libraries is TXTLIB.
- Text files. For a function stored as a text file in VM/CMS, the access method is automatic. When your request calls the function, the attached disks are searched in alphabetical order, provided that you have proper authorization.

The name of a text file must match the function name. The file type is TEXT. For example, the EXCHANGE function stored as a text file has the file identifier:

`EXCHANGE TEXT`

In addition to the preceding libraries and files, your site may have private collections of subroutines stored in separate libraries or text files.

### **Reference** Search Sequence on VM/CMS

The standard VM/CMS search sequence applies to functions:

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

### Searching a Function Library in VM/CMS

To search for a function stored in a load or text library, issue the CMS GLOBAL command. You can issue the GLOBAL command:

- Before entering FOCUS.
- In a profile.
- From a procedure.

You must also specify a system library for a function written in a language such as COBOL or PL/I, and for a function that calls a system function. FUSELIB functions do not require any other system libraries.

If you issue two GLOBAL commands of the same type, 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.

If you have a private subroutine collection, specify the function library or libraries in the GLOBAL command in addition to the FUSELIB library.

**Note:** FUSELIB functions now reside in FUSELIB LOADLIB (rather than in a TXTLIB). Issuing GLOBAL TXTLIB FUSELIB still works because the TXTLIB still exists. However, VM/CMS loads supplied functions from the LOADLIB before searching the TXTLIB.

### **Syntax**      **How to Search a Function Library**

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

where:

CMS

Is required if you issue the GLOBAL command from a procedure.

LOADLIB

Indicates the library is a load library.

TXTLIB

Indicates the library is a text library.

```
library1 library2 library3...
```

Are the names of the libraries containing the functions. The maximum number of libraries is 63.

### **Syntax**      **How to List Function Libraries Specified by the GLOBAL Command**

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

where:

LOADLIB

Indicates the library is a load library.

TXTLIB

Indicates the library is a text library.

### **Example**      **Searching a Function Library**

The following command, issued in the global profile, accesses the FUSELIB load library:

```
CMS GLOBAL LOADLIB FUSELIB
```

### **Example**    **Searching Multiple Function Libraries**

The following command, issued in a procedure, accesses the SUBLIB and BIGLIB libraries:

```
CMS GLOBAL TXTLIB SUBLIB BIGLIB
```

### **Adding or Deleting a Function Library**

The GLOBAL library list automatically contains the FUSELIB function library. To add or delete private subroutine libraries, use two CMS EXECs, FOCADLIB or FOCDELIB.

Before you add LOADLIBs to the GLOBAL library list, the existing list is saved. Then the required and optional LOADLIBs are added in front of any libraries you may have specified. After a request, the prior GLOBAL environment is restored.

Prior entries can be retained in the GLOBAL library list and new entries added by using the FOCADLIB EXEC. To delete entries while maintaining others in the list, use the FOCDELIB EXEC. For both FOCADLIB and FOCDELIB, the output from the EXEC is the return code of the GLOBAL command. FOCADLIB and FOCDELIB must be found in the VM/CMS search sequence (A–Z).

### **Syntax**    **How to Add or Delete a Function Library**

```
CMS EX {FOCADLIB|FOCDELIB} libtype lib1 [lib2 lib3...] [(QUIET )
```

where:

**FOCADLIB**

Adds libraries to the beginning of the GLOBAL library list.

**FOCDELIB**

Deletes libraries from the GLOBAL library list.

*libtype*

Is the library type, for example, LOADLIB or TXTLIB.

*lib1 lib2 lib3...*

Are the names of the libraries to be added or deleted.

**QUIET**

Suppresses messages from the GLOBAL command. The open parenthesis is required.

## Dynamic Language Environment Support

---

The IBMLE parameter setting controls the LE run-time environment by identifying which LE libraries to load. By default, the C and COBOL libraries are loaded. On OS/390, issue the SET IBMLE command in order to access LE-compiled PL/I or FORTRAN user-written subroutines. On VM/CMS, the setting has no effect; LE and non-LE versions of functions in all HLLs work properly regardless of the IBMLE setting. On OS/390, non-LE versions of functions work properly regardless of the IBMLE setting.

Loading extra libraries uses some additional memory below the line. Once this memory has been used, it cannot be released during the FOCUS session. Therefore, you can control this memory use by waiting to issue the SET IBMLE command until you need to execute a FOCEXEC that makes a call to an LE-compliant PL/I or FORTRAN function.

### *Syntax*

### **How to Control the LE Run-Time Environment**

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

where:

#### OFF

Loads the libraries for LE-compiled C and COBOL functions. This value is the default.

#### ON

Adds the libraries for LE-compiled PL/I subroutines to the C and COBOL libraries. Once the ON setting has been established, you cannot issue the OFF setting. You can issue the ALL setting to add libraries for LE-compiled FORTRAN functions.

#### ALL

Adds the libraries for LE-compliant FORTRAN and PL/I functions (if they are not already loaded) to the C and COBOL libraries. Once the ALL setting has been established, you cannot issue the OFF or ON setting.



---

---

## CHAPTER 4

### Character Functions

Character functions manipulate alphanumeric fields and character strings.

In addition to the functions discussed in this topic, there are character functions that are available only in the Maintain language. For information on these functions, see Chapter 5, *Maintain-specific Character Functions*.

**Topics:**

- ARGLEN: Measuring the Length of a Character String
- ASIS: Distinguishing Between a Space and a Zero
- BITSON: Determining If a Bit Is On or Off
- BITVAL: Evaluating a Bit String as a Binary Integer
- BYTVAL: Translating a Character to a Decimal Value
- CHKFMT: Checking the Format of a Character String
- CTRAN: Translating One Character to Another
- CTRFLD: Centering a Character String
- EDIT: Extracting or Adding Characters
- GETTOK: Extracting a Substring (Token)
- LCWORD: Converting a Character String to Mixed Case
- LJUST: Left-Justifying a Character String
- LOCASE: Converting Text to Lowercase
- OVLAY: Overlaying a Character String
- PARAG: Dividing Text Into Smaller Lines
- POSIT: Finding the Beginning of a Substring
- RJUST: Right-Justifying a Character String
- SOUNDEX: Comparing Character Strings Phonetically
- SQUEEZ: Reducing Multiple Spaces to a Single Space
- STRIP: Removing a Character From a String
- SUBSTR: Extracting a Substring
- TRIM: Removing Leading and Trailing Occurrences
- UPCASE: Converting Text to Uppercase
- Character Functions for AnV Fields

## ARGLEN: Measuring the Length of a Character String

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The ARGLEN function measures the length of a character string within a field, excluding trailing spaces. The field format in a Master File specifies the length of a field, including trailing spaces.

In Dialogue Manager, you can measure the length of a supplied character string using the .LENGTH suffix.

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

`ARGLEN(inlength, infield, outfield)`

where:

*inlength*

Integer

Is the length of the field containing the character string, or a field that contains the length.

*infield*

Alphanumeric

Is the name of the field containing the character string.

*outfield*

Integer

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.



**Example** **Measuring the Length of a Character String**

ARGLN determines the length of the character string in LAST\_NAME and stores the result in NAME\_LEN:

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

The output is:

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

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

---

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX

Available Languages: reporting

The ASIS function distinguishes between a space and a zero in Dialogue Manager. It differentiates between a numeric string, constant or variable defined as a numeric string (number within single quotation marks) and a field defined simply as numeric. ASIS forces a variable to be evaluated as it is entered rather than be converted to a number. It is used in Dialogue Manager equality expressions only.

**Syntax** **How to Distinguish Between a Space and a Zero**

*ASIS(argument)*

where:

*argument*

Alphanumeric

Is the value to be evaluated. Supply the actual value, the name of a field that contains the value, or an expression that returns the value. An expression can call a function.

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

**Example Distinguishing Between a Space and a Zero**

The first request does not use ASIS. No difference is detected between variables defined as a space 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 next request uses ASIS 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
```

## **BITSON: Determining If a Bit Is On or Off**

---

Available Operating Systems: All

Available Languages: reporting, Maintain

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

### **Syntax**    **How to Determine If a Bit Is On or Off**

`BITSON(bitnumber, string, outfield)`

where:

*bitnumber*

Integer

Is the number of the bit to be evaluated, counted from the left-most bit in the character string.

*string*

Alphanumeric

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

*outfield*

Integer or Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## *BITVAL: Evaluating a Bit String as a Binary Integer*

### **Example** Evaluating a Bit in a Field

BITSON evaluates the 24th bit of LAST\_NAME and stores the result in BIT\_24:

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

The output is:

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

## **BITVAL: Evaluating a Bit String as a Binary Integer**

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The BITVAL function evaluates a string of bits within a character string. The bit string can be any group of bits within the character string and can cross byte and word boundaries. The function evaluates the bit string as a binary integer and returns the corresponding value.

**Syntax**    **How to Evaluate a Bit String**

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

where:

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or a field or variable that contains the character 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 function returns a value of zero.

*number*

Integer

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

*outfield*

Integer

Is the name of the field that contains the binary integer equivalent, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## BYTVAL: Translating a Character to a Decimal Value

### **Example** Evaluating a Bit String

BITVAL evaluates the bits 12 through 20 of LAST\_NAME and stores the result in a field with the format I5:

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

The output is:

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

## BYTVAL: Translating a Character to a Decimal Value

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The BYTVAL function translates a character to the ASCII or EBCDIC decimal value that represents it, depending on the operating system.

### **Syntax** How to Translate a Character

```
BYTVAL(character, outfield)
```

where:

*character*

Alphanumeric

Is the character to be translated. You can specify a field or variable that contains the character, or the character itself enclosed in single quotation marks. If you supply more than one character, the function evaluates the first.

*outfield*

Integer

Is the name of the field that contains the corresponding decimal value, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Translating the First Character of a Field**

BYTVAL translates the first character of LAST\_NAME into its ASCII or EBCDIC decimal value and stores the result in LAST\_INIT\_CODE. Since the input string has more than one character, BYTVAL evaluates the first one.

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

The output on an ASCII platform is:

LAST_NAME	LAST_INIT_CODE
-----	-----
SMITH	83
JONES	74
MCCOY	77
BLACKWOOD	66
GREENSPAN	71
CROSS	67

The output on an EBCDIC platform is:

LAST_NAME	LAST_INIT_CODE
-----	-----
SMITH	226
JONES	209
MCCOY	212
BLACKWOOD	194
GREENSPAN	199
CROSS	195

**Example**    **Returning the EBCDIC Value With Dialogue Manager**

This Dialogue Manager request prompts for a character, then returns the corresponding number. The following reflects the results on the OS/390 platform.

```
-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  
>
```

---

## CHKFMT: Checking the Format of a Character String

Available Operating Systems: All

Available Languages: reporting, Maintain

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

If the mask is shorter than the character string, the function 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 no match with CHKFMT giving the first non-matching position as the result.



**Syntax**    **How to Check the Format of a Character String**

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

where:

*numchar*

Integer

Is the number of characters being compared to the mask.

*string*

Alphanumeric

Is the character string to be checked enclosed in single quotation marks, or a field or variable that contains the character string.

'*mask*'

Alphanumeric

Is the mask, which contains the comparison characters enclosed in single quotation marks.

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

Generic characters are:

A is any letter between A and Z (uppercase or lowercase).

9 is any digit between 0–9.

X is any letter between A–Z or any digit between 0-9.

\$ is any character.

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

*outfield*

Integer

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Checking the Format of a Field**

CHKFMT examines EMP\_ID for nine numeric characters starting with 11 and stores the result in CHK\_ID:

```
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 output is:

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**    **Checking the Format of a Field With MODIFY on OS/390**

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.

```
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 follows:

1. The procedure searches the data source for the ID 111w2. If it does not find this ID, it continues processing the transaction.
2. CHKFMT checks the ID against the mask 99999, which represents five digits.
3. The fourth character in the ID, the letter w, is not a digit. The function returns the value 4 to the BAD\_CHAR field.
4. The VALIDATE command 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.

## CTRAN: Translating One Character to Another

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The CTRAN function translates a character within a character string to another character based on its decimal value. This function is especially useful for changing replacement characters to unavailable characters, or to characters that are difficult to input or unavailable on your keyboard. It can also be used for inputting characters that are difficult to enter when responding to a Dialogue Manager -PROMPT command, such as a comma or apostrophe. It eliminates the need to enclose entries in single quotation marks.

To use CTRAN, you need to know the decimal equivalent of the characters in internal machine representation. Printable EBCDIC or ASCII characters and the decimal equivalents are listed in character charts.

### Syntax

#### How to Translate One Character to Another

*CTRAN(charlen, string, decimal, decvalue, outfield)*

where:

*charlen*

Integer

Is the length in characters of the character string, or a field that contains the length.

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or the field or variable that contains the character string.

*decimal*

Integer

Is the ASCII or EBCDIC decimal value of the character to be translated.

*decvalue*

Integer

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

*outfield*

Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Translating Spaces to Underscores on an ASCII Platform**

CTRAN translates the spaces in ADDRESS\_LN3 (ASCII decimal value 32) to underscores (ASCII decimal value 95), and stores the result in ALT\_ADDR:

```
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 output is:

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 Translating Spaces to Underscores on an EBCDIC Platform**

CTRAN translates the spaces in ADDRESS\_LN3 (EBCDIC decimal value 64) to underscores (EBCDIC decimal value 109) and stores the result in ALT\_ADDR:

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN3 AND COMPUTE
ALT_ADDR/A20 = CTRAN(20, ADDRESS_LN3, 64, 109, ALT_ADDR);
BY EMP_ID
WHERE TYPE EQ 'HSM'
END
```

The output is:

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** Inserting Accented Letter E's With MODIFY

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

If you are using the Hot Screen facility, some characters cannot be displayed. If Hot Screen does not support the character you need, disable Hot Screen with SET SCREEN=OFF and issue the RETYPE command. If your terminal can display the character, the character appears. The display of special characters depends upon your software and hardware; not all special characters may display.

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 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, first name, last name, and department assignment. It requests that you substitute an asterisk (\*) whenever the accented letter È appears in a name.
2. 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. CTRAN converts the asterisks into È's in both the first and last names (ADÈLE MOLIÈRE).

```
***** 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: MOLIÈRE
```

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

### **Example** Inserting Commas With MODIFY

This MODIFY request adds records of new employees to the EMPLOYEE data source. The PROMPT command prompts you for data one field at a time. CTRAN 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 CTRAN into a comma. The equivalent EBCDIC decimal value 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 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. 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. CTRAN 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.
5. This request displays the semicolon converted to a comma:

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

The output is:

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

## CTRFLD: Centering a Character String

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The CTRFLD function centers a character string within a field. The number of leading spaces is equal to or one less than the number of trailing spaces.

CTRFLD is useful for centering the contents of a field and its report column, or a heading that consists only of an embedded field. 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 CTRFLD.

**Limit:** Using CTRFLD in a styled report (StyleSheets feature) generally negates the effect of CTRFLD unless the item is also styled as a centered element. Also, if you are using CTRFLD on a platform for which the default font is proportional, either use a non-proportional font, or issue SET STYLE=OFF before running the request.

**Syntax**      **How to Center a Character String**

`CTRFLD(string, length, outfield)`

where:

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or a field or variable that contains the character string.

*length*

Integer

Is the length of *string* and *outfield* in characters, or a field that contains the length. This argument must be greater than 0. A length less than 0 can cause unpredictable results.

*outfield*

Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**      **Centering a Field**

CTRFLD centers LAST\_NAME and stores the result in CENTER\_NAME:

```
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 output is:

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

## EDIT: Extracting or Adding Characters

---

Available Operating Systems: All

Available Languages: reporting

The EDIT function extracts characters from or adds characters to an alphanumeric string. It can extract a substring from different parts of the parent string, and can also insert characters from a parent string into another substring. For example, it can extract the first two characters and the last two characters of a string to form a single substring.

EDIT works by comparing the characters in a mask to the characters in a source field. When it encounters a nine 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.

EDIT can also convert the format of a field. For more information on converting a field with EDIT, see *EDIT: Converting the Format of a Field* in Chapter 9.

### **Syntax**

#### **How to Extract or Add Characters**

```
EDIT(fieldname, 'mask');
```

where:

*fieldname*

Alphanumeric

Is the source field.

*mask*

Alphanumeric

Is a character string enclosed in single quotation marks.

The length of the mask, excluding any characters other than nine and \$, should be the length of the source field.

## GETTOK: Extracting a Substring (Token)

### **Example** Extracting and Adding a Character to a Field

EDIT extracts the first initial from the FIRST\_NAME field and stores the result in FIRST\_INIT. EDIT also adds dashes to the EMP\_ID field and stores the result in EMPIDEDIT:

```
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 output is:

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

## GETTOK: Extracting a Substring (Token)

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The GETTOK function divides a character string into substrings, called tokens, where a specific character, called a delimiter, occurs in the string. It then returns one of the tokens. GETTOK ignores leading and trailing blanks in the parent character string.

For example, suppose you want to extract the fourth word from a sentence. GETTOK divides the sentence into words using spaces as delimiters, then extracts the fourth word. If the string is not divided by a delimiter, use the PARAG function for this purpose.

**Syntax**    **How to Extract a Substring (Token)**

```
GETTOK(infield, inlen, token, 'delim', outlen, outfield)
```

where:

*infield*

Alphanumeric

Is the field containing the parent character string.

*inlen*

Integer

Is the length of the parent string in characters. If this argument is less than or equal to 0, the function returns spaces.

*token*

Integer

Is the number of the token to extract. If this argument is positive, the tokens are counted from left to right. If this argument is negative, the tokens are counted from right to left. For example -2 extracts the second token from the right. If this argument is 0, the function returns spaces. Leading and trailing null tokens are ignored.

'*delim*'

Alphanumeric

Is the delimiter in the parent string enclosed in single quotation marks. If you specify more than one character, only the first character is used.

**Note:** In Dialogue Manager, to prevent the conversion of a delimiter space character (' ') to a double precision zero, include a non-numeric character after the space (for example, '%'). GETTOK uses only the first character (the space) as a delimiter, while the extra character (%) prevents conversion to double precision.

*outlen*

Integer

Is the maximum size of the token. If this argument is less than or equal to 0, the function 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 that contains the token, or the format of the output value enclosed in single quotation marks. The delimiter is not included in the token.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Extracting a Token From a Field**

GETTOK extracts the last token from ADDRESS\_LN3 and stores the result in LAST\_TOKEN:

```
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 output is:

ADDRESS_LN3	LAST TOKEN (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

---

## LCWORD: Converting a Character String to Mixed Case

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS,

Available Languages: reporting, Maintain

The LCWORD function converts the letters in a character string to mixed case. It converts every alphanumeric character to lowercase except the first letter of each new word and 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.

If LCWORD encounters a number in the character string, it treats it as an uppercase character and continues to convert the following alphabetic characters to lowercase. The result of LCWORD is a word with an initial uppercase character followed by lowercase characters.

There is a version of the LCWORD function that is available only in the Maintain language. For information on this function, see *LCWORD and LCWORD2: Converting a Character String to Mixed Case* in Chapter 5.

**Syntax**    **How to Convert a Character String to Mixed Case**

```
LCWORD(length, string, outfield)
```

where:

*length*

Integer

Is the length in characters of the character string or field to be converted, or a field that contains the length.

*string*

Alphanumeric

Is the character string to be converted enclosed in single quotation marks, or a field or variable containing the character string.

*outfield*

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The length must be greater than or equal to the length of *length*.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting a Character String to Mixed Case**

LCWORD converts the LAST\_NAME field to mixed case and stores the result in 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 output is:

LAST_NAME	MIXED_CASE
-----	-----
STEVENS	Stevens
SMITH	Smith
BANNING	Banning
IRVING	Irving
ROMANS	Romans
MCKNIGHT	Mcknight

## **LJUST: Left-Justifying a Character String**

---

Available Operating Systems: All

Available Languages: reporting

The LJUST function left-justifies a character string within a field. All leading spaces become trailing spaces.

LJUST will not have any visible effect in a report that uses StyleSheets (SET STYLE=ON) unless you center the item.

There is a version of the LJUST function that is available only in the Maintain language. For information on this function, see *LJUST: Left-Justifying a Character String (Maintain)* in Chapter 5.

### **Syntax**

#### **How to Left-Justify a Character String**

*LJUST*(*length*, *string*, *outfield*)

where:

*length*

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length.

*string*

Alphanumeric

Is the character string to be justified, or a field or variable that contains the string.

*outfield*

Alphanumeric

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

In Dialogue Manager, you must specify the format.



**Example Left-Justifying a Field**

LJUST left-justifies the XNAME field and stores the result in YNAME:

```
SET STYLE=OFF

DEFINE FILE EMPLOYEE
XNAME/A25=IF LAST_NAME EQ 'BLACKWOOD' THEN '      ' |LAST_NAME ELSE
'' |LAST_NAME;
YNAME/A25=LJUST(15, XNAME, 'A25');
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME XNAME YNAME
END
```

The output is:

LAST_NAME	XNAME	YNAME
-----	-----	-----
STEVENS	STEVENS	STEVENS
SMITH	SMITH	SMITH
JONES	JONES	JONES
SMITH	SMITH	SMITH
BANNING	BANNING	BANNING
IRVING	IRVING	IRVING
ROMANS	ROMANS	ROMANS
MCCOY	MCCOY	MCCOY
BLACKWOOD	BLACKWOOD	BLACKWOOD
MCKNIGHT	MCKNIGHT	MCKNIGHT
GREENSPAN	GREENSPAN	GREENSPAN
CROSS	CROSS	CROSS

## LOCASE: Converting Text to Lowercase

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The LOCASE function converts alphanumeric text to lowercase. It is useful for converting input fields from FIDEL CRTFORMs and non-FOCUS applications to lowercase.

### **Syntax**    **How to Convert Text to Lowercase**

`LOCASE(length, string, outfield)`

where:

*length*

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length. The length must be greater than 0 and the same for both arguments; otherwise, an error occurs.

*string*

Alphanumeric

Is the character string to be converted in single quotation marks, or a field or variable that contains the string.

*outfield*

Alphanumeric

Is the name of the field in which to store the result, or the format of the output value enclosed in single quotation marks. The field name can be the same as *string*.

In Dialogue Manager, the format must be specified. In Maintain, the name of the field must be specified.

**Example**    **Converting a Field to Lowercase**

LOCASE converts the LAST\_NAME field to lowercase and stores the result in LOWER\_NAME:

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

The output is:

LAST_NAME	LOWER_NAME
-----	-----
SMITH	smith
JONES	jones
MCCOY	mccoy
BLACKWOOD	blackwood
GREENSPAN	greenspan
CROSS	cross

## **OVRLAY: Overlaying a Character String**

---

Available Operating Systems: All

Available Languages: reporting

The OVRLAY function overlays a base character string with a substring. When specified in a MODIFY procedure, the function enables you to edit part of an alphanumeric field without replacing the entire field.

There is a version of the OVRLAY function that is available only in the Maintain language. For information on this function, see *OVRLAY: Overlaying a Character String (Maintain)* in Chapter 5.

**Syntax**      **How to Overlay a Character String**

`OVERLAY(string1, stringlen, string2, sublen, position, outfield)`

where:

*string1*

Alphanumeric

Is the base character string.

*stringlen*

Integer

Is the length in characters of *string1* and *outfield*, or a field that contains the length. If this argument is less than or equal to 0, unpredictable results occur.

*string2*

Alphanumeric

Is the substring that will overlay *string1*.

*sublen*

Integer

Is the length of *string2*, or a field that contains the length. If this argument is less than or equal to 0, the function returns spaces.

*position*

Integer

Is the position in the base string at which the overlay begins. If this argument is less than or equal to 0, the function returns spaces. If this argument is larger than *stringlen*, the function returns the base string.

*outfield*

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. If the overlaid string is longer than the output field, the string is truncated to fit the field.

In Dialogue Manager, you must specify the format.

**Example Replacing Characters in a Character String**

OVERLAY replaces the last three characters of EMP\_ID with CURR\_JOBCODE to create a new security identification code and stores the result in NEW\_ID:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND CURR_JOBCODE AND COMPUTE
NEW_ID/A9 = OVERLAY(EMP_ID, 9, CURR_JOBCODE, 3, 7, NEW_ID);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

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 Overlaying a Character in a String With MODIFY**

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.

```
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 = OVERLAY(FIRST_NAME, 10, F_UP, 1, 1, 'A10');
ON MATCH TYPE "CHANGING FIRST NAME TO <FIRST_NAME "
ON MATCH UPDATE FIRST_NAME
DATA
END
```

## OVRLAY: Overlaying a Character String

The COMPUTE command invokes two functions:

- UPCASE extracts the first letter and converts it to uppercase.
- OVRLAY replaces the original first letter in the name with the uppercase initial.

The procedure processes as:

1. The procedure prompts you from a CRTFORM screen for an employee ID and a first name. Type the following data and press *Enter*:

Enter the employee's ID: 071382660

Enter the first name in lowercase: 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. UPCASE extracts the letter a from alfred and converts it to the letter A.
4. OVRLAY 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 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**

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The PARAG function divides a line of text into smaller lines by marking them with a delimiter. It scans a specific number of characters from the beginning of the line and replaces the last space in the group scanned with the delimiter. It then scans the next group of characters in the line, starting from the delimiter, and replaces the last space in this group with a second delimiter. It repeats this process until reaching the end of the line.

Each group of characters marked off by the delimiter becomes a sub-line. The GETTOK function can then place the sub-lines into different fields. If the function does not find any spaces in the group it scans, it replaces the first character after the group with the delimiter. Therefore, make sure that no word of text is longer than the number of characters scanned (the maximum sub-line length).

If the input lines of text are roughly equal in length, you can keep the sub-lines equal by specifying a sub-line length that evenly divides into the length of the text lines. For example, if the text lines are 120 characters long, divide each of them into two sub-lines of 60 characters or three sub-lines of 40 characters. This technique enables you to print lines of text in paragraph form.

However, if you divide the lines evenly, you may create more sub-lines than you intend. For example, suppose you divide 120-character text lines into two lines of 60 characters maximum, but one line is divided so that the first sub-line is 50 characters and the second is 55. This leaves room for a third sub-line of 15 characters. To correct this, insert a space (using weak concatenation) at the beginning of the extra sub-line, then append this sub-line (using strong concatenation) to the end of the one before it.

## **Syntax**      **How to Divide Text Into Smaller Lines**

`PARAG(length, string, 'delim', subsize, outfield)`

where:

*length*

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length.

*string*

Alphanumeric

Is the text enclosed in single quotation marks, or a field or variable that contains the text.

*delim*

Alphanumeric

Is the delimiter enclosed in single quotation marks. Choose a character that does not appear in the text.

*subsize*

Integer

Is the maximum length of each sub-line.

*outfield*

Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.



**Example**    **Dividing Text Into Smaller Lines**

PARAG divides ADDRESS\_LN2 into smaller lines of not more than ten characters using a comma as the delimiter. It then stores the result in PARA\_ADDR:

```
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 output is:

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.

**POSIT: Finding the Beginning of a Substring**

---

Available Operating Systems: All

Available Languages: reporting

The POSIT function finds the starting position of a substring within a larger string. For example, the starting position of the substring DUCT in the string PRODUCTION is four. If the substring is not in the parent string, the function returns the value 0.

There is a version of the POSIT function that is available only in the Maintain language. For information on this function, see *POSIT: Finding the Beginning of a Substring (Maintain)* in Chapter 5.

**Syntax**      **How to Find the Beginning of a Substring**

`POSIT(parent, inlength, substring, sublength, outfield)`

where:

*parent*

Alphanumeric

Is the parent character string enclosed in single quotation marks, or a field or variable that contains the parent character string.

*inlength*

Integer

Is the length of the parent character string in characters, or a field that contains the length. If this argument is less than or equal to 0, the function returns a 0.

*substring*

Alphanumeric

Is the substring whose position you want to find. This can be the substring enclosed in single quotation marks, or the 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 *inlength*, the function returns a 0.

*outfield*

Integer

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

In Dialogue Manager, you must specify the format.

**Example Finding the Position of a Letter**

POSIT determines the position of the first capital letter I in LAST\_NAME and stores the result in I\_IN\_NAME:

```
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 output is:

LAST_NAME	I_IN_NAME
-----	-----
STEVENS	0
SMITH	3
BANNING	5
IRVING	1
ROMANS	0
MCKNIGHT	5

**RJUST: Right-Justifying a Character String**

---

Available Operating Systems: All

Available Languages: reporting

The RJUST function right-justifies a character string. All trailing blanks become leading blanks. This is useful when you display alphanumeric fields containing numbers.

RJUST does not have any visible effect in a report that uses StyleSheets (SET STYLE=ON) unless you center the item. Also, if you use RJUST on a platform on which StyleSheets are turned on by default, issue SET STYLE=OFF before running the request.

There is a version of the RJUST function that is available only in the Maintain language. For information on this function, see *RJUST: Right-Justifying a Character String (Maintain)* in Chapter 5.

**Syntax**      **How to Right-Justify a Character String**

`RJUST(length, string, outfield)`

where:

*length*

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length. The lengths must be the same to avoid justification problems.

*string*

Alphanumeric

Is the character string, or a field or variable that contains the character string enclosed in single quotation marks.

*outfield*

Alphanumeric

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

In Dialogue Manager, you must specify the format.

**Example**      **Right-Justifying a Field**

RJUST right-justifies the LAST\_NAME field and stores the result in RIGHT\_NAME:

```
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 output is:

LAST_NAME	RIGHT_NAME
SMITH	SMITH
JONES	JONES
MCCOY	MCCOY
BLACKWOOD	BLACKWOOD
GREENSPAN	GREENSPAN
CROSS	CROSS

## SOUNDEX: Comparing Character Strings Phonetically

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The SOUNDEX function searches for a character string phonetically without regard to spelling. It converts character strings to four character codes. The first character must be the first character in the string. The last three characters represent the next three significant sounds in the character string.

To conduct a phonetic search, do the following:

1. Use SOUNDEX to translate data values from the field you are searching for to the phonetic codes.
2. Use SOUNDEX to translate your best guess target string to a phonetic code. Remember that the spelling of your target string need be only approximate; however, the first letter must be correct.
3. Use WHERE or IF criteria to compare the temporary fields created in step 1 to the temporary field created in Step 2.

### **Syntax** How to Compare Character Strings Phonetically

`SOUNDEX(inlength, string, outfield)`

where:

*inlength*

A2

Is the length in characters of *string*, or a field that contains the length. It can be a number enclosed in single quotation marks, or a field containing the number. The number must be from 1 to 99; a number larger than 99 causes the function to return asterisks (\*) as output.

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or a field or variable that contains the character string.

*outfield*

Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Comparing Character Strings Phonetically**

The following request creates three fields:

- PHON\_NAME contains the phonetic code of employee last names.
- PHON\_COY contains the phonetic code of your guess, MICOY.
- PHON\_MATCH contains YES if the phonetic codes match, NO if they do not.

The WHERE criteria selects the last name that matches your best guess.

```
DEFINE FILE EMPLOYEE
PHON_NAME/A4 = SOUNDEX('15', LAST_NAME, PHON_NAME);
PHON_COY/A4 WITH LAST_NAME = SOUNDEX('15', 'MICOY', PHON_COY);
PHON_MATCH/A3 = IF PHON_NAME IS PHON_COY THEN 'YES' ELSE 'NO';
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME
IF PHON_MATCH IS 'YES'
END
```

The output is:

```
LAST_NAME
-----
MCCOY
```

---

## **SQUEEZ: Reducing Multiple Spaces to a Single Space**

Available Operating Systems: All

Available Languages: reporting, Maintain

The SQUEEZ function reduces multiple contiguous spaces within a character string to a single space. The resulting character string has the same length as the original string but is padded on the right with spaces.

**Syntax**    **How to Reduce Multiple Spaces to a Single Space**

```
SQUEEZ(length, string, outfield)
```

where:

*length*

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length.

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or the field that contains the character string.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Reducing Multiple Spaces to a Single Space**

SQUEEZ reduces multiple spaces in the NAME field to a single blank and stores the result in a field with the format A30:

```
DEFINE FILE EMPLOYEE
NAME/A30 = FIRST_NAME | LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT NAME AND COMPUTE
SQNAME/A30 = SQUEEZ(30, NAME, 'A30');
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

NAME		SQNAME
----		-----
MARY	SMITH	MARY SMITH
DIANE	JONES	DIANE JONES
JOHN	MCCOY	JOHN MCCOY
ROSEMARIE	BLACKWOOD	ROSEMARIE BLACKWOOD
MARY	GREENSPAN	MARY GREENSPAN
BARBARA	CROSS	BARBARA CROSS

## STRIP: Removing a Character From a String

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The STRIP function removes all occurrences of a specific character from a string. The resulting character string has the same length as the original string but is padded on the right with spaces.

### **Syntax** How to Remove a Character From a String

`STRIP(length, string, char, outfield)`

where:

*length*

Integer

Is the length in characters of *string* and *outfield*, or a field that contains the length.

*string*

Alphanumeric

Is an alphanumeric string, or the field from which the character will be removed.

*char*

Alphanumeric

Is the character to be removed from the string. This can be an alphanumeric literal enclosed in single quotation marks, or a field that contains the character. If it is a field, the left-most character in the field will be used as the strip character.

**Note:** To remove single quotation marks, use two consecutive quotation marks. You must then enclose this character combination in single quotation marks.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format.



**Example Removing Occurrences of a Character From a String (Reporting)**

STRIP removes all occurrences of a period (.) from the DIRECTOR field and stores the result in a field with the format A17:

```
TABLE FILE MOVIES
PRINT DIRECTOR AND COMPUTE
SDIR/A17 = STRIP(17, DIRECTOR, '.', 'A17');
WHERE CATEGORY EQ 'COMEDY'
END
```

The output is:

DIRECTORS	DIR
-----	----
ZEMECKIS R.	ZEMECKIS R
ABRAHAMS J.	ABRAHAMS J
ALLEN W.	ALLEN W
HALLSTROM L.	HALLSTROM L
MARSHALL P.	MARSHALL P
BROOKS J.L.	BROOKS JL

**Example Removing Single Quotation Marks From a String (Reporting)**

STRIP removes all occurrences of a single quotation mark (') from the TITLE field and stores the result in a field with the format A17:

```
TABLE FILE MOVIES
PRINT TITLE AND COMPUTE
STITLE/A39 = STRIP(39, TITLE, ''', 'A39');
WHERE TITLE CONTAINS '''
END
```

The output is:

TITLE	STITLE
-----	-----
BABETTE'S FEAST	BABETTES FEAST
JANE FONDA'S COMPLETE WORKOUT	JANE FONDAS COMPLETE WORKOUT
JANE FONDA'S NEW WORKOUT	JANE FONDAS NEW WORKOUT
MICKEY MANTLE'S BASEBALLTIPS	MICKEY MANTLES BASEBALL TIPS

**Example Removing Commas From a String (Maintain)**

STRIP removes all occurrences of a comma from the TITLE field:

```
MAINTAIN FILE MOVIES
FOR 10 NEXT MOVIECODE INTO MOVSTK
  WHERE TITLE CONTAINS ', ' ;
COMPUTE I/I2=1;
REPEAT MOVSTK.FOCINDEX
TYPE "TITLE IS: <MOVSTK(I).TITLE"
COMPUTE NOCOMMA/A39=STRIP(39,MOVSTK().TITLE, ', ', NOCOMMA);
TYPE "NEW TITLE IS: <NOCOMMA";
COMPUTE I=I+1
ENDREPEAT
END
```

The output is:

```
TITLE IS: SMURFS, THE
NEW TITLE IS: SMURFS THE
```

---

## **SUBSTR: Extracting a Substring**

Available Operating Systems: All

Available Languages: reporting

The SUBSTR function extracts a substring based on where it begins and its length in the parent string. SUBSTR can vary the position of the substring depending on the values of other fields.

There is a version of the SUBSTR function that is available only in the Maintain language. For information on this function, see *SUBSTR: Extracting a Substring (Maintain)* in Chapter 5.

**Syntax**    **How to Extract a Substring**

`SUBSTR(inlength, parent, start, end, sublength, outfield)`

where:

*inlength*

Integer

Is the length of the parent string in characters, or a field that contains the length.

*parent*

Alphanumeric

Is the parent string enclosed in single quotation marks, or the field containing the parent string.

*start*

Integer

Is the starting position of the substring in the parent string. If this argument is less than one, the function returns spaces.

*end*

Integer

Is the ending position of the substring. If this argument is less than *start* or greater than *inlength*, the function returns spaces.

*sublength*

Integer

Is the length in characters of the substring (normally  $end - start + 1$ ). If *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 field to which the result is returned, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, the format must be specified.

**Example**    **Extracting a String**

POSIT determines the position of the first letter I in LAST\_NAME and stores the result in I\_IN\_NAME. SUBSTR then extracts three characters beginning with the letter I from LAST\_NAME, and stores the results in I\_SUBSTR.

```
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 output is:

LAST_NAME	I_IN_NAME	I_SUBSTR
BANNING	5	ING
IRVING	1	IRV
MCKNIGHT	5	IGH
ROMANS	0	
SMITH	3	ITH
STEVENS	0	

Since Romans and Stevens have no I in their names, SUBSTR extracts a blank string.

---

## TRIM: Removing Leading and Trailing Occurrences

Available Operating Systems: All

Available Languages: reporting

The TRIM function removes leading and/or trailing occurrences of a pattern within a character string.

There is a version of the TRIM function that is available only in the Maintain language. For information on this function, see *TRIM: Removing Trailing Occurrences (Maintain)* in Chapter 5.

**Syntax**    **How to Remove Leading and Trailing Occurrences**

`TRIM(trim_where, string, string_length, pattern, pattern_length, outfield)`

where:

*trim\_where*

Alphanumeric

Is one of the following, which indicates where to remove the pattern:

'L' removes leading occurrences.

'T' removes trailing occurrences.

'B' removes both leading and trailing occurrences.

*string*

Alphanumeric

Is the source character string enclosed in single quotation marks, or the field containing the string.

*string\_length*

Integer

Is the length of the string in characters.

*pattern*

Alphanumeric

Is the pattern to remove enclosed in single quotation marks.

*pattern\_length*

Integer

Is the number of characters in the pattern.

*outfield*

Alphanumeric

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

In Dialogue Manager, the format must be specified.

### Example Removing Leading Occurrences

TRIM removes leading occurrences of the characters BR from the DIRECTOR field and stores the result in a field with the format A17:

```
TABLE FILE MOVIES
PRINT  DIRECTOR AND
COMPUTE
    TRIMDIR/A17 = TRIM('L', DIRECTOR, 17, 'BR', 2, 'A17');
    WHERE DIRECTOR CONTAINS 'BR'
END
```

The output is:

DIRECTOR	TRIMDIR
ABRAHAMS J.	ABRAHAMS J.
BROOKS R.	OOKS R.
BROOKS J.L.	OOKS J.L.

### Example Removing Trailing Occurrences

TRIM removes trailing occurrences of the characters ER from the TITLE. In order to remove trailing non-blank characters, trailing spaces must be removed first. The TITLE field has trailing spaces. Therefore, TRIM does not remove the characters ER when creating field TRIMT. The SHORT field does not have trailing spaces. Therefore, TRIM removes the trailing ER characters when creating field TRIMS:

```
DEFINE FILE MOVIES
SHORT/A19 = SUBSTR(19, TITLE, 1, 19, 19, SHORT);
END

TABLE FILE MOVIES
PRINT  TITLE IN 1 AS 'TITLE: '
      SHORT IN 40 AS 'SHORT: ' OVER
COMPUTE
    TRIMT/A39 = TRIM('T', TITLE, 39, 'ER', 2, 'A39'); IN 1 AS 'TRIMT: '
COMPUTE
    TRIMS/A19 = TRIM('T', SHORT, 19, 'ER', 2, 'A19'); IN 40 AS 'TRIMS: '
WHERE TITLE LIKE '%ER'
END
```

The output is:

TITLE:	LEARN TO SKI BETTER	SHORT:	LEARN TO SKI BETTER
TRIMT:	LEARN TO SKI BETTER	TRIMS:	LEARN TO SKI BETT
TITLE:	FANNY AND ALEXANDER	SHORT:	FANNY AND ALEXANDER
TRIMT:	FANNY AND ALEXANDER	TRIMS:	FANNY AND ALEXAND

## UPCASE: Converting Text to Uppercase

---

Available Operating Systems: All

Available Languages: reporting

The UPCASE function converts a character string to uppercase. It is useful for sorting on a field that contains both mixed case and uppercase values. Sorting on a mixed case field produces incorrect results because the sorting sequence in EBCDIC always places lowercase letters before uppercase letters, while the ASCII sorting sequence always places uppercase letters before lowercase. To obtain correct 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 exactly as they were typed. Use UPCASE to convert entries for particular fields to uppercase.

There is a version of the UPCASE function that is available only in the Maintain language. For information on this function, see *UPCASE: Converting Text to Uppercase (Maintain)* in Chapter 5.

### **Syntax**

### **How to Convert Text to Uppercase**

`UPCASE(length, input, outfield)`

where:

*length*

Integer

Is the length in characters of *input* and *outfield*.

*input*

Alphanumeric

Is the character string enclosed in single quotation marks, or the field containing the character string.

*outfield*

Alphanumeric

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

In Dialogue Manager, the format must be specified.

**Example**    **Converting a Mixed Case Field to Uppercase**

UPCASE converts the LAST\_NAME\_MIXED field to uppercase:

```
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.

The output is:

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**    **Converting a Lowercase Field to Uppercase With MODIFY**

Suppose your company decides to store employee names in mixed case and the department assignments in uppercase.

To enter records for 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

```

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 exactly as typed.
2. You type the following data and press *Enter*:

```

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

```

3. The procedure searches the data source for the ID 444555666. If it does not find the ID, it continues processing the transaction.
4. UPCASE 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 transaction message indicates the number of transactions accepted or rejected:

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

## Character Functions for AnV Fields

---

AnV fields, which represent variable length data types supported by relational database management systems, can be used as arguments in any function that requires an alphanumeric argument. There are also character functions created specifically for use with AnV fields. These are:

- LENV
- LOCASV
- POSITV
- SUBSTV
- TRIMV
- UPCASV

### **Reference** Usage Notes for Using an AnV Field in a Function

The following affect the use of an AnV field in a function:

- When using an AnV argument in a function, the input parameter is treated as an An parameter and is padded with blanks to its declared size (*n*). If the last parameter specifies an AnV format, the function result is converted to type AnV with actual length set equal to its size.
- Many functions require both an alphanumeric string and its length as input arguments. If the supplied string is stored in an AnV field, you still must supply a length argument to satisfy the requirements of the function. However, the length that will be used in the function's calculations is the actual length stored as the first two bytes of the AnV field.
- In general, any input argument can be a field or a literal. In most cases, numeric input arguments are supplied to these functions as literals, and there is no reason not to supply an integer value. However, if the value is not an integer, it is truncated to an integer value regardless of whether it was supplied as a field or a literal.

## LENV: Returning the Length of an Alphanumeric Field

Available Operating Systems: All

Available Languages: reporting

LENV returns the actual length of an AnV field or the size of an An field.

### **Syntax** How to Find the Length of an Alphanumeric Field

`LENV(string, outfield)`

where:

*string*

Alphanumeric

Is the source field or an alphanumeric constant enclosed in single quotation marks. If it is a field, it can have An or AnV format. If it is a field of type AnV, its length is taken from the length bytes stored in the field.

*outfield*

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

### **Example** Finding the Length of an AnV Field

LENV creates an AnV field named TITLEV by removing trailing blanks from the TITLE value and returns the actual length of each instance of TITLEV to the ALEN field:

```
TABLE FILE MOVIES
PRINT
COMPUTE TITLEV/A39V = TRIMV('T', TITLE, 39, ' ', 1, TITLEV);
        ALEN/I2 = LENV(TITLEV,ALEN);
BY CATEGORY NOPRINT
WHERE CATEGORY EQ 'CHILDREN'
END
```

The output is:

TITLEV	ALEN
-----	----
SMURFS, THE	11
SHAGGY DOG, THE	15
SCOOBY-DOO-A DOG IN THE RUFF	28
ALICE IN WONDERLAND	19
SESAME STREET-BEDTIME STORIES AND SONGS	39
ROMPER ROOM-ASK MISS MOLLY	26
SLEEPING BEAUTY	15
BAMBI	5

## LOCASV: Creating a Variable Length Lowercase String

Available Operating Systems: All

Available Languages: reporting

LOCASV converts alphabetic characters to lowercase. This is similar to LOCASE, but LOCASV can return AnV output whose actual length is the lesser of the actual length of the AnV input field and an input parameter that specifies the length limit.

### **Syntax**

#### **How to Create a Variable Length Lowercase String**

```
LOCASV(length_limit, string, outfield)
```

where:

*length\_limit*

Numeric

Is the maximum length of the input string.

*string*

Alphanumeric

Is the character string to be converted in single quotation marks, or a field or variable that contains the string. If it is a field, it can have An or AnV format. If it is a field of type AnV, its length is taken from the length bytes stored in the field. If *length\_limit* is smaller than the actual length, the source string is truncated to this upper limit.

*outfield*

Alphanumeric

Is the name of the field in which to store the result, or the format of the output value enclosed in single quotation marks. This value can be for a field that is AnV or An format. Is the field has AnV or An format for the returned lowercase string or the format of the output value enclosed in single quotation marks.

**Example**    **Creating a Variable Length Lowercase String**

In this example, LOCASV converts the LAST\_NAME field to lowercase and specifies a length limit of five characters. The results are stored in the LOWCV\_NAME field:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWCV_NAME/A15 = LOCASV(5, LAST_NAME, LOWCV_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

LAST_NAME	LOWCV_NAME
SMITH	smith
JONES	jones
MCCOY	mccoy
BLACKWOOD	black
GREENSPAN	green
CROSS	cross

## POSITV: Finding the Beginning of a Variable Length Substring

Available Operating Systems: All

Available Languages: reporting

The POSITV function finds the starting position of a substring within a larger string. For example, the starting position of the substring DUCT in the string PRODUCTION is 4. If the substring is not in the parent string, the function returns the value 0. This is similar to POSIT; however, the lengths of its AnV parameters are based on the actual lengths of those parameters in comparison with two other parameters that specify their sizes.

### Syntax

#### How to Find the Beginning of a Variable Length Substring

```
POSITV(parent, in_limit, substring, sub_limit, outfield)
```

where:

*parent*

Alphanumeric

Is the parent character string enclosed in single quotation marks, or a field or variable that contains the parent character string. If it is a field, it can have An or AnV format. If it is a field of type AnV, its length is taken from the length bytes stored in the field. If *in\_limit* is smaller than the actual length, the source string is truncated to this upper limit.

*in\_limit*

Integer

Is the maximum length of the input field.

*substring*

Alphanumeric

Is the substring whose position you want to find. This can be the substring enclosed in single quotation marks, or the field that contains the string. If it is a field, it can have An or AnV format. If it is a field of type AnV, its length is taken from the length bytes stored in the field. If *sub\_limit* is smaller than the actual length, the source string is truncated to this upper limit.

*sub\_limit*

Numeric

Is the maximum length of the substring.

*outfield*

Integer

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

**Example Finding the Starting Position of a Variable Length Pattern**

POSITV finds the starting position of a trailing definite or indefinite article in a movie title (such as ", THE" in SMURFS, THE). First TRIMV removes the trailing blanks from the title so that the article will be the trailing pattern:

```
DEFINE FILE MOVIES
  TITLEV/A39V = TRIMV('T',TITLE, 39,' ', 1, TITLEV);
  PSTART/I4 = POSITV(TITLEV,LENV(TITLEV,'I4'),' ',1,'I4');
  PLEN/I4 = IF PSTART NE 0 THEN LENV(TITLEV,'I4') - PSTART +1
            ELSE 0;

END
TABLE FILE MOVIES
  PRINT TITLE
        PSTART AS 'Pattern,Start' IN 25
        PLEN AS 'Pattern,Length'
BY CATEGORY NOPRINT
WHERE PLEN NE 0
END
```

The output is:

TITLE	Pattern Start	Pattern Length
----	-----	-----
SMURFS, THE	7	5
SHAGGY DOG, THE	11	5
MALTESE FALCON, THE	15	5
PHILADELPHIA STORY, THE	19	5
TIN DRUM, THE	9	5
FAMILY, THE	7	5
CHORUS LINE, A	12	3
MORNING AFTER, THE	14	5
BIRDS, THE	6	5
BOY AND HIS DOG, A	16	3

**SUBSTV: Extracting a Variable Length Substring**

Available Operating Systems: All

Available Languages: reporting

The SUBSTV function extracts a substring based on where it begins and its length in the parent string. This is similar to SUBSTR; however, the end position for the string is calculated from the starting position and the substring length. Therefore, it has fewer parameters than SUBSTR. Also, the actual length of the output field if it is an AnV field is determined based on the substring length.

## **Syntax**    **How to Extract a Variable Length Substring**

`SUBSTV(in_limit, parent, start, sublength, outfield)`

where:

*in\_limit*

Numeric

Is the maximum length of the input string.

*parent*

Alphanumeric

Is the parent string enclosed in single quotation marks, or the field containing the parent string. If it is a field, it can have *An* or *AnV* format. If it is a field of type *AnV*, its length is taken from the length bytes stored in the field. If *in\_limit* is smaller than the actual length, the source string is truncated to this size. The final length value determined by this comparison will be referred to as *p\_length* (see the description of the *outfield* parameter).

*start*

Integer

Is the starting position of the substring in the parent string. The starting position can exceed the input string length.

*sublength*

Integer

Is the length in characters of the substring (normally  $end - start + 1$ ). The end position of the substring is  $end = start + sublength - 1$ . Note that the ending position can exceed the input string length depending on the provided values for *start* and *sublength* provided.

*outfield*

Alphanumeric

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. This field can be in *An* or *AnV* format.

If the format of *outfield* is *AnV*, the actual length, *outlen*, is computed as follows from the values for *end*, *start*, and *p\_length* (see the *parent* parameter):

If  $end > p\_length$  or  $end < start$ , then  $outlen = 0$  otherwise,  $outlen = end - start + 1$ .



**Example** **Extracting a Variable Length Substring**

The following request extracts a trailing definite or indefinite article from a movie title (such as ", THE" in "SMURFS, THE"). First it trims the trailing blanks so that the article is the trailing pattern. Next it finds the starting position and length of the pattern. Then SUBSTV extracts the pattern and TRIMV trims the pattern from the title:

```

DEFINE FILE MOVIES
  TITLEV/A39V = TRIMV('T',TITLE, 39,' ', 1, TITLEV);
  PSTART/I4 = POSITV(TITLEV,LENV(TITLEV,'I4'), ', ', 1, 'I4');
  PLEN/I4 = IF PSTART NE 0 THEN LENV(TITLEV,'I4') - PSTART +1
            ELSE 0;
  PATTERN/A20V= SUBSTV(39, TITLEV, PSTART, PLEN, PATTERN);
  NEWTIT/A39V = TRIMV('T',TITLEV, 39, PATTERN,LENV(PATTERN,'I4'), NEWTIT);
END
TABLE FILE MOVIES
  PRINT TITLE
    PSTART AS 'Pattern,Start' IN 25
    PLEN AS 'Pattern,Length'
    NEWTIT AS 'Trimmed,Title' IN 55
BY CATEGORY NOPRINT
WHERE PLEN NE 0
END

```

The output is:

TITLE	Pattern Start	Pattern Length	Trimmed Title
-----	-----	-----	-----
SMURFS, THE	7	5	SMURFS
SHAGGY DOG, THE	11	5	SHAGGY DOG
MALTESE FALCON, THE	15	5	MALTESE FALCON
PHILADELPHIA STORY, THE	19	5	PHILADELPHIA STORY
TIN DRUM, THE	9	5	TIN DRUM
FAMILY, THE	7	5	FAMILY
CHORUS LINE, A	12	3	CHORUS LINE
MORNING AFTER, THE	14	5	MORNING AFTER
BIRDS, THE	6	5	BIRDS
BOY AND HIS DOG, A	16	3	BOY AND HIS DOG

## TRIMV: Removing Characters From a String

Available Operating Systems: All

Available Languages: reporting

The TRIMV function removes leading and/or trailing occurrences of a pattern within a character string. TRIMV is similar to TRIM; however, TRIMV allows the input string and the pattern to be in AnV format.

TRIMV is useful for converting an An field to an AnV field (with the length bytes containing the actual length of the data up to the last non-blank character).

### **Syntax** How to Remove Characters From a String

```
TRIMV(trim_where, string, length_limit, pattern, plength_limit, outfield)
```

where:

*trim\_where*

Alphanumeric

Is one of the following, which indicates where to remove the pattern:

'L' removes leading occurrences.

'T' removes trailing occurrences.

'B' removes both leading and trailing occurrences.

*string*

Alphanumeric

Is the source character string enclosed in single quotation marks, or the field containing the string. If it is a field, it can have An or AnV format. If it is a field of type AnV, its length is taken from the length bytes stored in the field. If *length\_limit* is smaller than the actual length, the source string is truncated to this upper limit.

*length\_limit*

Integer

Is the maximum length of the input string.

*pattern*

Alphanumeric

Is the pattern to remove enclosed in single quotation marks. If it is a field, it can have An or AnV format. If it is a field of type AnV, its length is taken from the length bytes stored in the field. If *plength\_limit* is smaller than the actual length, the pattern is truncated to this limit.

*plength\_limit*

Integer

Is the maximum length of the pattern.

*outfield*

Alphanumeric

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. The field can be in *AnV* or *An* format.If the format of *outfield* is *AnV*, the actual length is equal to the number of characters left after trimming.**Example Creating an AnV Field by Removing Trailing Blanks**TRIMV creates an *AnV* field named TITLEV by removing trailing blanks from the TITLE value:

```
TABLE FILE MOVIES
PRINT DIRECTOR
COMPUTE TITLEV/A39V = TRIMV('T', TITLE, 39, ' ', 1, TITLEV);
BY CATEGORY
END
```

Here are the first 10 lines of the output:

<u>CATEGORY</u>	<u>DIRECTOR</u>	<u>TITLEV</u>
ACTION	SPIELBERG S.	JAWS
	VERHOVEN P.	ROBOCOP
	VERHOVEN P.	TOTAL RECALL
	SCOTT T.	TOP GUN
	MCDONALD P.	RAMBO III
CHILDREN		SMURFS, THE
	BARTON C.	SHAGGY DOG, THE
		SCOOBY-DOO-A DOG IN THE RUFF
	GEROMINI	ALICE IN WONDERLAND
		SESAME STREET-BEDTIME STORIES AND SONGS
		...

**UPCASV: Creating a Variable Length Uppercase String**

Available Operating Systems: All

Available Languages: reporting

UPCASV converts alphabetic characters to uppercase like UPCASE. However, UPCASV can return *AnV* output whose actual length is the lesser of the actual length of the *AnV* input field and an input parameter that specifies the size.

## **Syntax**    **How to Create a Variable Length Uppercase String**

```
UPCASV(length_limit, string, outfield)
```

where:

*length\_limit*

Numeric

Is a positive constant or a field whose integer portion represents the size and, therefore, the upper limit for the length of the input string.

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or the field containing the character string. If it is a field, it can have *An* or *AnV* format. If it is a field of type *AnV*, its length is taken from the length bytes stored in the field. If *length\_limit* is smaller than the actual length, the source string is truncated to this size.

*outfield*

Alphanumeric

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. This can be a field with *AnV* or *An* format.

If the format of *outfield* is *AnV*, then the actual length returned is equal to the smaller of the input string length and *length\_limit*.

## **Example**    **Creating a Variable Length Uppercase String**

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');
LAST_NAME_UPCASV/A15=UPCASV(5, LAST_NAME_MIXED, 'A15') ;
END
```

Suppose you execute a request that sorts by this field:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME_MIXED AND FIRST_NAME BY LAST_NAME_UPCASV
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
END
-RUN
```

The output is:

LAST_NAME_UPCASV	LAST_NAME_MIXED	FIRST_NAME
BANNI	Banning	JOHN
BLACK	BLACKWOOD	ROSEMARIE
CROSS	CROSS	BARBARA
MCCOY	MCCOY	JOHN
MCKNI	Mcknight	ROGER
ROMAN	Romans	ANTHONY



---

---

## CHAPTER 5

### Maintain-specific Character Functions

Character functions manipulate alphanumeric fields or character strings. The functions in this topic are available only in the Maintain language.

There are additional character functions that are available in both the reporting and Maintain languages. For information on these functions, see Chapter 4, *Character Functions*.

#### Topics:

- CHAR2INT: Translating a Character to an Integer Value
- INT2CHAR: Translating an Integer Value to a Character
- LCWORD and LCWORD2: Converting a Character String to Mixed Case
- LENGTH: Determining the Length of a Character String
- LJUST: Left-Justifying a Character String (Maintain)
- LOWER: Converting a Character String to Lowercase
- MASK: Extracting or Adding Characters
- NLSCHR: Converting Characters From the Native English Code Page
- OVERLAY: Overlaying a Character String (Maintain)
- POSIT: Finding the Beginning of a Substring (Maintain)
- RJUST: Right-Justifying a Character String (Maintain)
- SELECTS: Decoding a Value From a Stack
- STRAN: Substituting One Substring for Another
- STRCMP: Comparing Character Strings
- STRICMP: Comparing Character Strings and Ignoring Case
- STRNCMP: Comparing Character Substrings
- STRTOKEN: Extracting a Substrings Based on Delimiters
- SUBSTR: Extracting a Substring (Maintain)
- TRIM: Removing Trailing Occurrences (Maintain)
- TRIMLEN: Determining the Length of a String Excluding Trailing Spaces
- UPCASE: Converting Text to Uppercase (Maintain)

## CHAR2INT: Translating a Character to an Integer Value

---

Available Operating Systems: All

Available Languages: Maintain

The CHAR2INT function translates an ASCII or EBCDIC character to the integer value it represents, depending on the operating system.

### **Syntax** How to Translate a Character Into an Integer Value

```
CHAR2INT (" character ")
```

where:

*character*

Is the ASCII or EBCDIC character to translate into its integer value.

### **Example** Translating a Character Into an Integer Value

CHAR2INT translates the character X into its integer equivalent.

```
MAINTAIN  
INT/I3=CHAR2INT ("X");  
type "INT IS <INT";  
END
```

On an ASCII platform, the integer value would be 120.

On an EBCDIC platform, the integer value would be 231.



## INT2CHAR: Translating an Integer Value to a Character

---

Available Operating Systems: All

Available Languages: Maintain

The INT2CHAR function translates an integer into the equivalent ASCII or EBCDIC character, depending on the operating system.

### **Syntax**      **How to Translate an Integer Value Into a Character**

`INT2CHAR (value)`

where:

*value*

Is the integer to translate into its equivalent ASCII or EBCDIC character.

### **Example**      **Translating an Integer Value Into a Character**

INT2CHAR translates the integer value 93 into its character equivalent.

```
MAINTAIN
CHAR/A1=INT2CHAR(93);
TYPE "CHAR IS <CHAR>";
END
```

On an ASCII or EBCDIC platform, the result would be a right bracket (]).

## LCWORD and LCWORD2: Converting a Character String to Mixed Case

---

Available Operating Systems: All

Available Languages: Maintain

The LCWORD and LCWORD2 functions convert the letters in a character string to mixed case. These functions convert character strings in the following way:

- **LCWORD.** Converts every alphanumeric character to lowercase except the first letter of each new word and 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.

If LCWORD encounters a number in the character string, it treats it as an uppercase character and continues to convert the following alphabetic characters to lowercase.

- **LCWORD2.** Converts every alphanumeric character to lowercase except the first letter of each new word. If LCWORD2 encounters a lone single quotation mark, the next letter is converted to lowercase. For example, 'SMITH' would be changed to 'Smith,' and JACK'S would be changed to Jack's.

To use these functions, you must import the function library MNTUWS. For information on importing this library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is also an LCWORD function available for both the reporting and Maintain languages. For information on this function, see *LCWORD: Converting a Character String to Mixed Case* in Chapter 4.

### **Syntax**

### **How to Convert a Character String to Mixed Case**

```
{LCWORD|LCWORD2} (string)
```

where:

*string*

Alphanumeric

Is the character string to be converted, or a temporary field that contains the string.

**Example**    **Converting a Character String to Mixed Case**

LCWORD and LCWORD2 convert the string O'CONNOR to mixed case:

```
MAINTAIN FILE CAR
MODULE IMPORT (MNTUWS)
COMPUTE MYVAL1/A10="O'CONNOR";
  COMPUTE LC1/A10 = LCWORD(MYVAL1);
  COMPUTE LC2/A10 = LCWORD2(MYVAL1);
  TYPE "<<MYVAL1 <<LC1 <<LC2"
END
```

The output is:

```
MYVAL1      LC1            LC2
O'CONNOR    O'Connor    O'connor
```

**LENGTH: Determining the Length of a Character String**

---

Available Operating Systems: All

Available Languages: Maintain

The LENGTH function determines the length of a character string, including trailing spaces.

**Syntax**    **How to Determine the Length of a Character String**

```
LENGTH(string)
```

where:

```
string
```

Alphanumeric

Is the character string whose length is to be found, or a temporary field that contains the string.

**Example**    **Determining the Length of a Character String**

LENGTH determines the length of a variable in COUNTRY:

```
MAINTAIN FILE CAR
MODULE IMPORT (MNTUWS)
NEXT COUNTRY INTO STK1
COMPUTE LEN/I3 = LENGTH(STK1(1).COUNTRY);
TYPE "<<STK1(1).COUNTRY HAS A LENGTH OF <<LEN"
END
```

The result is:

```
ENGLAND HAS A LENGTH OF 10
```

## **LJUST: Left-Justifying a Character String (Maintain)**

---

Available Operating Systems: All

Available Languages: Maintain

The LJUST function left-justifies a character string within a field. All leading spaces are removed.

LJUST will not have any visible effect in a report that uses StyleSheets (SET STYLE=ON) unless you center the item.

To use this function, you must import the function library MNTUWS. For information on importing this library see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is also an LJUST function available for the reporting language. For information on this function, see *LJUST: Left-Justifying a Character String* in Chapter 4.

### **Syntax**    **How to Left-Justify a Character String**

*LJUST(string)*

where:

*string*

Alphanumeric

Is the character string to be justified, or a temporary field that contains the string.

## **LOWER: Converting a Character String to Lowercase**

---

Available Operating Systems: All

Available Languages: Maintain

The LOWER function converts a character string to lowercase.

To use this function, you must import the function library MNTUWS. For details on importing this library see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

### **Syntax**    **How to Convert a Character String to Lowercase**

*LOWER(string)*

where:

*string*

Alphanumeric

Is the character string to be converted, or a temporary field that contains the string.

## MASK: Extracting or Adding Characters

---

Available Operating Systems: All

Available Languages: Maintain

The MASK function extracts characters from or adds characters to an alphanumeric string. It can extract a substring from different parts of the parent string, and can insert characters from a parent string into another substring. For example, it can extract the first two characters and the last two characters of a string to form a single substring.

MASK works by comparing the characters in a mask to the characters in a source field. When it encounters a 9 in the mask, MASK copies the corresponding character from the source field to the new field. When it encounters a dollar sign in the mask, MASK ignores the corresponding character in the source field. When it encounters any other character in the mask, MASK copies that character to the corresponding position in the new field.

MASK replaces the masking functionality of the EDIT function that is available in the reporting language.

### **Syntax** How to Extract or Add Characters

```
MASK(fieldname, 'mask') 
```

where:

*fieldname*

Is the source field.

*mask*

Is a character string enclosed in single quotation marks, or a temporary field that contains the string.

### **Example** Extracting a Character From a Field

MASK extracts the first initial from the FIRST\_NAME field:

```
MASK(FIRST_NAME, '9$$$$$$$$$')
```

The following are sample values for FIRST\_NAME and the values for the result of the MASK function:

<i>FIRST_NAME</i>	<i>MASK(FIRST_NAME, '9\$\$\$\$\$\$\$\$\$')</i>
-----	-----
MARY	M
DIANE	D
JOHN	J
ROSEMARIE	R
MARY	M
BARBARA	B

**Example Adding Dashes to a Field**

MASK adds dashes to the EMP\_ID field:

```
MASK (EMP_ID, '999-99-9999')
```

The following are sample values for EMP\_ID and the values for the result of the MASK function:

EMP_ID	MASK (EMP_ID, '999-99-9999')
112847612	112-84-7612
117593129	117-59-3129
219984371	219-98-4371
326179357	326-17-9357
543729165	543-72-9165
818692173	818-69-2173

---

## NLSCHR: Converting Characters From the Native English Code Page

Available Operating Systems: All

Available Languages: Maintain

NLSCHR converts a character from the native English code page to the running code page. This is useful when hosting Web applications on an EBCDIC host with non-English code pages.

**Syntax How to Convert Characters From the Native English Code Page**

```
NLSCHR ("character")
```

where:

*character*

Is the character being converted from the native English code page.

**Example Converting Characters From the Native English Code Page**

NLSCHR forces the dollar sign to display whenever the variable ADOLLAR is used regardless of the code page being run.

```
MAINTAIN
ADOLLAR/A1=NLSCHR (" $ " );
.
.
.
END
```

## **OVLAY: Overlaying a Character String (Maintain)**

---

Available Operating Systems: All

Available Languages: Maintain

The OVLAY function overlays a base character string with a substring.

To use this function, you must import the function library MNTUWS. For information on importing this library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is also an OVLAY function available for the reporting language. For information on this function, see *OVLAY: Overlaying a Character String* in Chapter 4.

### **Syntax**    **How to Overlay a Character String**

```
OVLAY(string1, string2, position)
```

where:

*string1*

Alphanumeric

Is the base character string.

*string2*

Alphanumeric

Is the substring that will overlay *string1*.

*position*

Integer

Is the position in the base string at which the overlay begins.

**Example**    **Overlaying a Character String**

OVRLAY replaces the letters MCA in the MOVIECODE field with MHD:

```
MAINTAIN FILE movies
Module Import (mntuws);

Case Top
Infer moviecode into MCASTK
Compute MCASTK.NEWCODE/A6;
For all next Moviecode into stk1
Stack copy from stk1 into MCASTK
  where moviecode contains 'MCA';
Compute i/i2=1;
Type "Original Code    New Code"
repeat mcastk.Foccount
  Compute MCASTK(i).Newcode = OVRLAY(MCASTK(I).MOVIECODE, 'MHD', 4);
  Type " <<MCASTK(i).moviecode                    <<MCASTK(I).NEWCODE"
  Compute i=i+1;
endrepeat
EndCase
END
```

The following are sample values for MOVIECODE and the values for the result of the OVRLAY function:

Original Code	New Code
001MCA	001MHD
081MCA	081MHD
082MCA	082MHD
161MCA	161MHD
196MCA	196MHD
530MCA	530MHD
550MCA	550MHD
883MCA	883MHD



## **POSIT: Finding the Beginning of a Substring (Maintain)**

---

Available Operating Systems: All

Available Languages: Maintain

The POSIT function finds the starting position of a substring within a larger string. For example, the starting position of the substring DUCT in the string PRODUCTION is 4. If the substring is not in the parent string, the function returns the value 0.

To use this function, you must import the function library MNTUWS. For information on importing this library see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is also a POSIT function available for the reporting language. For information on this function, see *POSIT: Finding the Beginning of a Substring* in Chapter 4.

### ***Syntax***

#### **How to Find the Beginning of a Substring**

`POSIT(parent, substring)`

where:

*parent*

Alphanumeric

Is the parent string.

*substring*

Alphanumeric

Is the substring for which to find the position.

**Example Finding the Beginning of a Substring**

POSIT displays all movie titles containing the word ROOF and the starting position of the ROOF string:

```
MAINTAIN FILE movies
Module Import (mntuws);

Case Top
For all next Moviecode into stk1
  Where Title Contains 'ROOF';
Compute i/i2=1;
type "      Title          Start Position of word ROOF"
repeat stk1.Foccount
  Compute STK1(i).POS/I3 = POSIT(STK1(I).TITLE, 'ROOF');
  Type " <STK1(i).Title <<STK1(I).pos"
  Compute i=i+1;
endrepeat
EndCase
END
```

The following are sample values for MOVIECODE and values for the result of the POSIT function:

Title	Start Position of word ROOF
FIDDLER ON THE ROOF	16
CAT ON A HOT TIN ROOF	18

## RJUST: Right-Justifying a Character String (Maintain)

---

Available Operating Systems: All

Available Languages: Maintain

The RJUST function right-justifies a character string. All trailing blanks become leading blanks. This is useful when you display alphanumeric fields containing numbers.

RJUST does not have any visible effect in a report that uses StyleSheets (SET STYLE=ON) unless you center the item. Also, if you use RJUST on a platform on which StyleSheets are turned on by default, issue SET STYLE=OFF before running the request.

There is also an RJUST function available for the reporting language. For information on this function, see *RJUST: Right-Justifying a Character String* in Chapter 4.

### **Syntax**

### **How to Right-Justify a Character String**

`RJUST(string, length, char)`

where:

*string*

Is the character string, or a temporary field that contains the string.

*length*

Is the length in characters of the result. If this argument is less than the length of *string*, RJUST trims *string* from right to left. If this argument is zero, RJUST returns a variable length string of length zero.

*char*

Is the character to pad the character string with and right-justify it. RJUST uses *char* only when *length* is greater than the length of *string*.

## SELECTS: Decoding a Value From a Stack

---

Available Operating Systems: All

Available Languages: Maintain

The SELECTS function decodes a value from a stack.

### **Syntax** How to Decode a Value From a Stack

```
target SELECTS (code result, code result, ... [ELSE default])
```

where:

*target*

Is a valid expression. It can be either a field name or a variable that resolves to a single stack cell.

*code*

Is the value SELECTS searches for. Once the value is found, the input expression is assigned the corresponding result. The comma between the code and result is optional.

*result*

Is the value assigned when the input expression has the corresponding code.

*default*

Is the value to be assigned if the code is not found among the list of codes. If the default is omitted, a space or zero is assigned to non-matching codes.

### **Example** Decoding Values With SELECTS

The following computes a user-defined field based on the values in a stack:

```
COMPUTE Square = Stk(Cnt).Number SELECTS (1 1, 2 4, 3 9);
```

Because SELECTS is a binary operator, it can also be used in an expression:

```
COMPUTE Square_Plus = Stk(Cnt).Number SELECTS (1 1, 2 4, 3 9) +1;
```

**Example Decoding a Value From a Stack**

The following example uses MASK extracts the first character of the field CURR\_JOBCODE in the EMPLOYEE file. Then SELECTS creates a value for the field JOB\_CATEGORY:

```

MAINTAIN FILE Employee

Case Top
  FOR ALL NEXT EMPINFO.EMP_ID INTO EmpStack;
  COMPUTE
    DEPX_CODE/A1      = MASK(EmpStack().CURR_JOBCODE, '9$$');
    JOB_CATEGORY/A15 = DEPX_CODE SELECTS (A 'ADMINISTRATIVE' B 'DATA
PROCESSING') ;
EndCase
END

```

The following table shows sample values for CURR\_JOBCODE and the corresponding values for JOB\_CATEGORY:

CURR_JOBCODE	JOB_CATEGORY
-----	-----
A01	ADMINISTRATIVE
A07	ADMINISTRATIVE
A15	ADMINISTRATIVE
A17	ADMINISTRATIVE
B02	DATA PROCESSING
B03	DATA PROCESSING
B04	DATA PROCESSING
B14	DATA PROCESSING

## STRAN: Substituting One Substring for Another

---

Available Operating Systems: All

Available Languages: Maintain

The STRAN function substitutes a substring for another substring in a character string. STRAN enables you to edit part of a character string without replacing the field entirely.

To use this function, you must import the function library MNTUWS. For details on importing this library see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

### **Syntax**

### **How to Substitute a Substring**

`STRAN(string, substr1, substr2)`

where:

`string`

Alphanumeric

Is the character string into which you want to substitute one substring for another, or a temporary field that contains the string.

`substr1`

Alphanumeric

Is the substring to replace.

`substr2`

Alphanumeric

Is the substring to insert in place of `substr1`.

**Example Substituting One String for Another**

STRAN replaces the word DOOR with the word Seater in the MODEL field:

```

MAINTAIN FILE CAR
MODULE IMPORT (MNTUWS);
FOR ALL NEXT COUNTRY CAR MODEL INTO XSTK
  WHERE MODEL CONTAINS 'DOOR'
  COMPUTE XSTK.NEWMOD/A24;
  COMPUTE I/I2=1;
  REPEAT XSTK.FOCCOUNT
    COMPUTE XSTK(I).NEWMOD=STRAN(XSTK(I).MODEL,'DOOR','SEATER');
    TYPE "<<XSTK(I).CAR <<XSTK(I).MODEL <<XSTK(I).NEWMOD"
    COMPUTE I=I+1;
  ENDREPEAT
END

```

The following are sample values for MODEL and values for the result of the STRAN function:

CAR	MODEL	STRAN
---	-----	-----
PEUGEOT	504 4 DOOR	504 4 SEATER
ALFA ROMEO	2000 4 DOOR BERLINA	2000 4 SEATER BERLINA
MASERATI	DORA 2 DOOR	DORA 2 SEATER
DATSUN	B210 2 DOOR AUTO	B210 2 SEATER AUTO
TOYOTA	COROLLA 4 DOOR DIX AUTO	COROLLA 4 SEATER DIX AUT
AUDI	100 LS 2 DOOR AUTO	100 LS 2 SEATER AUTO
BMW	2002 2 DOOR	2002 2 SEATER
BMW	2002 2 DOOR AUTO	2002 2 SEATER AUTO
BMW	3.0 SI 4 DOOR	3.0 SI 4 SEATER
BMW	3.0 SI 4 DOOR AUTO	3.0 SI 4 SEATER AUTO
BMW	530I 4 DOOR	530I 4 SEATER
BMW	530I 4 DOOR AUTO	530I 4 SEATER AUTO

## **STRCMP: Comparing Character Strings**

---

Available Operating Systems: All

Available Languages: Maintain

The STRCMP function compares two character strings using the EBCDIC or ASCII collating sequence.

- If the first string is less than the second string, STRCMP returns a negative value.
- If the first string is greater than the second string, STRCMP returns a positive value.
- If the first string is equal to the second string, STRCMP returns zero.

### ***Syntax* How to Compare Character Strings**

`STRCMP(string1, string2)`

where:

`string1, string2`

Alphanumeric

Are the strings to compare, or temporary fields that contain the strings.



**Example Comparing Character Strings**

STRCMP compares the length of two fields:

```

MAINTAIN
COMPUTE STR1/A20 = 'STRING IS LONG';
        STR2/A20 = 'STRING IS LONGER';
COMPUTE DIF/I3= STRCMP(STR1, STR2);
TYPE "STR1 = <<STR1"
TYPE "STR2 = <<STR2"
IF DIF LT 0 THEN TYPE "STR2 IS GREATER THAN STR1"
ELSE IF DIF GT 0 THEN TYPE "STR2 IS LESS THAN STR1"
ELSE IF DIF EQ 0 THEN TYPE "STR2 EQUALS STR1"
TYPE " "
COMPUTE STR3/A20 = 'STRING IS LONGEST';
        STR4/A20 = 'STRING IS LONG';
TYPE "STR3 = <<STR3"
TYPE "STR4 = <<STR4"
COMPUTE DIF= STRCMP(STR3, STR4);
IF DIF LT 0 THEN TYPE "STR4 IS GREATER THAN STR3"
ELSE IF DIF GT 0 THEN TYPE "STR4 IS LESS THAN STR3"
ELSE IF DIF EQ 0 THEN TYPE "STR4 EQUALS STR3"
TYPE " "
COMPUTE DIF= STRCMP(STR1, STR4);
IF DIF LT 0 THEN TYPE "STR1 IS GREATER THAN STR4"
ELSE IF DIF GT 0 THEN TYPE "STR1 IS LESS THAN STR4"
ELSE IF DIF EQ 0 THEN TYPE "STR1 EQUALS STR4"
END

```

The result is:

```

STR1 = STRING IS LONG
STR2 = STRING IS LONGER
STR2 IS GREATER THAN STR1

STR3 = STRING IS LONGEST
STR4 = STRING IS LONG
STR4 IS LESS THAN STR3

STR1 EQUALS STR4

```

## STRICMP: Comparing Character Strings and Ignoring Case

---

Available Operating Systems: All

Available Languages: Maintain

The STRICMP function compares two character strings using the EBCDIC or ASCII collating sequence, but ignores case differences.

- If the first string is less than the second string, STRICMP returns a negative value.
- If the first string is greater than the second string, STRICMP returns a positive value.
- If the first string is equal to the second string, STRICMP returns zero.

### **Syntax**

### **How to Compare Character Strings and Ignore Case**

```
STRICMP(string1, string2)
```

where:

```
string1, string2
```

Alphanumeric

Are the strings to compare, or temporary fields that contain the strings.

## STRNCMP: Comparing Character Substrings

---

Available Operating Systems: All

Available Languages: Maintain

The STRNCMP function compares a specified number of characters in two character strings starting at the beginning of the strings using the EBCDIC or ASCII collating sequence.

- If the first string is less than the second string, STRNCMP returns a negative value.
- If the first string is greater than the second string, STRNCMP returns a positive value.
- If the first string is equal to the second string, STRNCMP returns zero.

### **Syntax** How to Compare Character Substrings

`STRNCMP(string1, string2, number)`

where:

`string1, string2`

Alphanumeric

Are the strings that contain the substrings to compare.

`number`

Integer

Is the number of characters to compare in `string1` and `string2` you want to compare.

## STRTOKEN: Extracting a Substrings Based on Delimiters

---

Available Operating Systems: All

Available Languages: Maintain

The STOKEN function returns a substring, that consists of a string's characters from the beginning of a string to a specified character, called a delimiter.

To use this function, you must import the function library MNTUWS. For details on importing this library see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

### **Syntax**

### **How to Extract a Substring**

`STRTOKEN(string, delimiters)`

where:

*string*

Alphanumeric

Is the character string, or a variable that contains the string enclosed in double quotation marks.

*delimiters*

Alphanumeric

Is a character string, or variable enclosed in double quotation marks that contains a list of delimiters. Separate the delimiters with semicolons.

**Example**    **Extracting a Substring**

STRTOKEN returns a substring of the first five STREET values in the VIDEOTRK data source based on the delimiters period, space, or asterisk.

```

MAINTAIN FILE VIDEOTRK
MODULE IMPORT (MNTUWS);
FOR ALL NEXT CUSTID INTO CSTACK ;
COMPUTE CNT/I5 = 1;
TYPE "    ";
REPEAT WHILE CNT LE 5;
COMPUTE SUBSTREET/A20 = STRTOKEN(CSTACK(CNT) . STREET, ". ; , *");
TYPE " STREET =          <CSTACK(CNT) . STREET"
TYPE " SUBSTREET =      <SUBSTREET "
COMPUTE CNT = CNT +1;
ENDREPEAT
END

```

The output is:

```

STREET =          86 ELLIOTT AVE.
SUBSTREET =       86
STREET =          7 DAVENPORT LA.
SUBSTREET =       7
STREET =          8 MAGNOLIA LA.
SUBSTREET =       8
STREET =          35 POWELL ST.
SUBSTREET =       35
STREET =          10 COW LA.
SUBSTREET =       10

```

## **SUBSTR: Extracting a Substring (Maintain)**

---

Available Operating Systems: All

Available Languages: Maintain

The SUBSTR function extracts a substring based on where it begins and its length in the parent string. SUBSTR can vary the position of the substring depending on the values of other fields.

There is also a SUBSTR function available for the reporting language. For information on this function, see *SUBSTR: Extracting a Substring* in Chapter 4.

### **Syntax**

#### **How to Extract a Substring**

*SUBSTR*(*string*, *start*, *length*)

where:

*string*

Alphanumeric

Is the parent string enclosed in single quotation marks, or a field or variable containing the character string.

*start*

Integer

Is the starting position of the substring in the parent string.

*length*

Integer

Is the length in characters of the substring.

**Example**    **Extracting the First Character of a String in Maintain**

SUBSTR extracts the first letter of FIRST\_NAME, combines it with LAST\_NAME, and stores the result in UID:

```
MAINTAIN FILE EMPLOYEE
CASE TOP
INFER EMP_ID FIRST_NAME LAST_NAME INTO ADDSTACK
COMPUTE UID/A9 = SUBSTR(ADDSTACK().FIRST_NAME,1,1) ||
                ADDSTACK().LAST_NAME;
ENDCASE
END
```

The following table shows sample values for FIRST\_NAME and LAST\_NAME, and the corresponding values for UID:

FIRST_NAME	LAST_NAME	UID
JOE	SMITH	JSMITH
SAM	JONES	SJONES
TERRI	WHITE	TWHITE

**TRIM: Removing Trailing Occurrences (Maintain)**

---

Available Operating Systems: All

Available Languages: Maintain

The TRIM function removes trailing occurrences of a pattern within a character string.

There is also a TRIM function available for the reporting language. For information on this function, see *TRIM: Removing Leading and Trailing Occurrences* in Chapter 4.

**Syntax**    **How to Remove Trailing Occurrences**

```
TRIM(string)
```

where:

*string*

Alphanumeric

Is the character string enclosed in single quotation marks, or the field containing the string.

## TRIMLEN: Determining the Length of a String Excluding Trailing Spaces

---

Available Operating Systems: All

Available Languages: Maintain

The TRIMLEN function determines the length of a character string excluding trailing spaces.

### **Syntax**    **How to Determine the Length of a String Excluding Trailing Spaces**

`TRIMLEN (string)`

where:

`string`

Alphanumeric

Is the string to be measured.

### **Example**    **Determining the Length of a String Excluding Trailing Spaces**

TRIMLEN determines the length of a field in COUNTRY excluding trailing blanks:

```
MAINTAIN FILE CAR
MODULE IMPORT (MNTUWS)
NEXT COUNTRY INTO STK1
COMPUTE LEN/I3 = LENGTH(STK1(1).COUNTRY);
COMPUTE LEN2/I3 = TRIMLEN(STK1(1).COUNTRY);
TYPE "<STK1(1).COUNTRY HAS A LENGTH OF <LEN2 WITHOUT TRAILING BLANKS"
END
```

The result is:

```
ENGLAND HAS A LENGTH OF 7 WITHOUT TRAILING BLANKS
```



## UPCASE: Converting Text to Uppercase (Maintain)

---

Available Operating Systems: All

Available Languages: Maintain

The UPCASE function converts a character string to uppercase. It is useful for sorting on a field that contains both mixed case and uppercase values. Sorting on a mixed case field produces incorrect results because the sorting sequence in EBCDIC always places lowercase letters before uppercase letters, while the ASCII sorting sequence always places uppercase letters before lowercase. To obtain correct results, define a new field with all of the values in uppercase, and sort on that.

To use this function, you must import the function library MNTUWS. For information on importing this library see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is also an UPCASE function available for the reporting language. For information on this function, see *UPCASE: Converting Text to Uppercase* in Chapter 4.

### **Syntax**    **How to Convert Text to Uppercase**

`UPCASE(string)`

where:

*string*

Alphanumeric

Is the character string to be converted to uppercase.

*UPCASE: Converting Text to Uppercase (Maintain)*

---

---

## CHAPTER 6

# Data Source and Decoding Functions

### Topics:

- DECODE: Decoding Values
- FIND: Verifying the Existence of a Value in an Indexed Field
- LAST: Retrieving the Preceding Value
- LOOKUP: Retrieving a Value From a Cross-referenced Data Source

Data source and decoding functions search for data source records, retrieve data source records or values, and assign values based on the value of an input field.

The result of a data source function must be stored in a field. The result cannot be stored in a Dialogue Manager variable.

## DECODE: Decoding Values

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The DECODE function assigns values based on the coded value of an input field. DECODE is useful for giving a coded value in a field a more meaningful value. For example, the field GENDER may have the code F for female employees and M for male employees for efficient storage (for example, one character instead of six for female). DECODE expands (decodes) these values to ensure correct interpretation on a report.

You can use DECODE by supplying values directly in the function or by reading values from a separate file.

The use of DECODE with Maintain is limited. For information on decoding values with subscripted stack values, see *SELECTS: Decoding a Value From a Stack* in Chapter 5.

### Syntax

### How to Supply Values in the Function

```
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 the coded value DECODE searches for. If the value has embedded blanks, commas, or other special characters, enclose it in single quotation marks. When DECODE finds the specified value, it assigns the corresponding result.

*result*

Any Supported Format

Is the value assigned to a code. If the value has embedded blanks or commas or contains a negative number, enclose it in single quotation marks.

*default*

Any Supported Format

Is the value assigned if the code is not found. If you omit a default value, DECODE assigns a blank or zero to non-matching codes.

You can use up to 40 lines to define the code and result pairs for any given DECODE function, or 39 lines if you also use an ELSE phrase. Use either a comma or blank to separate the code from the result, or one pair from another.

**Example**    **Supplying Values in the Function**

EDIT extracts the first character of the CURR\_JOBCODE field, then DECODE returns either ADMINISTRATIVE or DATA PROCESSING depending on the value extracted.

```
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 output is:

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

**Syntax**    **How to Read Values From a 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 containing the decoded values.

*default*

Any Supported Format

Is the value assigned if the code is not found. If you omit a default, DECODE assigns a blank or zero to non-matching codes.

## Reference Guidelines for Reading Values From a File

- Each record in the file is expected to contain pairs of elements separated by a comma or blank.
- If each record in the file consists of only one element, this element is interpreted as the code, and the result becomes either a blank 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 criteria specified in a computational expression. For example:

```
TAKE = DECODE SELECT (filename ELSE 1);  
VALUE = IF TAKE IS 0 THEN... ELSE...;
```

TAKE is 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...;
```

- The file can contain up to 32,767 characters in the file.
- All data is interpreted in ASCII format on UNIX, or in EBCDIC format on OS/390 or VM/CMS, and converted to the USAGE format of the DECODE pairs.
- Leading and trailing blanks are ignored.
- The remainder of each record is 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 HOLD command, which writes fields in the internal format, and the DECODE pairs are interpreted accordingly. In this case, extraneous data in the record is ignored.

**Example**    **Reading Values From a File**

The following example has two parts. The first part creates a file with a list of IDs and reads the EDUCFILE data source. The second part reads the EMPLOYEE data source 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 whose 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
```

The output is:

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

## **FIND: Verifying the Existence of a Value in an Indexed Field**

---

Available Operating Systems: All

Available Languages: MODIFY, Maintain

The FIND function determines if an incoming data value is in an indexed FOCUS data source field. The function sets a temporary field to a non-zero value if the incoming value is in the data source field, and to 0 if it is not. A value greater than zero confirms the presence of the data value, not the number of instances in the data source field.

You can also use FIND in a VALIDATE command to determine if a transaction field value exists in another FOCUS data source. If the field value is not in that data source, the function returns a value of 0, causing the validation test to fail and the request to reject the transaction.

You can use any number of FINDs in a COMPUTE or VALIDATE command. However, more FINDs increase processing time and require more buffer space in memory.

**Limit:** FIND does not work on files with different DBA passwords.

The opposite of FIND is NOT FIND. The NOT FIND function sets a temporary field to 1 if the incoming value is not in the data source and 0 if the incoming value is in the data source.

### **Syntax**

#### **How to Verify the Existence of an Indexed Field**

```
FIND(fieldname [AS dbfield] IN file);
```

where:

*fieldname*

Is the name of the field that contains the incoming data value.

*AS dbfield*

Is the name of the data source field whose values are compared to the values in the incoming field. This field must be indexed. If the incoming field and the data source field have the same name, omit this phrase.

*file*

Is the name of the indexed FOCUS data source.

Do not include a space between FIND and the left parenthesis.



**Example Verifying the Existence of a Value in an Indexed Field**

FIND determines if a supplied value in the EMP\_ID field is in the EDUCFILE data source. The procedure then displays a message indicating the result of the search.

```

MODIFY FILE EMPLOYEE
PROMPT EMP_ID
COMPUTE
    EDTEST = FIND(EMP_ID IN EDUCFILE);
    MSG/A40 = IF EDTEST NE 0 THEN
        'STUDENT LISTED IN EDUCATION FILE' ELSE
        'STUDENT NOT LISTED IN EDUCATION FILE';
MATCH EMP_ID
    ON NOMATCH TYPE "<MSG"
    ON MATCH TYPE "<MSG"
DATA

```

A sample execution is:

```

>
EMPLOYEE ON 12/04/2001 AT 12.09.03
DATA FOR TRANSACTION          1

EMP_ID          =
112847612
STUDENT LISTED IN EDUCATION FILE
DATA FOR TRANSACTION          2

EMP_ID          =
219984371
STUDENT NOT LISTED IN EDUCATION FILE
DATA FOR TRANSACTION          3

```

The procedure processes as follows:

1. The procedure prompts you for an employee ID. You enter 112847612.
2. The procedure searches the EDUCFILE data source for the employee ID 112847612. It finds the ID so prints STUDENT LISTED IN EDUCATION FILE.
3. The procedure prompts you for an employee ID. You enter 219984371.
4. The procedure searches the EDUCFILE data source for the employee ID 219984371. It does not find the ID so prints STUDENT NOT LISTED IN EDUCATION FILE.

**Example**    **Rejecting a Transaction When a Value Is Not Found**

The following updates the number of hours an employee spent in class. The VALIDATE command rejects a transaction for an employee whose ID is not found in the EDUCFILE data source, which records class attendance.

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID ED_HRS
VALIDATE
    EDTEST = FIND(EMP_ID IN EDUCFILE);
MATCH EMP_ID
    ON NOMATCH REJECT
    ON MATCH UPDATE ED_HRS
DATA
```

A sample execution is:

```
>
EMPLOYEE ON 12/04/2001 AT 12/26/08
DATA FOR TRANSACTION      1

EMP_ID      =
112847612
ED_HRS      =
7
DATA FOR TRANSACTION      2

EMP_ID      =
219984371
ED_HRS      =
0
(FOC421) TRANS 2 REJECTED INVALID EDTEST
219984371, 0, $
DATA FOR TRANSACTION      3
```

The procedure processes as follows:

1. The procedure prompts you for an employee ID and the number of hours the employee spent in class. You enter the following data:

EMP\_ID: 112847612

ED\_HRS: 7

2. The procedure updates the number of hours for the ID 112847612.
3. The procedure prompts you for an employee ID and the number of hours the employee spent in class. You enter the following data:

EMP\_ID: 219984371

ED\_HRS: 0

4. The procedure rejects the record for the ID 219984371 because it does not exist in the EDUCFILE data source, and an error message is returned.

## LAST: Retrieving the Preceding Value

---

Available Operating Systems: All

Available Languages: reporting

The LAST function retrieves the preceding value for a field.

The effect of LAST depends on whether it appears in a DEFINE or COMPUTE command:

- In a DEFINE command, the LAST value applies to the previous record retrieved from the data source before sorting takes place.
- In a COMPUTE command, the LAST value applies to the record in the previous line of the internal matrix.

Do not use LAST with the -SET command in Dialogue Manager.

### **Syntax**

#### **How to Retrieve the Preceding Value**

*LAST* *fieldname*

where:

*fieldname*

Alphanumeric or Numeric

Is the field name.

**Example Retrieving the Preceding Value**

LAST retrieves the previous value of the DEPARTMENT field to determine whether to restart the running total of salaries by department. If the previous value equals the current value, CURR\_SAL is added to RUN\_TOT to generate a running total of salaries within each 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 output is:

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

## LOOKUP: Retrieving a Value From a Cross-referenced Data Source

---

Available Operating Systems: All

Available Languages: MODIFY

The LOOKUP function retrieves a data value from a cross-referenced FOCUS data source in a MODIFY request. You can retrieve data from a data source cross-referenced statically in a Master File or a data source joined dynamically to another by the JOIN command. LOOKUP retrieves a value, but does not activate the field. LOOKUP is required because a MODIFY request, unlike a TABLE request, cannot read cross-referenced data sources freely.

LOOKUP allows a request to use the retrieved data in a computation or message, but it does not allow you to modify a cross-referenced data source. To modify more than one data source in one request, use the COMBINE command or the Maintain facility.

LOOKUP can read a cross-referenced segment that is linked directly to a segment in the host data source (the host segment). This means that the cross-referenced segment must have a segment type of KU, KM, DKU, or DKM (but not KL or KLU) or must contain the cross-referenced field specified by the JOIN command. Because LOOKUP retrieves a single cross-referenced value, it is best used with unique cross-referenced segments.

The cross-referenced segment contains two fields used by LOOKUP:

- The field containing the retrieved value. Alternatively, you can retrieve all the fields in a segment at one time. The field, or your decision to retrieve all the fields, is specified in LOOKUP.

For example, LOOKUP retrieves all the fields from the segment

```
RTN = LOOKUP (SEG . DATE_ATTEND) ;
```

- The cross-referenced field. This field shares values with a field in the host segment called the host field. These two fields link the host segment to the cross-referenced segment. LOOKUP uses the cross-referenced field, which is indexed, to locate a specific segment instance.

When using LOOKUP, the MODIFY request reads a transaction value for the host field. It then searches the cross-referenced segment for an instance containing this value in the cross-referenced field:

- If there are no instances of the value, the function sets a return variable to 0. If you use the field specified by LOOKUP in the request, the field assumes a value of blank if alphanumeric and 0 if numeric.
- If there are instances of the value, the function sets the return variable to 1 and retrieves the value of the specified field from the first instance it finds. There can be more than one if the cross-referenced segment type is KM or DKM, or if you specified the ALL keyword in the JOIN command.

**Syntax**      **How to Retrieve a Value From a Cross-referenced Data Source**

`LOOKUP(field);`

where:

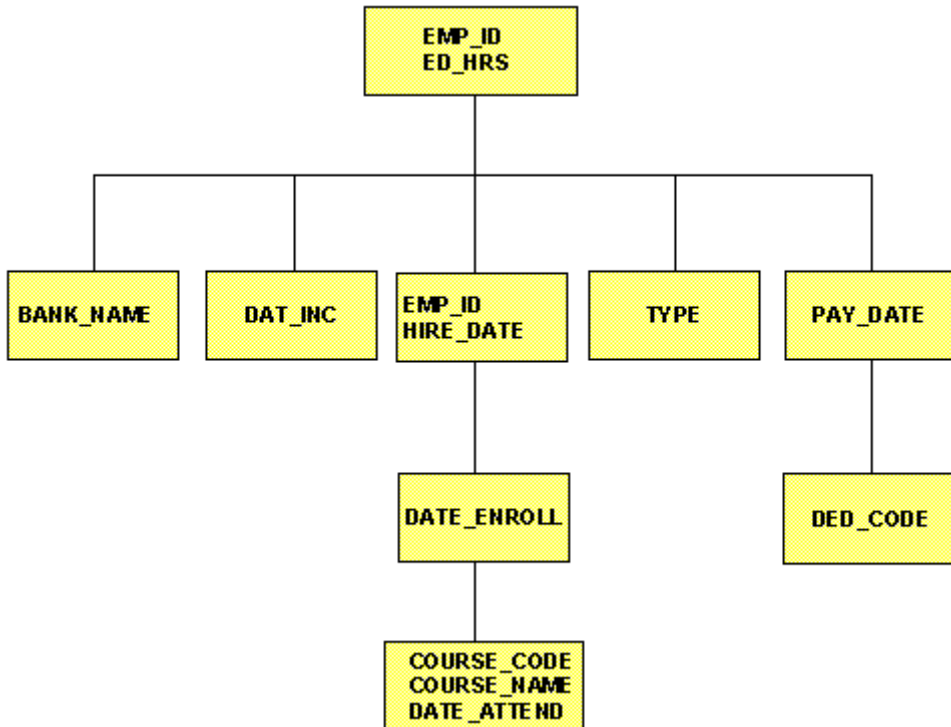
*field*

Is the name of the field to retrieve in the cross-referenced file. If the field name also exists in the host data source, you must qualify it here.

Do not include a space between LOOKUP and the left parenthesis.

**Example**      **Reading a Value From a Cross-referenced Data Source**

You may need to determine if employees were hired before or after a specific date, for example, January 1, 1982. The employee IDs (EMP\_ID) and hire date (HIRE\_DATE) are located in the host segment. The file structure is shown in this diagram:



The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID ED_HRS
COMPUTE
    EDTEST          = LOOKUP(HIRE_DATE);
COMPUTE
    ED_HRS = IF DATE_ENROLL GE 820101 THEN ED_HRS * 1.1
            ELSE ED_HRS;
MATCH EMP_ID
    ON MATCH UPDATE ED_HRS
    ON NOMATCH REJECT
DATA
```

A sample execution is:

1. The request prompts you for the employee ID and number of class hours. Enter the ID 117593129 and 10 class hours.
2. LOOKUP locates the first instance in the cross-referenced segment containing the employee ID 117593129. Since the instance exists, the function returns a 1 to the EDTEST variable. This instance lists the enroll date as 821028 (October 28, 1982).
3. LOOKUP retrieves the value 821028 for the DATE\_ENROLL field.
4. The COMPUTE command tests the value of DATE\_ENROLL. Since October 28, 1982 is after January 1, 1982, the ED\_HRS are increased from 10 to 11.
5. The request updates the classroom hours for employee 117593129 with the new value.

### **Example** Using a Value in a Host Segment to Search a Data Source

You can use a field value in a host segment instance to search a cross-referenced segment. Do the following:

- In the MATCH command that selects the host segment instance, activate the host field with the ACTIVATE command.
- In the same MATCH command, code LOOKUP after the ACTIVATE command.

This request displays the employee ID, date of salary increase, employee name, and the employee position after the raise was granted:

- The employee ID and name (EMP\_ID) are in the root segment.
- The date of increase (DAT\_INC) is in the descendant host segment.
- The job position is in the cross-referenced segment.
- The shared field is JOBCODE. You never enter a job code; the values are stored in the data source.

The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID DAT_INC
MATCH EMP_ID
    ON NOMATCH REJECT
    ON MATCH CONTINUE
MATCH DAT_INC
    ON NOMATCH REJECT
    ON MATCH ACTIVATE JOBCODE
    ON MATCH COMPUTE
    RTN = LOOKUP (JOB_DESC) ;
ON MATCH TYPE
    "EMPLOYEE ID:           <EMP_ID"
    "DATE INCREASE:        <DAT_INC"
    "NAME:                  <D.FIRST_NAME           <D.LAST_NAME"
    "POSITION:              <JOB_DESC"
DATA
```



A sample execution is:

1. The request prompts you for the employee ID and date of pay increase. Enter the employee ID 071382660 and the date 820101 (January 1, 1982).
2. The request locates the instance containing the ID 071382660, then locates the child instance containing the date of increase 820101.
3. This child instance contains the job code A07. The ACTIVATE command makes this value available to LOOKUP.
4. LOOKUP locates the job code A07 in the cross-referenced segment. It returns a 1 the RTN variable and retrieves the corresponding job description SECRETARY.
5. The TYPE command displays the values:

```
EMPLOYEE ID:      071382660
DATE INCREASE:   82/01/01
NAME:            ALFRED STEVENS
POSITION:        SECRETARY
```

Fields retrieved by LOOKUP do not require the D. prefix. FOCUS treats the field values as transaction values.

You may also need to activate the host field if you are using LOOKUP within a NEXT command. This request displays the latest position held by an employee:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID
MATCH EMP_ID
  ON NOMATCH REJECT
  ON MATCH CONTINUE
NEXT DAT_INC
  ON NONEXT REJECT
  ON NEXT ACTIVATE JOBCODE
  ON NEXT COMPUTE
  RTN = LOOKUP (JOB_DESC) ;
  ON MATCH TYPE
    "EMPLOYEE ID:      <EMP_ID"
    "DATE OF POSITION:  <DAT_INC"
    "NAME:            <D.FIRST_NAME <D.LAST_NAME"
    "POSITION:        <JOB_DESC"
DATA
```

**Example Using the LOOKUP Function With a VALIDATE Command**

When you use LOOKUP, reject transactions containing values for which there is no corresponding instance in the cross-reference segment. To do this, place the function in a VALIDATE command. If the function cannot locate the instance in the cross-referenced segment, it sets the value of the return variable to 0, causing the request to reject the transaction.

The following request updates an employee's classroom hours (ED\_HRS). If the employee enrolled in classes on or after January 1, 1982, the request increases the number of classroom hours by 10%. The enrollment dates are stored in a cross-referenced segment (field DATE\_ATTEND). The shared field is the employee ID.

The request is as follows:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID ED_HRS
VALIDATE
    TEST_DATE = LOOKUP (DATE_ENROLL) ;
COMPUTE
    ED_HRS = IF DATE_ENROLL GE 820101 THEN ED_HRS * 1.1
            ELSE ED_HRS;
MATCH EMP_ID
    ON MATCH UPDATE ED_HRS
    ON NOMATCH REJECT
DATA
```

If an employee record is not found in the cross-referenced segment, that employee never enrolled in a class. A transaction is an error and is rejected.

## Using the Extended LOOKUP Function

If the LOOKUP function cannot locate a value of the host field in the cross-referenced segment, use extended syntax to locate the next highest or lowest cross-referenced field value in the cross-referenced segment.

To use this feature, create the index with the INDEX parameter set to NEW (the binary tree scheme). To determine the type of index used by a data source, enter the? FDT command.

### **Syntax** How to Use the Extended LOOKUP Function

```
COMPUTE
LOOKUP(field action);
```

where:

*field*

Is the name of the field in the cross-referenced data source, used in a MODIFY computation. If the field name also exists in the host data source, you must qualify it here.

*action*

Specifies the action the request takes. Valid values are:

EQ causes LOOKUP to take no further action if an exact match is not found. If a match is found, the value of *rcode* is set to 1; otherwise, it is set to 0. This is the default.

GE causes LOOKUP to locate the instance with the next highest value of the cross-referenced field. The value of *rcode* is set to 2.

LE causes LOOKUP to locate the instance with the next lowest value of the cross-referenced field. The value of *rcode* is set to -2.

Do not include a space between LOOKUP and the left parenthesis.

The following table shows the value of *rcode*, depending on which instance LOOKUP locates:

Value	Action
1	Exact cross-referenced value located.
2	Next highest cross-referenced value located.
-2	Next lowest cross-referenced value located.
0	Cross-referenced value not located.

*LOOKUP: Retrieving a Value From a Cross-referenced Data Source*

---

---

## CHAPTER 7

# Date and Time Functions

### Topics:

- Date and Time Function Terminology
- Using Standard Date and Time Functions
- Using Legacy Date Functions

Date and time functions manipulate date and time values. There are two types of date and time functions:

- Standard date and time functions for use with non-legacy dates. For details, see *Using Standard Date and Time Functions* on page 7-2.
- Legacy date functions for use with legacy dates. For more information, see *Using Legacy Date Functions* on page 7-47.

If a date is in an alphanumeric or numeric field that contains date display options (for example, I6YMD), you must use the legacy date functions.

In addition to the functions discussed in this topic, there are date and time functions that are available only in the Maintain language. For information on these functions, see Chapter 8, *Maintain-specific Date and Time Functions*.

## Date and Time Function Terminology

---

Date and time functions are created for use with a date format date, or a legacy date. The following is the difference between a non-legacy date, also called a date format, and a legacy date:

- **Standard functions.** Standard date and time functions are for use with date format. 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 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 date format was formerly called a smart date.

- **Legacy functions.** Legacy date functions are for use with legacy dates. A legacy date refers to an integer, packed decimal, 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 returns the data in the field. The sample value 030599 displays as 03/05/99.

## Using Standard Date and Time Functions

---

When using standard date and time functions, you need to understand the settings that alter the behavior of these functions, as well as the acceptable formats and how to supply values in these formats.

You can affect the behavior of date and time functions in the following ways:

- Defining which days of the week are work days and which are not. Then, when you use a date function involving work days, dates that are not work days are ignored. For details, see *Specifying Work Days* on page 7-3.
- Determining whether to display leading zeros when a date function in Dialogue Manager returns a date. For details, see *Enabling Leading Zeros For Date and Time Functions in Dialogue Manager* on page 7-5.

## Specifying Work Days

You can determine which days are work days and which are not. Work days affect the DATEADD, DATEDIF, and DATEMOV functions. You identify work days in the following ways:

- Specify business days. See *Specifying Business Days* on page 7-3.
- Specify holidays. See *Specifying Holidays* on page 7-4.

### Specifying Business Days

Business days are traditionally Monday through Friday, but not every business has this schedule. For example, if your company does business on Sunday, Tuesday, Wednesday, Friday, and Saturday, you can tailor business day units to reflect that schedule.

#### **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:

- To identify a day of the week as a business day, enter the first letter of that day in that day's position.
- To identify a non-business day, enter an underscore ( `_` ) in that day's position.

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

#### **Example** Setting Business Days to Reflect Your Work Week

The following designates work days as Sunday, Tuesday, Wednesday, Friday, and Saturday:

```
SET BUSDAYS = S_TW_FS
```

**Syntax**     **How to View the Current Setting of Business Days**

? SET BUSDAYS

**Specifying Holidays**

You can specify a list of dates that are designated as holidays in your company. These dates are excluded when using functions that perform calculations based on working days. For example, if Thursday in a given week is designated as a holiday, the next working day after Wednesday is Friday.

To define a list of holidays, you must:

1. Create a holiday file using a standard text editor.
2. Select the holiday file by issuing the SET command with the HDAY parameter.

**Reference**     **Rules for Creating a Holiday File**

- Dates must be in YYMD format.
- Dates must be in ascending order.
- Each date must be on its own line.
- Each year for which data exists must be included. Calling a date function with a date value outside the range of the holiday file returns a zero for business day requests.
- You may include an optional description of the holiday, separated from the date by a space.

**Procedure**     **How to Create a Holiday File**

1. In a text editor, create a list of dates designated as holidays using the *Rules for Creating a Holiday File* on page 7-4.
2. Save the file:
  - In OS/390, the file must be a member of ERRORS named HDAYxxxx.
  - In VM/CMS, the file must be HDAYxxxx ERRORS.

where:

xxxx

Is a string of text four characters long.



**Syntax**      **How to Select a Holiday File**

```
SET HDAY = xxxx
```

where:

```
xxxx
```

Is the part of the name of the holiday file after HDAY. This string must be four characters long.

**Example**      **Creating and Selecting a Holiday File**

The following is the HDAYTEST file, which establishes holidays:

```
19910325 TEST HOLIDAY
19911225 CHRISTMAS
```

This request uses HDAYTEST in its calculations:

```
SET BUSDAYS = SMTWTFS
SET HDAY = TEST
TABLE FILE MOVIES
PRINT TITLE RELDATE
COMPUTE NEXTDATE/YMD = DATEADD(RELDATE, 'BD', 1);
WHERE RELDATE GE '19910101';
END
```

**Enabling Leading Zeros For Date and Time Functions in Dialogue Manager**

If you use a date and time function in Dialogue Manager that returns a numeric integer format, Dialogue Manager truncates any leading zeros. For example, if a function returns the value 000101 (indicating January 1, 2000), Dialogue Manager truncates the leading zeros, producing 101, an incorrect date. To avoid this problem, use the LEADZERO parameter.

LEADZERO only supports an expression that makes a direct call to a function. An expression that has nesting or another mathematical function always truncates leading zeros. For example,

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

truncates leading zeros regardless of the LEADZERO parameter setting.

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

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

where:

ON

Displays leading zeros if present.

OFF

Truncates leading zeros. This value is the default.

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

The AYM function adds one month to the input date of December 1999:

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

Using the default LEADZERO setting, this yields:

1

This represents the date of January 2000 incorrectly. Setting the LEADZERO parameter in the request as follows

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

results in the following:

0001

This correctly indicates January 2000.

## Using Date and Time Formats

There are three types of date formats that are valid in date-time values: numeric string format, formatted-string format, and translated-string format. In each format, two-digit years are interpreted using the `DEFCENT` and `YRTHRESH` parameters.

Time components are separated by colons and may be followed by A.M., P.M., a.m., or p.m.

### Numeric String Format

The numeric string format is exactly two, four, six, or eight digits. Four-digit strings are considered to be a year (century must be specified), and the month and day are set to January 1. Six and eight-digit strings contain two or four digits for the year, followed by two for the month, and two for the day. Because the component order is fixed with this format, the `DATEFORMAT` setting is ignored.

If a numeric-string format longer than eight digits is encountered, it is treated as a combined date-time string in the `Hnn` format.

#### **Example** Using Numeric String Format

The following are examples of numeric string date constants:

String	Date
99	January 1, 1999
1999	January 1, 1999
19990201	February 1, 1999

### Formatted-string Format

The formatted-string format contains a one or two-digit day, a one or two-digit month, and a two or four-digit year, each component separated by a space, slash, hyphen, or period. All three components must be present and follow the `DATEFORMAT` setting. If any of the three fields is four digits, it is interpreted as the year, and the other two fields must follow the order given by the `DATEFORMAT` setting.

#### **Example** Using Formatted-string Format

The following are examples of formatted-string date constants and specify May 20, 1999:

```
1999/05/20
5 20 1999
99.05.20
1999-05-20
```

## Translated-string Format

The translated-string format contains the full or abbreviated month name. The year must also be present in four-digit or two-digit form. If the day is missing, day 1 of the month is assumed; if present, it can have one or two digits. If the string contains both a two-digit year and a two-digit day, they must be in the order given by the DATEFORMAT setting.

### **Example** Using Translated-string Format

The following date is in translated-string format:

January 6 2000

## Time Format

Time components are separated by colons and may be followed by A.M., P.M., a.m., or p.m.

Seconds can be expressed with a decimal point or be followed by a colon. If there is a colon after seconds, the value following it represents milliseconds. There is no way to express microseconds using this notation.

A decimal point in the seconds value indicates the decimal fraction of a second. Microseconds can be represented using six decimal digits.

### **Example** Using Time Components

The following are examples of acceptable time components:

14:30:20:99           (99 milliseconds)  
14:30  
14:30:20.99           (99/100 seconds)  
14:30:20.999999      (999999 microseconds)  
02:30:20:500pm

## Assigning Date-Time Values

A date-time value is a constant in character format assigned by one of the following:

- A sequential data source.
- An expression that defines WHERE or IF criteria or creates a temporary field using the DEFINE or COMPUTE command.

### **Syntax** How to Assign Date-Time Values

#### In a character file

*date\_string* [*time\_string*]

or

*time\_string* [*date\_string*]

#### In a COMPUTE, DEFINE, or WHERE expression

DT(*date\_string* [*time\_string*])

or

DT(*time\_string* [*date\_string*])

#### In an IF expression

'*date\_string* [*time\_string*']

or

'*time\_string* [*date\_string*']

where:

*time\_string*

Is a time string in acceptable format. A time string cannot contain blanks.

*date\_string*

Is a date string in either numeric string, formatted-string, or translated-string format.

In an IF criteria, if the value does not contain blanks or special characters, the single quotation marks are not necessary.

**Note:** The date and time strings must be separated by at least one blank space. Blank spaces are also permitted at the beginning and end of the date-time string.

**Example Assigning a Date-Time Value in a COMPUTE Command**

The following uses the DT function in a COMPUTE command to create a new field containing an assigned date-time value.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME AND COMPUTE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
WHERE CURR_JOBCODE LIKE 'B%'
END
```

The output is:

LAST_NAME	FIRST_NAME	NEWSAL	RAISETIME
SMITH	MARY	\$14,520.00	2000/01/01 9:00AM
JONES	DIANE	\$20,328.00	2000/01/01 9:00AM
ROMANS	ANTHONY	\$23,232.00	2000/01/01 9:00AM
MCCOY	JOHN	\$20,328.00	2000/01/01 9:00AM
BLACKWOOD	ROSEMARIE	\$23,958.00	2000/01/01 9:00AM
MCKNIGHT	ROGER	\$17,710.00	2000/01/01 9:00AM

**Example Assigning a Date-Time Value in WHERE Criteria**

The following uses the DT function to create a new field containing an assigned date-time value. This value is then used as a WHERE criteria.

```
DEFINE FILE EMPLOYEE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME NEWSAL RAISETIME
WHERE RAISETIME EQ DT(20000101 09:00AM)
END
```

The output is:

LAST_NAME	FIRST_NAME	NEWSAL	RAISETIME
-----	-----	-----	-----
STEVENS	ALFRED	\$12,100.00	2000/01/01 9:00AM
SMITH	MARY	\$14,520.00	2000/01/01 9:00AM
JONES	DIANE	\$20,328.00	2000/01/01 9:00AM
SMITH	RICHARD	\$10,450.00	2000/01/01 9:00AM
BANNING	JOHN	\$32,670.00	2000/01/01 9:00AM
IRVING	JOAN	\$29,548.20	2000/01/01 9:00AM
ROMANS	ANTHONY	\$23,232.00	2000/01/01 9:00AM
MCCOY	JOHN	\$20,328.00	2000/01/01 9:00AM
BLACKWOOD	ROSEMARIE	\$23,958.00	2000/01/01 9:00AM
MCKNIGHT	ROGER	\$17,710.00	2000/01/01 9:00AM
GREENSPAN	MARY	\$9,900.00	2000/01/01 9:00AM
CROSS	BARBARA	\$29,768.20	2000/01/01 9:00AM

### **Example** Assigning a Date-Time Value in IF Criteria

The following uses the DT function to create a new field containing an assigned date-time value. This value is then used as an IF criteria.

```

DEFINE FILE EMPLOYEE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME NEWSAL RAISETIME
IF RAISETIME EQ '20000101 09:00AM'
END

```

The output is:

LAST_NAME	FIRST_NAME	NEWSAL	RAISETIME
-----	-----	-----	-----
STEVENS	ALFRED	\$12,100.00	2000/01/01 9:00AM
SMITH	MARY	\$14,520.00	2000/01/01 9:00AM
JONES	DIANE	\$20,328.00	2000/01/01 9:00AM
SMITH	RICHARD	\$10,450.00	2000/01/01 9:00AM
BANNING	JOHN	\$32,670.00	2000/01/01 9:00AM
IRVING	JOAN	\$29,548.20	2000/01/01 9:00AM
ROMANS	ANTHONY	\$23,232.00	2000/01/01 9:00AM
MCCOY	JOHN	\$20,328.00	2000/01/01 9:00AM
BLACKWOOD	ROSEMARIE	\$23,958.00	2000/01/01 9:00AM
MCKNIGHT	ROGER	\$17,710.00	2000/01/01 9:00AM
GREENSPAN	MARY	\$9,900.00	2000/01/01 9:00AM
CROSS	BARBARA	\$29,768.20	2000/01/01 9:00AM

**Reference Arguments for Use With Date and Time Functions**

The following component names and values are supported as arguments for the date-time functions that require them:

Component Name	Valid Values
<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

- For an argument that specifies a length of eight or ten characters, use eight to include milliseconds and ten to include microseconds in the returned value.
- The last argument is always a USAGE format that indicates the data type returned by the function. The type may be A (alphanumeric), I (integer), D (floating-point double precision), H (date-time), or a date format (for example, YYMD).



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

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

The DATEADD function adds a unit to or subtracts a unit from a date format. A unit is one of the following:

- **Year.**
- **Month.** If the calculation using the month unit creates an invalid date, DATEADD corrects it to the last day of the month. For example, adding one month to October 31 yields November 30, not November 31 since November has 30 days.
- **Day.**
- **Weekday.** When using the weekday unit, DATEADD does not count Saturday or Sunday. For example, if you add one day to Friday, the result is Monday.
- **Business day.** When using the business day unit, DATEADD uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. If Monday is not a working day, then one business day past Sunday is Tuesday. See *Rules for Creating a Holiday File* on page 7-4 for more information.

Do not use DATEADD with Dialogue Manager. DATEADD requires a date to be in date format; Dialogue Manager interprets a date as alphanumeric or numeric.

You add or subtract non day-based dates (for example, YM or YQ) directly without using DATEADD.

**Syntax**      **How to Add or Subtract a Date Unit to or From a Date**

`DATEADD(date, 'unit', #units[, outfield])`

where:

*date*

Date

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

*unit*

Alphanumeric

Is one of the following enclosed in single quotation marks:

Y indicates a year unit.

M indicates a month unit.

D indicates a day unit.

WD indicates a weekday unit.

BD indicates a business day unit.

*#units*

Integer

Is the number of date units added to or subtracted from *date*. If this number is not a whole unit, it is rounded down to the next largest integer.

*outfield*

Alphanumeric

Is the field that contains the result. This value is required only for Maintain.

**Example**      **Truncation 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 the Weekday Unit**

If you use the weekday unit and a Saturday or Sunday is the input date, DATEADD changes the input date to Monday. The function

`DATEADD(910623, 'WD', 1)`

in which DATE is either Saturday or Sunday yields Tuesday; Saturday and Sunday are not weekdays, so DATEADD begins with Monday and adds one.

**Example Adding Weekdays to a Date (Reporting)**

DATEADD adds three weekdays to NEW\_DATE. In some cases, it adds more than three days because HIRE\_DATE\_PLUS\_THREE would otherwise be on a weekend.

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

The output is:

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

**Example Determining If a Date Is a Work Day (Reporting)**

DATEADD determines which values in the TRANSDATE field do not represent work days by adding zero days to TRANSDATE using the business day unit. If TRANSDATE does not represent a business day, DATEADD returns the next business day to DATEX, which may not be the same as TRANSDATE. TRANSDATE is then compared to DATEX, and the day of the week is printed for all dates that do not match between the two fields, resulting in a list of all non-work days.

```
DEFINE FILE VIDEOTRK
DATEX/YMD = DATEADD(TRANSDATE, 'BD', 0);
DATEINT/I8YYMD = DATECVT(TRANSDATE, 'YMD', 'I8YYMD');
END

TABLE FILE VIDEOTRK
SUM TRANSDATE NOPRINT
COMPUTE DAYNAME/A8 = DOWKL(DATEINT, DAYNAME); AS 'Day of Week'
BY TRANSDATE AS 'Date'
WHERE TRANSDATE NE DATEX
END
```

The output is:

Date	Day of Week
91/06/22	SATURDAY
91/06/23	SUNDAY
91/06/30	SUNDAY

### **Example** Adding Months to a Date (Maintain)

DATEADD adds months to the DATE1 field:

```
MAINTAIN
compute DATE1/yymd = '20000101'
compute DATE2/yymd=DATEADD(date1, 'M', 2, date2);
type "DATE1 = <<DATE1 + 2 MONTHS = DATE2 = <<DATE2"
END
```

The result is:

```
DATE1 = 2000/01/01+ 2 MONTHS = DATE2 = 2000/03/01
```

## **DATECVT: Converting the Format of a Date**

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

The DATECVT function converts the format of a date in an application without requiring an intermediate calculation. If you supply an invalid format, DATECVT returns a zero or a blank.

### **Syntax** How to Convert a Date Format

```
DATECVT(date, 'infmt', 'outfmt' [, outfield])
```

where:

*date*

Date

Is the date to be converted. If you supply an invalid date, DATECVT returns zero. When the conversion is performed, a legacy date obeys any DEFCENT and YRTHRESH parameter settings supplied for that field.

*infmt*

Alphanumeric

Is the format of the date enclosed in single quotation marks. It is one of the following:

- A non-legacy date format (for example, YYMD, YQ, M, DMY, JUL).
- A legacy date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in *infmt* functions as an offset from the base date of a YYMD field (12/31/1900).

*outfmt*

Alphanumeric

Is the output format enclosed in single quotation marks. It is one of the following:

- A non-legacy date format (for example, YYMD, YQ, M, DMY, JUL).
- A legacy date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in *infmt* functions as an offset from the base date of a YYMD field (12/31/1900).

*outfield*

Alphanumeric

Is the field that contains the result. This value is required only for Maintain.

**Example** **Converting a YYMD Date to DMY**

DATECVT converts 19991231 to 311299 and stores the result in CONV\_FIELD:

```
CONV_FIELD/DMY = DATECVT(19991231, 'YYMD', 'DMY');
```

### Example **Converting a Legacy Date to Date Format (Reporting)**

DATECVT converts HIRE\_DATE from I6YMD legacy date format to YYMD date format:

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

The output is:

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

### **DATEDIF: Finding the Difference Between Two Dates**

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

The DATEDIF function returns the difference between two dates in units. A unit is one of the following:

- **Year.** Using the year unit with DATEDIF yields the inverse of DATEADD. If subtracting one year from date X creates date Y, then the count of years between X and Y is one. Subtracting one year from February 29 produces the date February 28.
- **Month.** Using the month unit with DATEDIF yields the inverse of DATEADD. If subtracting one month from date X creates date Y, then the count of months between X and Y is one. If the to-date is the end-of-month, then the month difference may be rounded up (in absolute terms) to guarantee the inverse rule.

If one or both of the 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.

- **Day.**
- **Weekday.** With the weekday unit, DATEDIF does not count Saturday or Sunday when calculating days. This means that the difference between Friday and Monday is one day.

- **Business day.** With the business day unit, DATEDIF uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. This means that if Monday is not a working day, the difference between Friday and Tuesday is one day. See *Rules for Creating a Holiday File* on page 7-4 for more information.

DATEDIF returns a whole number. If the difference between two dates is not a whole number, DATEDIF truncates the value to the next largest integer. For example, the number of years between March 2, 2001, and March 1, 2002, is zero. If the end date is before the start date, DATEDIF returns a negative number.

You can find the difference between non-day based dates (for example YM or YQ) directly without using DATEDIF.

## Syntax

### How to Find the Difference Between Two Dates

```
DATEDIF(from_date, to_date, 'unit'[, outfield])
```

where:

*from\_date*

Date

Is the start date from which to calculate the difference.

*to\_date*

Date

Is the end date from which to calculate the difference.

*unit*

Alphanumeric

Is one of the following enclosed in single quotation marks:

Y indicates a year unit.

M indicates a month unit.

D indicates a day unit.

WD indicates a weekday unit.

BD indicates a business day unit.

*outfield*

Alphanumeric

Is the field that contains the result. This value is required only for Maintain.

**Example Truncation With DATEDIF**

DATEDIF calculates the difference between March 2, 1996, and March 1, 1997, and returns a zero because the difference is less than a year:

```
DATEDIF(19960302, 19970301, 'Y')
```

**Example Using Month Calculations**

The following expressions return a result of minus one month:

```
DATEDIF(19990228, 19990128, 'M')
```

```
DATEDIF(19990228, 19990129, 'M')
```

```
DATEDIF(19990228, 19990130, 'M')
```

```
DATEDIF(19990228, 19990131, 'M')
```

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.
```

**Example Finding the Number of Weekdays Between Two Dates (Reporting)**

DATECVT converts the legacy dates in HIRE\_DATE and DAT\_INC to the date format YYMD. DATEDIF then uses those date formats to determine the number of weekdays between NEW\_HIRE\_DATE and NEW\_DAT\_INC:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND
COMPUTE NEW_HIRE_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AND
COMPUTE NEW_DAT_INC/YYMD = DATECVT(DAT_INC, 'I6YMD', 'YYMD'); AND
COMPUTE WDAYS_HIRED/I8 = DATEDIF(NEW_HIRE_DATE, NEW_DAT_INC, 'WD');
BY LAST_NAME
IF WDAYS_HIRED NE 0
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

LAST_NAME	FIRST_NAME	NEW_HIRE_DATE	NEW_DAT_INC	WDAYS_HIRED
IRVING	JOAN	1982/01/04	1982/05/14	94
MCKNIGHT	ROGER	1982/02/02	1982/05/14	73
SMITH	RICHARD	1982/01/04	1982/05/14	94
STEVENS	ALFRED	1980/06/02	1982/01/01	414
	ALFRED	1980/06/02	1981/01/01	153



**Example Finding the Number of Years Between Two Dates (Maintain)**

DATEDIF determines the number of years between DATE2 and DATE1:

```
MAINTAIN
```

```
Case Top
```

```
compute DATE1/ymd = '20020717';
```

```
compute DATE2/ymd = '19880705';
```

```
COmpute DIFF/I3= DATEDIF(DATE2, DATE1, 'Y', DIFF);
```

```
type "<<DATE1 - <<DATE2 = <DIFF YEARS"
```

```
ENDCASE
```

```
END
```

The result is:

```
2002/07/17 - 1988/07/05 = 14 YEARS
```

**DATEMOV: Moving a Date to a Significant Point**

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

The DATEMOV function moves a date to a significant point on the calendar.

**Syntax**      **How to Move a Date to a Significant Point**

`DATEMOV(date, 'move-point'[, outfield])`

where:

*date*

Date

Is the date to be moved. It must be date format with a day component. For example, it can be MDYY but not MYY.

*move-point*

Alphanumeric

Is the significant point the date is moved to enclosed in single quotation marks. An invalid point results in a return code of zero. Valid values are:

EOM is the end of month.

BOM is the beginning of month.

EOQ is the end of quarter.

BOQ is the beginning of quarter.

EOY is the end of year.

BOY is the beginning of year.

EOW is the end of week.

BOW is the beginning of week.

NWD is the next weekday.

NBD is the next business day.

PWD is the prior weekday.

PBD is the prior business day.

WD- is a weekday or earlier.

BD- is a business day or earlier.

WD+ is a weekday or later.

BD+ is a business day or later.

A business day calculation is affected by the BUSDAYS and HDAY parameter settings.

*outfield*

Alphanumeric

Is the field that contains the result. This value is required only for Maintain.

**Example Determining Significant Points for a Date (Reporting)**

The BUSDAYS parameter sets the business days to Monday, Tuesday, Wednesday, and Thursday. DATECVT converts the legacy date HIRE\_DATE to the date format YYMD and provides date display options. DATEMOV then determines significant points for HIRE\_DATE.

```

SET BUSDAY = _MTWT__
TABLE FILE EMPLOYEE
PRINT
COMPUTE NEW_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AND
COMPUTE NEW_DATE/WT = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AS 'DOW' AND
COMPUTE NWD/WT = DATEMOV(NEW_DATE, 'NWD'); AND
COMPUTE PWD/WT = DATEMOV(NEW_DATE, 'PWD'); AND
COMPUTE WDP/WT = DATEMOV(NEW_DATE, 'WD+'); AS 'WD+' AND
COMPUTE WDM/WT = DATEMOV(NEW_DATE, 'WD-'); AS 'WD-' AND
COMPUTE NBD/WT = DATEMOV(NEW_DATE, 'NBD'); AND
COMPUTE PBD/WT = DATEMOV(NEW_DATE, 'PBD'); AND
COMPUTE WBP/WT = DATEMOV(NEW_DATE, 'BD+'); AS 'BD+' AND
COMPUTE WBM/WT = DATEMOV(NEW_DATE, 'BD-'); AS 'BD-' BY LAST_NAME NOPRINT
HEADING
"Examples of DATEMOV"
"Business days are Monday, Tuesday, Wednesday, + Thursday "
" "
"START DATE.. | MOVE POINTS....."
WHERE DEPARTMENT EQ 'MIS';
END

```

The output is:

```

Examples of DATEMOV
Business days are Monday, Tuesday, Wednesday, + Thursday

START DATE.. | MOVE POINTS.....
NEW_DATE     DOW  NWD  PWD  WD+  WD-  NBD  PBD  BD+  BD-
-----
1982/04/01   WED  THU  TUE  WED  WED  SUN  TUE  WED  WED
1981/11/02   SUN  MON  THU  SUN  SUN  MON  WED  SUN  SUN
1982/04/01   WED  THU  TUE  WED  WED  SUN  TUE  WED  WED
1982/05/01   FRI  MON  WED  SUN  THU  MON  TUE  SUN  WED
1981/07/01   TUE  WED  MON  TUE  TUE  WED  MON  TUE  TUE
1981/07/01   TUE  WED  MON  TUE  TUE  WED  MON  TUE  TUE

```

**Example** **Determining the End of the Week (Reporting)**

DATEMOV determines the end of the week for each date in NEW\_DATE and stores the result in EOW:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND
COMPUTE NEW_DATE/YYMDWT = DATECVT(HIRE_DATE, 'I6YMD', 'YYMDWT'); AND
COMPUTE EOW/YYMDWT = DATEMOV(NEW_DATE, 'EOW');
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

LAST_NAME	FIRST_NAME	NEW_DATE	EOW
BANNING	JOHN	1982 AUG 1, SUN	1982 AUG 6, FRI
IRVING	JOAN	1982 JAN 4, MON	1982 JAN 8, FRI
MCKNIGHT	ROGER	1982 FEB 2, TUE	1982 FEB 5, FRI
ROMANS	ANTHONY	1982 JUL 1, THU	1982 JUL 2, FRI
SMITH	RICHARD	1982 JAN 4, MON	1982 JAN 8, FRI
STEVENS	ALFRED	1980 JUN 2, MON	1980 JUN 6, FRI

**Example** **Determining the End of the Week (Maintain)**

DATEMOV determines the end of the week for each date:

```
MAINTAIN
COMPUTE X/YYMDWT='20020717';
COMPUTE Y/YYMDWT=DATEMOV(X, 'EOW', Y);
TYPE "<<X <<Y END OF WEEK "
END
```

The result is:

2002/07/17, WED 2002/07/19, FRI END OF WEEK

## HADD: Incrementing a Date-Time Value

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HADD function increments a date-time value by a given number of units.

### **Syntax**    **How to Increment a Date-Time Value**

`HADD(value, 'component', increment, length, outfield)`

where:

*value*

Is the date-time value to be incremented, the name of a date-time field that contains the value, or an expression that returns the value.

*component*

Is the name of the component to be incremented enclosed in single quotation marks. For a list of valid components, see *Arguments for Use With Date and Time Functions* on page 7-12.

*increment*

Is the number of units by which to increment the component, 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 indicates a time value that includes milliseconds.

10 indicates a time value that includes microseconds.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in date-time format (data type H).

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

**Example Incrementing the Month Component of a Date-Time Field (Reporting)**

HADD adds two months to each value in TRANSDATE and stores the result in ADD\_MONTH. If necessary, the day is adjusted so that it is valid for the resulting month.

```
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

**Example Incrementing the Month Component of a Date-Time Field (Maintain)**

HADD adds two months to the DT1 field:

```
MAINTAIN FILE DATETIME
FOR 1 NEXT ID DT1 INTO DTSTK
COMPUTE
NEW_DATE/HYYMDS = HADD(DTSTK.DT1, 'MONTH', 2,10, NEW_DATE);
TYPE "DT1 IS: <DTSTK(1).DT1 "
TYPE "NEW_DATE IS: <NEW_DATE "
```

The result is:

```
DT1 IS: 2000/1/1 02:57:25
NEW_DATE IS: 2000/3/1 02:57:25
TRANSACTIONS: COMMITS = 1 ROLLBACKS = 0
SEGMENTS : INCLUDED = 0 UPDATED = 0 DELETED = 0
```

## HCNVRT: Converting a Date-Time Value to Alphanumeric Format

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HCNVRT function converts a date-time value to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.

### Syntax

#### How to Convert a Date-Time Value to Alphanumeric Format

```
HCNVRT(value, '(fmt)', length, outfield)
```

where:

*value*

Is the date-time value to be converted, the name of a date-time field that contains the value, or an expression that returns the value.

*fmt*

Is the format of the date-time field enclosed in parentheses and single quotation marks. It must be a date-time format (data type H).

*length*

Is the length of the alphanumeric field that is 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 *length* is smaller than the number of characters needed to display the alphanumeric field, the function returns a blank.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in alphanumeric format.

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting a Date-Time Field to Alphanumeric Format (Reporting)**

HCNVRT converts the TRANSDATE field to alphanumeric format. The first function does not include date-time display options for the field; the second function does for readability. It also specifies the display of seconds in the input field.

```
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, '(HYMDS)', 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

**Example**    **Converting a Date-Time Field to Alphanumeric Format (Maintain)**

HCNVRT converts the DT1 field to alphanumeric format:

```
MAINTAIN FILE DATETIME
FOR ALL NEXT ID INTO STK;
COMPUTE
RESULT_HCNVRT/A20 = HCNVRT(STK.DT1, '(HYMDH)', 20, RESULT_HCNVRT);
TYPE "STK(1).DT1 = "STK(1).DT1;
TYPE "RESULT_HCNVRT = " RESULT_HCNVRT;
END
```

**HDATE: Converting the Date Portion of a Date-Time Value to a Date Format**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HDATE function converts the date portion of a date-time value to the date format YYMD. You can then convert the result to other date formats.



**Syntax**    **How to Convert the Date Portion of a Date-Time Value to a Date Format**

```
HDATE(value, 'YYMD'[, outfield])
```

where:

*value*

Is the date-time value to be converted, the name of a date-time field that contains the value, or an expression that returns the value.

*YYMD*

Is the output format. The value must be YYMD.

*outfield*

Alphanumeric

Is the field that contains the result. This value is required only for Maintain.

**Example**    **Converting the Date Portion of a Date-Time Field to a Date Format (Reporting)**

HDATE converts the date portion of the TRANSDATE field to the 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

**Example**    **Converting the Date Portion of a Date-Time Field to a Date Format (Maintain)**

HDATE converts the date portion of DT1 to date format YYMD:

```
MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
DT1_DATE/YYMD = HDATE(STK.DT1, DT1_DATE);
TYPE "STK(1).DT1 = <STK(1).DT1";
TYPE "DT1_DATE = <DT1_DATE";
END
```

The output is:

```
STK(1).DT1 = 2000/1/1 02:57:25
DT1_DATE = 2000/01/01
```

## **HDIFF: Finding the Number of Units Between Two Date-Time Values**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HDIFF function calculates the number of units between two date-time values.

### **Syntax**    **How to Find the Number of Units Between Two Date-Time Values**

```
HDIFF(value1, value2, 'component', outfield)
```

where:

*value1*

Is the end date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

*value2*

Is the start date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

*component*

Is the name of the component to be used in the calculation enclosed in single quotation marks. If the component is a week, the WEEKFIRST parameter setting is used in the calculation.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be floating-point double-precision.

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

**Example Finding the Number of Days Between Two Date-Time Fields (Reporting)**

HDIFF calculates the number of days between the TRANSDATE and ADD\_MONTH fields and stores the result in DIFF\_PAYS, which has the format D12.2:

```
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

**Example Finding the Number of Days Between Two Date-Time Fields (Maintain)**

HDIFF calculates the number of days between ADD\_MONTH and DT1:

```
MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
NEW_DATE/HYYMDS = HADD(STK.DT1, 'MONTH', 2, 10, NEW_DATE);
DIFF_DAYS/D12.2 = HDIFF(NEW_DATE, STK.DT1, 'DAY', DIFF_DAYS);
TYPE "STK(1).DT1 = "STK(1).DT1;
TYPE "NEW_DATE = "NEW_DATE;
TYPE "DIFF_DAYS = "DIFF_DAYS
END
```

## HDTTM: Converting a Date Value to a Date-Time Value

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HDTTM function converts a date value to a date-time field. The time portion is set to midnight.

### **Syntax** How to Convert a Date Value to a Date-Time Value

`HDTTM(date, length, outfield)`

where:

*date*

Is the date value to be converted, the name of a date 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 indicates a time value that includes milliseconds.

10 indicates a time value that includes microseconds.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

### **Example** Converting a Date Field to a Date-Time Field (Reporting)

HDTTM 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

**Example**    **Converting a Date Field to a Date-Time Field (Maintain)**

HDTTM converts the date field DT1\_DATE to a date-time field:

```

MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
DT1_DATE/YYMD = HDATE(DT1, DT1_DATE);
DT2/HYYMDIA = HDTTM(DT1_DATE, 8, DT2);
TYPE "STK(1).DT1 = <STK(1).DT1";
TYPE "DT1_DATE = <DT1_DATE";
TYPE "DT2 = <DT2";
END

```

**HGETC: Storing the Current Date and Time in a Date-Time Field**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HGETC function stores the current date and time in a date-time field. If millisecond or microsecond values are not available in your operating environment, the function retrieves the value zero for these components.

**Syntax**    **How to Store the Current Date and Time in a Date-Time Field**

*HGETC*(*length*, *outfield*)

where:

*length*

Is the length of the returned date-time value. A valid values is:  
8 indicates a time value that includes milliseconds.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

**Example Storing the Current Date and Time in a Date-Time Field (Reporting)**

HGETC stores the current date and time in 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

**Example Storing the Current Date and Time in a Date-Time Field (Maintain)**

HGETC stores the current date and time in DT2:

```
MAINTAIN
COMPUTE DT2/HYYMDm = HGETC(10, DT2);
TYPE "DT2 = <DT2";
END
```

**HHMMSS: Retrieving the Current Time**

Available Operating Systems: All

Available Languages: reporting

The HHMMSS function retrieves the current time from the operating system as an eight character string, separating the hours, minutes, and seconds with periods.

A compiled MODIFY procedure must use HHMMSS to obtain the time; it cannot use the &TOD variable, which also returns the time. The &TOD variable is made current only when you execute a MODIFY, SCAN, or FSCAN procedure.

There is also an HHMMSS function available in the Maintain language. For information on this function, see *HHMMSS: Retrieving the Current Time (Maintain)* in Chapter 8.

**Syntax**    **How to Retrieve the Current Time**

```
HHMMSS(outfield)
```

where:

```
outfield
```

Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Retrieving the Current Time**

HHMMSS retrieves the current time and displays it in the page 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 output is:

```
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**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HINPUT function converts an alphanumeric string to a date-time value.

### **Syntax**

#### **How to Convert an Alphanumeric String to a Date-Time Value**

```
HINPUT(inputlength, 'inputstring', length, outfield)
```

where:

*inputlength*

Is the length of the alphanumeric string to be converted. 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 be converted enclosed in single quotation marks, the name of an alphanumeric field that contains the string, or an expression that returns the string. The string can consist of any valid date-time input value as described in *Describing Data*.

*length*

Is the length of the returned date-time value. Valid values are:  
8 indicates a time value that includes milliseconds.  
10 indicates a time value that includes microseconds.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.



**Example** Converting an Alphanumeric String to a Date-Time Value (Reporting)

HCVRT converts the TRANSDATE field to alphanumeric format, then HINPUT converts the alphanumeric string to a date-time value:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME/A20 = HCVRT(TRANSDATE, '(H17)', 17, 'A20');
DT_FROM_ALPHA/HYMD5 = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYMD5');
WHERE DATE EQ 2000;
END
```

The output is:

CUSTID	DATE-TIME	ALPHA_DATE_TIME	DT_FROM_ALPHA
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

**Example** Converting an Alphanumeric String to a Date-Time Value (Maintain)

HINPUT converts the DT1 field to alphanumeric format:

```
MAINTAIN FILE DATETIME
COMPUTE
RESULT/HMtdYYmA = HINPUT(20, '19971029133059888999', 10, RESULT);
TYPE RESULT;
END
```

## HMIDNT: Setting the Time Portion of a Date-Time Value to Midnight

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HMIDNT function changes the time portion of a date-time value to midnight (all zeroes by default). This allows you to compare a date field with a date-time field.

### Syntax

#### How to Set the Time Portion of a Date-Time Value to Midnight

```
HMIDNT(value, length, outfield)
```

where:

*value*

Is the date-time value whose time is to be set to midnight, the name of a date-time 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 indicates a time value that includes milliseconds.

10 indicates a time value that includes microseconds.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

### Example Setting the Time to Midnight (Reporting)

HMIDNT sets the time portion of the TRANSDATE field to midnight first in the 24-hour system and then in the 12-hour system:

```
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

**Example Setting the Time to Midnight (Maintain)**

HMIDNT sets the time portion of DT1 to midnight in both the 24- and 12-hour systems:

```

MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
DT_MID_24/HYYMDS = HMIDNT(STK(1).DT1, 8, DT_MID_24);
DT_MID_12/HYYMDSA= HMIDNT(STK(1).DT1, 8, DT_MID_12);
TYPE "STK(1).DT1 = "STK(1).DT1";
TYPE "DT_MID_24 = <DT_MID_24";
TYPE "DT_MID_12 = <DT_MID_12";
END

```

**HNAME: Retrieving a Date-Time Component in Alphanumeric Format**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HNAME function extracts a specified component from a date-time value in alphanumeric format.

**Syntax How to Retrieve a Date-Time Component in Alphanumeric Format**

```
HNAME(value, 'component', outfield)
```

where:

*value*

Is the date-time value from which a component is to be extracted, the name of a date-time field containing the value that contains the value, or an expression that returns the value.

*component*

Is the name of the component to be retrieved enclosed in single quotation marks. See *Arguments for Use With Date and Time Functions* on page 7-12 for a list of valid components.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in alphanumeric format.

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

The function converts all other components to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

**Example Retrieving the Week Component in Alphanumeric Format (Reporting)**

HNAME returns the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST parameter setting changes the value of the 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
```

When WEEKFIRST is set to seven, the output is:

CUSTID	DATE-TIME	WEEK_COMPONENT
1118	2000/06/26 05:45	26
1237	2000/02/05 03:30	06

When WEEKFIRST is set to three, the output is:

CUSTID	DATE-TIME	WEEK_COMPONENT
1118	2000/06/26 05:45	25
1237	2000/02/05 03:30	05

For details on WEEKFIRST, see the *Developing Applications* manual.

**Example Retrieving the Day Component in Alphanumeric Format (Reporting)**

HNAME retrieves 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 Retrieving the Day Component in Alphanumeric Format (Maintain)**

HNAME extracts the day in alphanumeric format from DT1:

```

MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
DAY_COMPONENT/A2=HNAME(STK.DT1, 'DAY', DAY_COMPONENT);
TYPE "STK(1).DT1 = "STK(1).DT1;
TYPE "DAY_COMPONENT = <DAY_COMPONENT"
END

```

**HPART: Retrieving a Date-Time Component in Numeric Format**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HPART function extracts a specified component from a date-time value and returns it in numeric format.

**Syntax How to Retrieve a Date-Time Component in Numeric Format**

```
HPART(value, 'component', outfield)
```

where:

*value*

Is a date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

*component*

Is the name of the component to be retrieved enclosed in single quotation marks. See *Arguments for Use With Date and Time Functions* on page 7-12 for a list of valid components.

*outfield*

Numeric

Is the field that contains the result, or the integer format of the output value enclosed in single quotation marks.

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

**Example Retrieving the Day Component in Numeric Format (Reporting)**

HPART retrieves 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

**Example Retrieving the Day Component in Numeric Format (Maintain)**

HPART extracts the day in integer format from DT1:

```
MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
DAY_COMPONENT/I2 = HPART(STK.DT1, 'DAY', DAY_COMPONENT);
TYPE "STK(1).DT1 = <STK(1).DT1";
TYPE "DAY_COMPONENT = <DAY_COMPONENT";
END
```

**HSETPT: Inserting a Component Into a Date-Time Value**

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HSETPT function inserts the numeric value of a specified component into a date-time value.

**Syntax**    **How to Insert a Component Into a Date-Time Value**

```
HSETPT(dtfield, 'component', value, length, outfield)
```

where:

*dtfield*

Is a date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

*component*

Is the name of the component to be inserted enclosed in single quotation marks. See *Arguments for Use With Date and Time Functions* on page 7-12 for a list of valid components.

*value*

Is the numeric value to be inserted for the requested component, 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 indicates a time value that includes milliseconds.

10 indicates a time value that includes microseconds.

*outfield*

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Inserting the Day Component Into a Date-Time Field (Reporting)**

HSETPT inserts the day as 28 into the ADD\_MONTH field and stores the result in INSERT\_DAY:

```
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

**Example**    **Inserting the Day Component Into a Date-Time Field (Maintain)**

HSETPT inserts the day into ADD\_MONTH:

```
MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE
ADD_MONTH/HYYMDS = HADD(STK.DT1, 'MONTH', 2, 8, ADD_MONTH);
INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, INSERT_DAY);
TYPE "STK(1).DT1 = <STK(1).DT1";
TYPE "ADD_MONTH = <ADD_MONTH";
TYPE "INSERT_DAY = <INSERT_DAY";
END
```



## HTIME: Converting the Time Portion of a Date-Time Value to a Number

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HTIME function converts the time portion of a date-time value to the number of milliseconds if the first argument is eight, or microseconds if the first argument is ten. To include microseconds, the input date-time value must be 10-bytes.

### **Syntax** How to Convert the Time Portion of a Date-Time Field to a Number

```
HTIME(length, value, outfield)
```

where:

*length*

Is the length of the input date-time value. Valid values are:

8 indicates a time value that includes milliseconds.

10 indicates a time value that includes microseconds.

*value*

Is the date-time value from which to convert the time, the name of a date-time field that contains the value, or an expression that returns the value.

*outfield*

Numeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be floating-point double-precision.

In FOCUS, you must specify the format. In Maintain, you must specify the name of the field.

### **Example** Converting the Time Portion of a Date-Time Field to a Number (Reporting)

HTIME converts the time portion of the TRANSDATE field to the 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

### **Example**    **Converting the Time Portion of a Date-Time Field to a Number (Maintain)**

HTIME converts the time portion of the DT1 field to the number of milliseconds:

```
MAINTAIN FILE DATETIME
FOR 1 NEXT ID INTO STK;
COMPUTE MILLISEC/D12.2 = HTIME(8, STK.DT1, MILLISEC);
TYPE "STK(1).DT1 = <STK(1).DT1";
TYPE "MILLISEC = <MILLISEC";
END
```

### **TODAY: Returning the Current Date**

Available Operating Systems: All

Available Languages: reporting

The TODAY function retrieves the current date from the operating system in the format MM/DD/YY or MM/DD/YYYY. It always returns a date that is current. Therefore, if you are running an application late at night, use TODAY. You can remove the default embedded slashes with the EDIT function.

You can also retrieve the date in the same format (separated by slashes) using the Dialogue Manager 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.

A compiled MODIFY procedure must use TODAY to obtain the date. It cannot use the system variables.

### **Syntax**    **How to Retrieve the Current Date**

`TODAY(outfield)`

where:

*outfield*

Alphanumeric, at least A8

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The following apply:

- If DATEFNS=ON and the format is A8 or A9, TODAY returns the 2-digit year.
- If DATEFNS=ON and the format 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*.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Retrieving the Current Date**

TODAY retrieves the current date and stores it in the DATE field. The request then displays the date in the page 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 output is:

```
SALARY REPORT RUN ON 12/13/1999
DEPARTMENT      CURR_SAL
-----
MIS              $108,002.00
PRODUCTION      $114,282.00
```

**Using Legacy Date Functions**

---

The functions listed in this topic are legacy date functions. They were created for use with dates in integer, packed decimal, or alphanumeric format.

**Using Old Versions of Legacy Date Functions**

All legacy date functions support dates for the year 2000 and later. The old versions of these functions may not work correctly with dates after December 31, 1999. However, in some cases you may want to use the old version of a function, for example, if you do not use year 2000 dates. You can "turn off" the current version with the DATEFNS parameter.

**Syntax How to Activate an Old Legacy Date Functions**

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

where:

ON

Activates the function that supports dates for the year 2000 and later. This value is the default.

OFF

Deactivates a function that supports dates for the year 2000 and later.

## Using Dates With Two- and Four-Digit Years

Legacy date functions accept dates with two- or four-digit years. Four-digit years that display the century, such as 2000 or 1900, can be used if their formats are specified as I8YYMD, P8YYMD, D8YYMD, F8YYMD, or A8YYMD. Two-digit years can use the DEFCENT and YRTHRESH parameters to assign century values if the field has a length of six (for example, I6YMD). For information on these parameters, see *Customizing Your Environment in Developing Applications*.

### **Example** Using Four-Digit Years

The EDIT function creates dates with four-digit years. The functions JULDAT and GREGDAT then convert these dates to Julian and Gregorian formats.

```
DEFINE FILE EMPLOYEE
DATE/I8YYMD = EDIT('19' | EDIT(HIRE_DATE));
JDATE/I7 = JULDAT(Date, 'I7');
GDATE/I8 = GREGDT(JDATE, 'I8');
END

TABLE FILE EMPLOYEE
PRINT DATE JDATE GDATE
END
```

The output is:

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

**Example Using Two-Digit Years**

The AYMD function returns an eight-digit date when the input argument has a six-digit legacy date format. Since DEFCENT is 19 and YRTHRESH is 83, year values from 83 through 99 are interpreted as 1983 through 1999, and year values from 00 through 82 are interpreted as 2000 through 2082.

```
SET DEFCENT=19, YRTHRESH=83

DEFINE FILE EMPLOYEE
NEW_DATE/I8YYMD = AYMD(EFFECT_DATE, 30, 'I8');
END

TABLE FILE EMPLOYEE
PRINT EFFECT_DATE NEW_DATE BY EMP_ID
END
```

The output is:

EMP_ID	EFFECT_DATE	NEW_DATE
-----	-----	-----
071382660		
112847612		
117593129	82/11/01	2082/12/01
119265415		
119329144	83/01/01	1983/01/31
123764317	83/03/01	1983/03/31
126724188		
219984371		
326179357	82/12/01	2082/12/31
451123478	84/09/01	1984/10/01
543729165		
818692173	83/05/01	1983/05/31

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

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

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

### **Syntax**    **How to Add or Subtract Months to or From a Date**

*AYM(indate, months, outfield)*

where:

*indate*

Numeric

Is the original date in year-month format, the name of a field that contains the date, or an expression that returns the date. If the date is not valid, the function returns a 0.

*months*

Integer

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

*outfield*

Integer

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

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

**Example Adding Months to a Date**

The COMPUTE command converts the dates in HIRE\_DATE from year-month-day to year-month format and stores the result in HIRE\_MONTH. AYM then adds six months to HIRE\_MONTH and stores the result in AFTER6MONTHS.

```
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 output is:

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 a Date**

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The AYMD function adds days to or subtracts days from a date in year-month-day format. You can convert a date to this format using the CHGDAT or EDIT function.

If the addition or subtraction of days crosses forward or backward into another century, the century digits of the output year are adjusted.

## **Syntax**      **How to Add or Subtract Days to or From a Date**

`AYMD(indate, days, outfield)`

where:

*indate*

Numeric

Is the original date in year-month-day format, the name of a field that contains the date, or an expression that returns the date. If *indate* is a field name, the field format must be I6, I6YMD, I8, I8YYMD, P6, P6YMD, F6, F6YMD, D6, or D6YMD. If the date is not valid, the function returns a 0.

*days*

Integer

Is the number of days you are adding to or subtracting from *indate*. To subtract days, use a negative number.

*outfield*

I6, I6YMD, I8, or I8YYMD

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. If *indate* is a field, *outfield* must have the same format.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## **Example**      **Adding Days to a Date**

AYMD adds 35 days to each value in the HIRE\_DATE field, and stores the result in AFTER35DAYS:

```
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 output is:

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



## CHGDAT: Changing Format of a Date

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The CHGDAT function rearranges the year, month, and day portions of a date and converts a date between long and short date format. Long format contains the year, month, and day; short format contains one or two of these elements, such as year and month, or just day. A format can specify either two digits for the year (for example, 97), or four digits (for example, 1997).

**Note:** Since CHGDAT returns the date in alphanumeric format with 17 characters, use the EDIT function to truncate this field or convert the date to numeric format.

The format of the date to be converted and the resulting date contain the following characters in any combination:

- D** Day of the month (01 through 31).
- M** Month of the year (01 through 12).
- Y[Y]** Year. Y indicates a two-digit year (such as 94); YY indicates a four-digit year (such as 1994).

To spell out the month rather than use a number, append one of the following to the format of the resulting date:

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

The function ignores any other character in the format.

**Reference Short to Long Format Conversion**

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

Portion of Date Missing	Portion Supplied by Function
Day (for example, from YM to YMD)	Last day of the month.
Month (for example, from Y to YM)	Last month of the year (December).
Year (for example, from MD to YMD)	The year 99.
Converting year from two-digit to four-digit (for example, from YMD to YYMD)	<p>If DATEFNS=ON, the century will be determined by the 100-year window defined by DEFCENT and YRTHRESH. See <i>Customizing Your Environment in Developing Applications</i> or <i>Working With Cross-Century Dates</i> in the iBase archive for details on DEFCENT and YRTHRESH.</p> <p>If DATEFNS=OFF, the year 19xx is supplied, where xx is the last two digits in the year.</p>

**Syntax**    **How to Change the Format of a Date**

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

where:

*'oldformat'*

A5

Is the format of the original date enclosed in single quotation marks.

*'newformat'*

A5

Is the format of the converted date enclosed in single quotation marks.

*indate*

Alphanumeric

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

*outfield*

Alphanumeric or A17

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Converting From YMD to MDYYX Format**

The EDIT function changes HIRE\_DATE from numeric to alphanumeric format. CHGDAT then converts each value in ALPHA\_HIRE from YMD to MDYYX format and stores the result in HIRE\_MDY, which has the format A17. The option X in the new format displays the full name of the month.

```
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 'PRODUCTION';
END
```

The output is:

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

**DA Functions: Converting a Date to an Integer**

Available Operating Systems: All

Available Languages: reporting, Maintain

The DA functions convert a date to the number of days between December 31, 1899 and that date. By converting a date to the number of days, you can add and subtract dates and calculate the intervals between them. You can convert the result back to a date using the DT functions discussed in *DT Functions: Converting an Integer to a Date* on page 7-61.

There are six DA functions; each one accepts a date in a different format.

**Syntax**    **How to Convert a Date to an Integer**

*function*(*indate*, *outfield*)

where:

*function*

Is one of the following:

DADMY converts a date in day-month-year format.

DADYM converts a date in day-year-month format.

DAMDY converts a date in month-day-year format.

DAMYD converts a date in month-year-day format.

DAYDM converts a date in year-day-month format.

DAYMD converts a date in year-month-day format.

*indate*

Numeric

Is the date to be converted, or the name of a field that contains the date. The date is truncated to an integer before conversion. The format of the date depends on the function.

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

*outfield*

Integer

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting Dates and Calculating the Difference Between Them**

DAYMD converts the DAT\_INC and HIRE\_DATE fields to the number of days since December 31, 1899, and the smaller number is then subtracted from the larger number:

```
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 output is:

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

**DMY, MDY, YMD: Calculating the Difference Between Two Dates**

Available Operating Systems: All

Available Languages: reporting, Maintain

The DMY, MDY, and YMD functions calculate the difference between two dates in integer, alphanumeric, or packed format.

**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 format.

MDY calculates the difference between two dates in month-day-year format.

YMD calculates the difference between two dates in year-month-day format.

*begin*

Numeric

Is the beginning date, or the name of a field that contains the date.

*end*

Numeric

Is the end date, or the name of a field that contains the date.

**Example**    **Calculating the Number of Days Between Two Dates**

YMD calculates the number of days between the dates in HIRE\_DATE and DAT\_INC:

```
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 output is:

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

Available Operating Systems: All

Available Languages: reporting, Maintain

The DOWK and DOWKL functions find the day of the week that corresponds to a date. DOWK returns the day as a three letter abbreviation; DOWKL displays the full name of the day.

### **Syntax**

#### **How to Find the Day of the Week**

```
{DOWK|DOWKL}(indate, outfield)
```

where:

*indate*

Numeric

Is the input date in year-month-day format. If the date is not valid, the function returns spaces. If the date specifies a two digit year and DEFCENT and YRTHRESH values have not been set, the function assumes the 20th century.

*outfield*

DOWK: A4

DOWKL: A12

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.



**Example Finding the Day of the Week**

DOWK determines the day of the week that corresponds to the value in the HIRE\_DATE field and stores the result in DATED:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND HIRE_DATE AND COMPUTE
DATED/A4 = DOWK(HIRE_DATE, DATED);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

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 Functions: Converting an Integer to a Date**

Available Operating Systems: All

Available Languages: reporting, Maintain

The DT functions convert an integer representing the number of days elapsed since December 31, 1899 to the corresponding date. They are useful when you are performing arithmetic on a date converted to the number of days (see *DA Functions: Converting a Date to an Integer* on page 7-56). The DT functions convert the result back to a date.

There are six DT functions; each one converts a number into a date of a different format.

**Syntax**      **How to Convert an Integer to a Date**

*function(number, outfield)*

where:

*function*

Is one of the following:

DTDMY converts a number to a day-month-year date.

DTDYM converts a number to a day-year-month date.

DTMDY converts a number to a month-day-year date.

DTMYD converts a number to a month-year-day date.

DTYDM converts a number to a year-day-month date.

DTYMD converts a number to a year-month-day date.

*number*

Numeric

Is the number of days since December 31, 1899. The number is truncated to an integer.

*outfield*

Integer

Is the name of the field containing the result or the format of the output value enclosed in single quotation marks. The output format depends on the function being used.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting an Integer to a Date**

DTMDY converts the NEWF field (which was converted to the number of days by DAYMD) to the corresponding date and stores the result in NEW\_HIRE\_DATE:

```

-* THIS PROCEDURE CONVERTS HIRE_DATE, WHICH IS IN I6YMD FORMAT,
-* TO A DATE IN I8MDYY FORMAT.
-* FIRST IT USES THE DAYMD FUNCTION TO CONVERT HIRE_DATE
-* TO A NUMBER OF DAYS.
-* THEN IT USES THE DTMDY FUNCTION 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 output is:

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

**GREGDT: Converting From Julian to Gregorian Format**

Available Operating Systems: All

Available Languages: reporting, Maintain

The GREGDT function converts a date in Julian format to Gregorian format (year-month-day).

A date in Julian format is a five- or seven-digit number. 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, 1999 in Julian format is either 99001 or 1999001.

### Reference **DATEFNS Settings for GREGDT**

GREGDT converts a Julian date to either YMD or YYMD format using the DEFCENT and YRTHRESH parameter settings to determine the century, if required. GREGDT returns a date as follows:

<b>DATEFNS Setting</b>	<b>I6 or I7 Format</b>	<b>I8 Format or Greater</b>
<b>ON</b>	YMD	YYMD
<b>OFF</b>	YMD	YMD

### Syntax **How to Convert From Julian 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 five- or seven-digit number after truncation. If the date is invalid, the function returns a 0.

*outfield*

I6 or I8

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Converting From Julian to Gregorian Format**

GREGDT converts the JULIAN field to YYMD (Gregorian) format. It determines the century using the default DEFCENT and YRTHRESH parameter settings.

```
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 output is:

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

**JULDAT: Converting From Gregorian to Julian Format**

Available Operating Systems: All

Available Languages: reporting, Maintain

The JULDAT function converts a date from Gregorian format (year-month-day) to Julian format (year-day). A date in Julian format is a five- or seven-digit number. 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, 1999 in Julian format is either 99001 or 1999001.

**Reference DATEFNS Settings for JULDAT**

JULDAT converts a Gregorian date to either YYNNN or YYYYNNN format, using the DEFCENT and YRTHRESH parameter settings to determine if the century is required.

JULDAT returns dates as follows:

DATEFNS Setting	I5 or I6 Format	I7 Format or Greater
ON	YYNNN	YYYYNNN
OFF	YYNNN	YYNNN

## **Syntax**      **How to Convert From Gregorian to Julian Format**

`JULDAT(indate, outfield)`

where:

*indate*

Numeric

Is the date or the name of the field that contains the date in year-month-day format (YMD or YYMD).

*outfield*

I5 or I7

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## **Example**      **Converting From Gregorian to Julian Format**

JULDAT converts the HIRE\_DATE field to Julian format. It determines the century using the default DEFCENT and YRTHRESH parameter settings.

```
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 output is:

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

## YM: Calculating Elapsed Months

Available Operating Systems: All

Available Languages: reporting, Maintain

The YM function 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 or EDIT function.

### **Syntax** How to Calculate Elapsed Months

*YM(fromdate, todate, outfield)*

where:

*fromdate*

Numeric

Is the start date in year-month format (for example, I4YM). If the date is not valid, the function returns a 0.

*todate*

Numeric

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

*outfield*

Integer

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Note:** If *fromdate* or *todate* is in integer year-month-day format (I6YMD or I8YYMD), simply divide by 100 to convert to year-month format and set the result to an integer. This drops the day portion of the date, which is now after the decimal point.

**Example**    **Calculating Elapsed Months**

The COMPUTE commands convert the dates from year-month-day to year-month format; then YM calculates the difference between the values in the HIRE\_DATE/100 and DAT\_INC/100 fields:

```
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 output is:

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 8

# Maintain-specific Date and Time Functions

### Topics:

- Maintain-specific Standard Date and Time Functions
- Maintain-specific Legacy Date Functions

Maintain-specific date and time functions manipulate date and time values. These functions are available only in Maintain.

There are additional date and time functions available in both the reporting and Maintain languages. For information on these functions, see Chapter 7, *Date and Time Functions*.

## Maintain-specific Standard Date and Time Functions

---

Standard date and time functions are for use with non-legacy dates. For a definition of standard dates and times, see Chapter 7, *Date and Time Functions*.

### HHMMSS: Retrieving the Current Time (Maintain)

Available Operating Systems: All

Available Languages: Maintain

The HHMMSS function retrieves the current time from the operating system as an 8-character string, separating the hours, minutes, and seconds with periods.

To use this function, you must import the function library MNTUWS. For information on importing a function library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is also an HHMMSS function available in the reporting language. For information on this function, see *HHMMSS: Retrieving the Current Time* in Chapter 7.

#### **Syntax**

#### **How to Retrieve the Current Time**

HHMMSS ()

#### **Example**

#### **Retrieving the Current Time**

HHMMSS retrieves the current time from the operating system:

```
MAINTAIN
Module Import (mntuws);
Case Top
Compute now/a10 = hhmmss ();
type "Now = <<now"
EndCase
END
```

The output is:

```
Now = 14.25.33
```

## **Initial\_HHMMSS: Returning the Time the Application Was Started**

Available Operating Systems: All

Available Languages: Maintain

The Initial\_HHMMSS function returns the time when the Maintain application was started as an 8-character string, separating the hours, minutes, and seconds with periods.

To use this function, you must import the function library MNTUWS. For details on importing this library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

### **Syntax**    **How to Retrieve the Initial Time**

```
Initial_HHMMSS()
```

## **Initial\_TODAY: Returning the Date the Application Was Started**

Available Operating Systems: All

Available Languages: Maintain

The Initial\_TODAY function returns the date when the Maintain application was started as an 8-character string with embedded colons separating the hours, minutes, and seconds with periods.

To use this function, you must import the function library MNTUWS. For details on importing this library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

### **Syntax**    **How to Retrieve the Initial Date**

```
Initial_TODAY()
```

## **TODAY: Retrieving the Current Date (Maintain)**

Available Operating Systems: All

Available Languages: Maintain

The TODAY function retrieves the current date from the system in the format MM/DD/YY or MM/DD/YYYY. TODAY always returns a date that is current. Therefore, if you are running an application late at night, use TODAY. You can remove the embedded slashes using the EDIT function.

To use this function, you must import the function library MNTUWS. For information on importing this library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

There is a version of the TODAY function that is available only in the reporting language. For information on this function, see *TODAY: Returning the Current Date* in Chapter 7.

**Syntax**      **How to Retrieve the Current Date**

```
TODAY ()
```

**Example**      **Retrieving the Current Date**

TODAY retrieves the current date from the system:

```
MAINTAIN
Module Import (mntuws);

Case Top
Compute date1/a8 = today();
type "Date1 = <<date1"
Endcase
END
```

The result is:

```
Date1 = 07/17/02
```

**TODAY2: Returning the Current Date**

Available Operating Systems: All

Available Languages: Maintain

The TODAY2 function retrieves the current date from the operating system in the format MM/DD/YY or MM/DD/YYYY.

To use this function, you must import the function library MNTUWS. For information on importing this library, see *How to Access the Maintain MNTUWS Function Library* in Chapter 3.

**Syntax**      **How to Retrieve the Current Date**

```
TODAY2 ()
```

**Example**      **Retrieving the Current Date**

TODAY2 retrieves the current date from the system:

```
MAINTAIN
Module Import (mntuws);

Case Top
Compute date2/a10 = today2();
type "Date2 = <<date2"
Endcase
END
```

The result is:

```
Date2 = 07/17/2002
```

## Maintain-specific Legacy Date Functions

---

Legacy date functions are for use with legacy dates. For a definition of legacy dates and times, see *Using Legacy Date Functions* in Chapter 7.

### ADD: Adding Days to a Date

Available Operating Systems: All

Available Languages: Maintain

The ADD function adds a given number of days to a date.

#### **Syntax** How to Add Days to a Date

`ADD(date, value)`

or

`date.ADD(value)`

where:

*date*

Is the date to add days to, or a field containing the date.

*value*

Is the number of days by which to increase the date.

This function changes the value of *date*.

#### **Example** Adding Days to a Date

ADD adds 10 days to the each value in the DateVar field:

`ADD(DateVar, 10)`

The following are sample values for DateVar and the corresponding values for ADD(DateVar, 10):

<code>DateVar</code>	<code>ADD(DateVar, 10);</code>
-----	-----
12/31/1999	01/10/2000
01/01/2000	01/11/2000
01/02/2000	01/12/2000

## DAY: Extracting the Day of the Month From a Date

Available Operating Systems: All

Available Languages: Maintain

The DAY function extracts the day of the month from a date and returns the result as an integer.

### **Syntax** How to Extract the Day of the Month From a Date

`DAY (date) ;`

where:

*date*

Is the date (in date format) to extract the day of the month from, or a field containing the date.

### **Example** Extracting the Day of the Month From a Date

DAY extracts the day of the month from the DATE field:

`DAY (DATE)`

The following are sample values for DATE and the corresponding values for DAY(DATE):

<code>DATE</code>	<code>DAY (DATE)</code>
01/01/2000	1
01/02/2000	2
01/03/2000	3

## JULIAN: Determining How Many Days Have Elapsed in the Year

Available Operating Systems: All

Available Languages: Maintain

The JULIAN function determines the number of days that have elapsed in the given year up to a given date, and returns the result as an integer.

### **Syntax** How to Determine How Many Days Have Elapsed in the Year

`JULIAN (date) ;`

where:

*date*

Is the date (in date format) for which to determine the number of days elapsed in the given year, or a field containing the date.

**Example Determining How Many Days Have Elapsed in the Year**

JULIAN determines the number of days that have elapsed up to the date in the DATE field:

`JULIAN (DATE)`

The following are sample values for DATE and the corresponding values for JULIAN(DATE):

<code>DATE</code>	<code>JULIAN (DATE)</code>
----	-----
01/01/2000	1
02/01/2000	32
03/01/2000	61

**MONTH: Extracting the Month From a Date**

Available Operating Systems: All

Available Languages: Maintain

The MONTH function extracts the month from a date and returns the result as an integer.

**Syntax How to Extract the Month From a Date**

`MONTH (date) ;`

where:

*date*

Is the date (in date format) to extract the month from, or a field containing the date.

**Example Extracting the Month From a Date**

MONTH extracts the month from each value in the DATE field:

`MONTH (DATE)`

The following are sample values for DATE and the corresponding values for MONTH(DATE):

<code>DATE</code>	<code>MONTH (DATE)</code>
----	-----
01/01/2000	1
02/01/2000	2
03/01/2000	3

## QUARTER: Determining the Quarter

Available Operating Systems: All

Available Languages: Maintain

The QUARTER function determines the quarter of the year in which a date resides, and returns the result as an integer.

### **Syntax**      **How to Determine the Quarter for a Date**

`QUARTER (date) ;`

where:

*date*

Is the date (in date format) to determine the quarter for, or a field containing the date.

### **Example**      **Determining the Quarter for a Date**

QUARTER extracts the quarter component from each value in the DATE field:

`QUARTER (DATE)`

The following are sample values for DATE and the corresponding values for QUARTER(DATE):

<code>DATE</code>	<code>QUARTER (DATE)</code>
01/01/2000	1
04/01/2000	2
07/01/2000	3

## SETMDY: Setting the Value to a Date

Available Operating Systems: All

Available Languages: Maintain

The SETMDY function sets a value to a date based on numeric values representing a day, month, and year. SETMDY returns a 0 if the function is successful, and a negative number if the function fails.



**Syntax**    **How to Set a Value to a Date**

```
SETMDY(date, month, day, year);
```

or

```
date.SETMDY(month, day, year);
```

where:

*date*

Is the date, in date format, or a field containing the date.

*month*

Is an integer value representing a month.

*day*

Is an integer value representing the day of the month.

*year*

Is an integer value representing a year.

**Example**    **Setting a Value to a Date**

SETMDY sets the value of DateVar, which is formatted as a date that displays as wrMtrDYY (for example, Saturday, January 1, 2000):

```
SETMDY(DateVar, month, day, year);
```

The following are sample values for month, day, and year, and the corresponding dates for DateVar:

month	day	year	DateVar
-----	---	----	-----
04	05	1965	Monday, April 5, 1965
02	01	1997	Saturday, February 1, 1997
01	01	2000	Saturday, January 1, 2000

## **SUB: Subtracting a Value From a Date**

Available Operating Systems: All

Available Languages: Maintain

The SUB function subtracts a given number of days from a date.

### **Syntax**    **How to Subtract a Value From a Date**

`SUB(date, value)`

or

`date.SUB(value)`

where:

*date*

Is the date to subtract the value from, or a field containing the date.

*value*

Is the value to subtract from the date.

### **Example**    **Subtracting Days From a Date**

SUB subtracts 10 days from each value in the DateVar field.

`SUB(DateVar, 10)`

The following are sample values for DateVar and the corresponding values for SUB(DateVar, 10):

<code>DateVar</code>	<code>SUB(DateVar, 10);</code>
-----	-----
12/31/1999	12/21/2000
01/01/2000	12/22/2000
01/02/2000	12/23/2000

## WEEKDAY: Determining the Day of the Week for a Date

Available Operating Systems: All

Available Languages: Maintain

The WEEKDAY function determines the day of the week for a date and returns the result as an integer (1=Monday, 2=Tuesday, and so on).

### **Syntax** How to Determine the Day of the Week for a Date

`WEEKDAY (date) ;`

where:

*date*

Is the date (in date format) to determine the weekday for, or a field containing the date.

### **Example** Determining the Day of the Week for a Date

WEEKDAY determines the day of the week for each date in the DATE field, and stores that day as a number corresponding to a weekday:

`WEEKDAY (DATE)`

The following are sample values for DATE and the corresponding values for WEEKDAY(DATE)

<code>DATE</code>	<code>WEEKDAY (DATE)</code>
----	-----
01/01/2000	6
01/02/2000	7
01/03/2000	1

## **YEAR: Extracting the Year From a Date**

Available Operating Systems: All

Available Languages: Maintain

The YEAR function extracts the year from a date.

### **Syntax**      **How to Extract the Year From a Date**

`YEAR (date) ;`

where:

`date`

Is the date to extract the year from, or a field containing the date.

### **Example**      **Extracting a Year From a Date**

YEAR extracts the year from the DATE field, and stores that year in the YEAR(DATE) field:

`YEAR (DATE)`

The following are sample values for DATE and the corresponding values for YEAR(DATE):

<code>DATE</code>	<code>YEAR (DATE)</code>
01/01/2000	2000
02/01/2001	2001
03/01/2002	2002

---

---

## CHAPTER 9

# Format Conversion Functions

### Topics:

- ATODBL: Converting an Alphanumeric String to Double-Precision Format
- EDIT: Converting the Format of a Field
- FTOA: Converting a Number to Alphanumeric Format
- HEXBYT: Converting a Decimal Integer to a Character
- ITONUM: Converting a Large Binary Integer to Double-Precision Format
- ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format
- ITOZ: Converting a Number to Zoned Format
- PCKOUT: Writing a Packed Number of Variable Length
- UFMT: Converting an Alphanumeric String to Hexadecimal

Format conversion functions convert fields from one format to another. For information on field formats see the *Describing Data* manual.

## **ATODBL: Converting an Alphanumeric String to Double-Precision Format**

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The ATODBL function converts a number in alphanumeric format to decimal (double-precision) format.

### **Syntax**      **How to Convert an Alphanumeric String to Double-Precision Format**

`ATODBL(string, length, outfield)`

where:

*string*

Alphanumeric

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

*length*

Alphanumeric

Is the two character length of *infield* in bytes. This can be a numeric constant, or a field or variable that contains the value. If you specify a numeric constant, enclose it in single quotation marks. The maximum value is 15.

*outfield*

Decimal or Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting an Alphanumeric Field to Double-Precision Format**

ATODBL converts the EMP\_ID field into double-precision format and stores the result in D\_EMP\_ID:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND
EMP_ID AND
COMPUTE D_EMP_ID/D12.2 = ATODBL(EMP_ID, '09', D_EMP_ID);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

LAST_NAME	FIRST_NAME	EMP_ID	D_EMP_ID
-----	-----	-----	-----
SMITH	MARY	112847612	112,847,612.00
JONES	DIANE	117593129	117,593,129.00
MCCOY	JOHN	219984371	219,984,371.00
BLACKWOOD	ROSEMARIE	326179357	326,179,357.00
GREENSPAN	MARY	543729165	543,729,165.00
CROSS	BARBARA	818692173	818,692,173.00

**Example**    **Converting an Alphanumeric Value to Double-Precision Format With MODIFY**

In the following example, the Master File contains the MISSING attribute for the CURR\_SAL field. If you do not enter a value for this field, it is interpreted as the 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, $
.
.
.
```

## ATODBL: Converting an Alphanumeric String to Double-Precision Format

ATODBL converts the value supplied for TCSAL to double-precision format:

```
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 on VM/CMS is:

```
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 follows:

1. For the first transaction, the procedure prompts for an employee ID. You enter 071382660.
2. The procedure displays the last and first name of the employee, STEVENS ALFRED.
3. The procedure prompts for a current salary. You enter N/A.
4. A period displays.
5. For the second transaction, the procedure prompts 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 for a current salary. Enter 45000.
8. \$45,000.00 displays.

## **EDIT: Converting the Format of a Field**

---

Available Operating Systems: OS/390, UNIX, VM/CMS

Available Languages: reporting

The EDIT function converts an alphanumeric field that contains numeric characters to numeric format or converts a numeric field to alphanumeric format. It is useful when you need to manipulate a field using a command that requires a particular format.

When EDIT assigns a converted value to a new field, the format of the new field must correspond to the format of the returned value. For example, if EDIT converts a numeric field to alphanumeric format, you must give the new field an alphanumeric format. For example:

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE) ;
```

EDIT deals with a symbol in the following way:

- When an alphanumeric field is converted to numeric format, a sign or decimal point in the field is acceptable and is 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 achieve the specified field length. Converting a number with more than nine significant digits in floating-point or packed-decimal format may produce an incorrect result.

EDIT also extracts characters from or add characters to an alphanumeric string. For more information, see *EDIT: Extracting or Adding Characters* in Chapter 4.

**Syntax**      **How to Convert the Format of a Field**

```
EDIT(fieldname);
```

where:

*fieldname*

Alphanumeric or Numeric

Is the field name.

**Example**      **Converting From Numeric to Alphanumeric Format**

EDIT converts HIRE\_DATE (a legacy date format) to alphanumeric format. CHGDAT is then able to use the field, which it expects in alphanumeric format:

```
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 output is:

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

## FTOA: Converting a Number to Alphanumeric Format

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The FTOA function converts a number up to 16 digits long from numeric format to alphanumeric format. It retains the decimal positions of a number and right-justifies it with leading spaces. You can also add edit options to a number converted by FTOA.

When using FTOA to convert a number containing decimals to a character string, you must specify an alphanumeric format large enough to accommodate both the integer and decimal portions of the number. For example, a D12.2 format is converted to A14. If the output format is not large enough, decimals are truncated.

### Syntax

### How to Convert a Number to Alphanumeric Format

*FTOA*(*number*, '(*format*)', *outfield*)

where:

*number*

Numeric

Is the number to be converted, or the name of the field that contains the number.

*format*

Alphanumeric

Is the output format of the number enclosed in both single quotation marks and parentheses. Only floating point single-precision and double-precision formats are supported. Include any edit options that you want to appear in the output. The D (floating-point double-precision) format automatically supplies commas.

If you use a field name for this argument, specify the name without quotation marks or parentheses. If you specify a format, the format must be enclosed in parentheses.

*outfield*

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The length of this argument must be greater than the length of *number* and must account for edit options and a possible negative sign.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting From Numeric to Alphanumeric Format**

FTOA converts the GROSS field from floating point double-precision to alphanumeric format and stores the result in ALPHA\_GROSS:

```
TABLE FILE EMPLOYEE
PRINT GROSS AND COMPUTE
ALPHA_GROSS/A15 = FTOA(GROSS, '(D12.2)', ALPHA_GROSS);
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME
WHERE (GROSS GT 800) AND (GROSS LT 2300);
END
```

The output is:

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

---

## HEXBYT: Converting a Decimal Integer to a Character

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The HEXBYT function obtains the ASCII or EBCDIC character equivalent of a decimal integer. It returns a single alphanumeric character in the ASCII or EBCDIC character set. You can use this function to produce characters that are not on your keyboard, similar to the CTRAN function.

The display of special characters depends on your software and hardware; not all special characters may display. Printable ASCII and EBCDIC characters and the integer equivalents are listed in character charts.

**Syntax**    **How to Convert a Decimal Integer to a Character**

```
HEXBYT(input, output)
```

where:

*input*

Numeric

Is the decimal integer to be converted to a single character. A value greater than 255 is treated as the remainder of *input* divided by 256.

*output*

Alphanumeric

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting a Decimal Integer to a Character**

HEXBYT converts LAST\_INIT\_CODE to its character equivalent and stores the result in LAST\_INIT:

```
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 output for an ASCII platform is:

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

## HEXBYT: Converting a Decimal Integer to a Character

The output for an EBCDIC platform is:

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** Inserting Braces for Mainframe

HEXBYT converts the decimal integer 192 to its EBCDIC character equivalent, which is a left brace; and the decimal integer 208 to its character equivalent, which is a right brace. If the value of CURR\_SAL is less than 12000, the value of LAST\_NAME is enclosed in braces.

```
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 output is:

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

## **ITONUM: Converting a Large Binary Integer to Double-Precision Format**

---

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The ITONUM function converts a large binary integer in a non-FOCUS data source to double-precision format. Some programming languages and some non-FOCUS data storage systems use large binary integer formats. However, large binary integers (more than 4 bytes in length) are not supported in the Master File so they require conversion to double-precision format.

You must specify how many of the right-most bytes in the input field are significant. The result is an 8-byte double-precision field.

### **Syntax**

### **How to Convert a Large Binary Integer to Double-Precision Format**

*ITONUM(maxbytes, infield, outfield)*

where:

*maxbytes*

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 ignores the left-most 3 bytes.

6 ignores the left-most 2 bytes.

7 ignores the left-most byte.

*infield*

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats of the field must be A8.

*outfield*

Numeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be *Dn* or *Dn.d*.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

### **Example**    **Converting a Large Binary Integer to Double-Precision Format**

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 named BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The following request converts the field to double-precision format:

```
DEFINE FILE EUROCAR  
MYFLD/D14 = ITONUM(6, BINARYFLD, MYFLD);  
END  
TABLE FILE EUROCAR  
PRINT MYFLD BY CAR  
END
```

## **ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format**

---

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The ITOPACK function converts a large binary integer in a non-FOCUS data source 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. However, large binary integers (more than 4 bytes in length) are not supported in the Master File so they require conversion to packed decimal format.

You must specify how many of the right-most bytes in the input field are significant. The result is an 8-byte packed-decimal field of up to 15 significant numeric positions (for example, P15 or P16.2).

**Limit:** For a field defined as 'PIC 9(15) COMP' or the equivalent (15 significant digits), the maximum number that can be converted is 167,744,242,712,576.



**Syntax**    **How to Convert a Large Binary Integer to Packed-Decimal Format**

`ITOPACK(maxbytes, infield, outfield)`

where:

*maxbytes*

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 ignores the left-most 3 bytes (up to 11 significant positions).

6 ignores the left-most 2 bytes (up to 14 significant positions).

7 ignores the left-most byte (up to 15 significant positions).

*infield*

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats of the field must be A8.

*outfield*

Numeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be *Pn* or *Pn.d*.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **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 in the EUROCAR Master File as a field named BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The following request converts the field to packed-decimal format:

```
DEFINE FILE EUROCAR
PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, PACKFLD);
END
TABLE FILE EUROCAR
PRINT PACKFLD BY CAR
END
```

## IT0Z: Converting a Number to Zoned Format

---

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The IT0Z function converts a number in numeric format to zoned format. Although a request cannot process zoned numbers, it can write zoned fields to an extract file for use by an external program.

### **Syntax**      **How to Convert to a Zoned Format**

*IT0Z(outlength, number, outfield)*

where:

*outlength*

Numeric

Is the length of *number* in bytes. The maximum number of bytes is 15. 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 result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting a Number to Zoned Format**

The following request creates an extract file containing employee IDs and salaries in zoned format for a COBOL program:

```
DEFINE FILE EMPLOYEE
ZONE_SAL/A8 = ITOZ(8, CURR_SAL, ZONE_SAL);
END

TABLE FILE EMPLOYEE
PRINT CURR_SAL 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|ALPHANUMERIC]  RECORD NAMED  SALARIES
FIELDNAME              ALIAS        FORMAT        LENGTH

EMP_ID                 EID          A9             9
CURR_SAL               CSAL         D12.2M        12
ZONE_SAL               A8           A8             8

TOTAL                                     29

DCB USED WITH FILE SALARIES IS DCB=(RECFM=FB,LRECL=00029,BLKSIZE=00580)
```

If you remove the SAVE command and run the request, the output for an EBCDIC platform follows. The left brace in EBCDIC is hexadecimal C0; this indicates a positive sign and a final digit of 0. The capital B in EBCDIC is hexadecimal C2; this indicates a positive sign and a final digit of 2.

```
EMP_ID          CURR_SAL      ZONE_SAL
-----          -
071382660      $11,000.00   0001100{
112847612      $13,200.00   0001320{
117593129      $18,480.00   0001848{
119265415       $9,500.00   0000950{
119329144      $29,700.00   0002970{
123764317      $26,862.00   0002686B
126724188      $21,120.00   0002112{
219984371      $18,480.00   0001848{
326179357      $21,780.00   0002178{
451123478      $16,100.00   0001610{
543729165       $9,000.00   0000900{
818692173      $27,062.00   0002706B
```

## PCKOUT: Writing a Packed Number of Variable Length

---

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The PCKOUT function writes a packed number of variable length to an extract file. When a request saves a packed number to an extract file, it typically writes it as an 8- or 16-byte field regardless of its format specification. With PCKOUT, you can vary the field's length between 1 to 16 bytes.

### **Syntax**      **How to Write a Packed Number of Variable Length**

`PCKOUT(infield, outlength, outfield)`

where:

*infield*

Numeric

Is the input field that contains the values. The field can be in packed, integer, floating-point, or double-precision format. If the field is not in integer format, its values are rounded to the nearest integer.

*outlength*

Numeric

Is the length of *outfield* from 1 to 16 bytes.

*outfield*

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The function returns the field as alphanumeric although it contains packed data.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Writing a Packed Number of Variable Length**

PCKOUT converts the CURR\_SAL field to a 5-byte packed field and stores the result in 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
```

The resulting extract file is:

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

[EBCDIC|ALPHANUMERIC]  RECORD NAMED  SAVE
FIELDNAME              ALIAS          FORMAT          LENGTH

LAST_NAME              LN           A15             15
SHORT_SAL              A5          A5              5
HIRE_DATE              HDT         I6YMD           6

TOTAL                                     26
DCB USED WITH FILE SAVE      IS DCB=(RECFM=FB,LRECL=00026,BLKSIZE=00520)
```

## UFMT: Converting an Alphanumeric String to Hexadecimal

---

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS

Available Languages: reporting, Maintain

The UFMT function converts characters in an alphanumeric field to the hexadecimal representation. This function is useful for examining data of unknown format. As long as you know the length of the data, you can examine its content.

### **Syntax**

### **How to Convert an Alphanumeric String to Hexadecimal**

`UFMT(string, inlength, outfield)`

where:

*string*

Alphanumeric

Is the alphanumeric string to be converted enclosed in single quotation marks, or the field that contains the string.

*inlength*

Numeric

Is the length in characters of *string*.

*outfield*

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format of *outfield* must be alphanumeric and its length must be twice that of *inlength*.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Converting an Alphanumeric String to Hexadecimal**

UFMT converts each value in JOBCODE to its hexadecimal representation and stores the result in HEXCODE:

```
DEFINE FILE JOBFIL  
HEXCODE/A6 = UFMT(JOBCODE, 3, HEXCODE);  
END  
TABLE FILE JOBFIL  
PRINT JOBCODE HEXCODE  
END
```

The output is:

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

*UFMT: Converting an Alphanumeric String to Hexadecimal*



---

---

## CHAPTER 10

### Numeric Functions

#### Topics:

- ABS: Calculating Absolute Value
- ASIS: Distinguishing Between a Blank and a Zero
- BAR: Producing a Bar Chart
- CHKPCK: Validating a Packed Field
- DMOD, FMOD, and IMOD: Calculating the Remainder From a Division
- EXP: Raising “e” to the Nth Power
- EXPN: Evaluating a Number in Scientific Notation
- FMLINFO: Returning FOR Values
- INT: Finding the Greatest Integer
- LOG: Calculating the Natural Logarithm
- MAX and MIN: Finding the Maximum or Minimum Value
- NORMSDST and NORMSINV: Calculating Cumulative Normal Distribution
- PRDNOR and PRDUNI: Generating Reproducible Random Numbers
- RDNORM and RDUNIF: Generating Random Numbers
- SQRT: Calculating the Square Root

Numeric functions perform calculations on numeric constants and fields.

## ABS: Calculating Absolute Value

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The ABS function returns the absolute value of a number.

### **Syntax**    **How to Calculate Absolute Value**

`ABS(argument)`

where:

*argument*

Numeric

Is the value for which the absolute value is returned, the name of a field that contains the value, or an expression that returns the value. If you use an expression, use parentheses as needed to ensure the correct order of evaluation.

### **Example**    **Calculating Absolute Value**

The COMPUTE command creates the DIFF field, then ABS calculates the absolute value of DIFF:

```
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 output is:

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

## ASIS: Distinguishing Between a Blank and a Zero

---

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX

Available Languages: reporting

The ASIS function distinguishes between a blank and a zero in Dialogue Manager. It differentiates between a numeric string constant or variable defined as a numeric string, and a field defined simply as numeric.

For details on ASIS, see *ASIS: Distinguishing Between a Space and a Zero* in Chapter 4.

## BAR: Producing a Bar Chart

---

Available Operating Systems: AS/400, OpenVMS, S/390, UNIX, VM/CMS

Available Languages: reporting, Maintain

The BAR function produces a horizontal bar chart using repeating characters to form each bar. Optionally, you can create a scale to clarify the meaning of a bar chart. Do this by replacing the title of the column containing the bar with a scale.

### Syntax

#### How to Produce a Bar Chart

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

where:

*barlength*

Numeric

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

*infield*

Numeric

Is the data field plotted as a bar chart.

*maxvalue*

Numeric

Is the maximum value of a bar. This value must be greater than the maximum value stored in *infield*. If *infield* is larger than *maxvalue*, the function uses *maxvalue* and returns a bar of maximum length.

## BAR: Producing a Bar Chart

*'char'*

Alphanumeric

Is the repeating character that creates the bars enclosed in single quotation marks. If you specify more than one character, only the first character is used.

*outfield*

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The output field must be large enough to contain a bar of maximum length as defined by *barlength*.

In Dialogue Manager, you must specify the format.

### Example Producing a Bar Chart

BAR creates a bar chart for the CURR\_SAL field, and stores the output in SAL\_BAR. The bar created can be no longer than 30 characters long, and the value it represents can be no greater than 30,000.

```
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 output is:

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**    **Creating a Bar Chart With a Scale**

BAR creates a bar chart for the CURR\_SAL field. The request then replaces the field name SAL\_BAR with a scale using the AS phrase.

To run this request on a platform for which the default font is proportional, use a non-proportional font or issue SET STYLE=OFF.

```

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 output is:

```

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     =====
    
```

## CHKPCK: Validating a Packed Field

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The CHPCK function validates the data in a field described as packed format (if available on your platform). The function prevents a data exception from occurring when a request reads a field that is expected to contain a valid packed number but does not.

To use CHPCK:

1. Ensure that the Master File (USAGE and ACTUAL attributes) or the MODIFY FIXFORM command defines the field as alphanumeric, not packed. This does *not* change the field data, which remains packed, but it enables the request to read the data without a data exception.
2. Call CHPCK to examine the field. The function returns the output to a field defined as packed. If the value it examines is a valid packed number, the function returns the value; if the value is not packed, the function returns an error code.

**Syntax**      **How to Validate a Packed Field**

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

where:

*inlength*

Numeric

Is the length of the packed field. It can be between 1 and 16 bytes.

*infield*

Alphanumeric

Is the name of the packed field. The field is described as alphanumeric, not packed.

*error*

Numeric

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

*outfield*

Packed

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

## Example Validating Packed Data

1. Prepare a data source that includes invalid packed data. The following creates TESTPACK, which contains the PACK\_SAL field. PACK\_SAL is defined as alphanumeric but actually contains packed data. The invalid packed data is stored as AAA.

```
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 following is the result of the preceding request.

```
>
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                  PS            A8              8

TOTAL                                27

[DCB USED WITH FILE TESTPACK IS
DCB=(RECFM=FB,LRECL=00027,BLKSIZE=00540)]
SAVED...
```

2. Create a Master File for the TESTPACK data source. Define the PACK\_SAL field as alphanumeric in the USAGE and ACTUAL attributes.

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



3. Create a request that uses `CHKPCK` to validate the values in the `PACK_SAL` field, and store the result in the `GOOD_PACK` field. Values not in packed format return the error code `-999`. Values in packed format display accurately.

```
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 output is:

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

## DMOD, FMOD, and IMOD: Calculating the Remainder From a Division

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The MOD functions calculate the remainder from a division. Each function returns the remainder in a different format.

The functions use the following formula.

$$\text{remainder} = \text{dividend} - \text{INT}(\text{dividend}/\text{divisor}) * \text{divisor}$$

- `DMOD` returns the remainder as a decimal number.
- `FMOD` returns the remainder as a floating-point number.
- `IMOD` returns the remainder as an integer.

For information on the `INT` function see *INT: Finding the Greatest Integer* on page 10-16.

**Syntax**      **How to Calculate the Remainder From a Division**

*function*(*dividend*, *divisor*, *outfield*)

where:

*function*

Is one of the following:

DMOD returns the remainder as a decimal number.

FMOD returns the remainder as a floating-point number.

IMOD returns the remainder as an integer.

*dividend*

Numeric

Is the number being divided.

*divisor*

Numeric

Is the number dividing the dividend.

*outfield*

Numeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format is determined by the result returned by the specific function.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Calculating the Remainder From a Division**

IMOD divides ACCTNUMBER by 1000 and returns the remainder to LAST3\_ACCT:

```
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 output is:

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

## EXP: Raising “e” to the Nth Power

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The EXP function raises the value “e” (approximately 2.72) to a specified power. This function is the inverse of the LOG function, which returns an argument’s logarithm.

EXP calculates the result by adding terms of an infinite series. If a term adds less than .000001 percent to the sum, the function ends the calculation and returns the result as a double-precision number.

**Syntax**      **How to Raise “e” to the Nth Power**

`EXP(power, outfield)`

where:

*power*

Numeric

Is the power that “e” is raised to.

*outfield*

Double-precision

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**      **Raising “e” to the Nth Power**

EXP raises “e” to the power designated by the &POW variable, specified here as 3. The result is then rounded to the nearest integer with the .5 rounding constant and returned to the variable &RESULT. The format of the output value is D15.3.

```
-SET &POW = '3';  
-SET &RESULT = EXP(&POW, 'D15.3') + 0.5;  
-TYPE E TO THE &POW POWER IS APPROXIMATELY &RESULT
```

The output is:

```
E TO THE 3 POWER IS APPROXIMATELY 20
```

## EXPN: Evaluating a Number in Scientific Notation

---

Available Operating Systems: AS/400, OpenVMS, OS/390

Available Languages: reporting

The EXPN function evaluates a number expressed in scientific notation.

### **Syntax** How to Evaluate a Number in Scientific Notation

`EXPN(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.

+, -

Indicates if *p* is positive or negative.

*p*

Is the power of 10 to which to raise the number. Supply the actual value, the name of a field that contains the value, or an expression that returns the value. The expression can also call a function.

### **Example** Evaluating a Number in Scientific Notation

You can use scientific notation to express 103 as:

`1.03E+2`

Then

`EXPN(1.03E+2)`

returns 103 as the result.

## FMLINFO: Returning FOR Values

---

Available Operating Systems: All

Available Languages: reporting

The FMLINFO function returns the FOR value associated with each row in an FML report. With FMLINFO, you can use the appropriate FOR value in a COMPUTE command to do drill-downs and sign changes for each row in the report, even when the row is a summary row created using an OR list or a Financial Modeling Language (FML) Hierarchy ADD command.

**Note:** The FORMULTIPLE SET parameter FORMULTIPLE must be set to ON in order to use the FMLINFO function. This enables an incoming record to be used on more than one line in an FML report.

### **Syntax**

### **How to Retain FOR Values in an FML Request**

```
FMLINFO('FORVALUE',outfield)
```

where:

**'FORVALUE'**

Alphanumeric

Returns the FOR value associated with each row in an FML report. If the FML row was generated as a sum of data records using the OR phrase, FMLINFO returns the first FOR value specified in the list of values. If the OR phrase was generated by an FML Hierarchy ADD command, FMLINFO returns the FOR value associated with the parent specified in the ADD command.

*outfield*

Alphanumeric

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

**Example Retrieving FOR Values for FML Hierarchy Rows**

The following request creates a field called PRINT\_AMT that is the negative of the NAT\_AMOUNT field for account numbers less than 2500 in the CENTSYSF data source. The CENTGL data source contains the hierarchy information for CENTSYSF. Therefore, CENTGL is joined to CENTSYSF for the request:

```

SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10 IN 30
COMPUTE PRINT_AMT/D10 = IF FMLINFO('FORVALUE', 'A7') LT '2500'
    THEN 0-NAT_AMOUNT ELSE NAT_AMOUNT;
COMPUTE FORV/A4 = FMLINFO('FORVALUE', 'A4');
COMPUTE ACTION/A9 = IF FORV LT '2500'
    THEN 'CHANGED' ELSE 'UNCHANGED';
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD AS CAPTION
END
    
```

Note that the parent value specified in the WITH CHILDREN ADD command (2000) is returned for the first row on the report. Each subsequent row is also a consolidated subsection of the hierarchy with a parent value that is returned by FMLINFO:

	Month Actual	PRINT_AMT	FORV	ACTION
	-----	-----	----	-----
Gross Margin	-25,639,223	25,639,223	2000	CHANGED
Sales Revenue	-62,362,490	62,362,490	2100	CHANGED
Retail Sales	-49,355,184	49,355,184	2200	CHANGED
Mail Order Sales	-6,899,416	6,899,416	2300	CHANGED
Internet Sales	-6,107,890	6,107,890	2400	CHANGED
Cost Of Goods Sold	36,723,267	36,723,267	2500	UNCHANGED
Variable Material Costs	27,438,625	27,438,625	2600	UNCHANGED
Direct Labor	6,176,900	6,176,900	2700	UNCHANGED
Fixed Costs	3,107,742	3,107,742	2800	UNCHANGED

**Example Using FMLINFO With an OR Phrase**

The FOR value printed for the summary line is 1010, but FMLINFO returns the first value specified on the OR list, 1030:

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLINFO('FORVALUE', 'A8');
FOR ACCOUNT
1010                                OVER
1020                                OVER
1030                                OVER
BAR                                 OVER
1030 OR 1020 OR 1010
END
```

The output is:

	AMOUNT	RETURNEDFOR
1010	8,784	1010
1020	4,494	1020
1030	7,961	1030
	-----	-----
1010	21,239	1030

---

## INT: Finding the Greatest Integer

Available Operating Systems: All

Available Languages: reporting, Maintain

The INT function returns the integer component of a number.

**Syntax How to Find the Greatest Integer**

```
INT(argument)
```

where:

*argument*

Numeric

Is the value for which the integer component is returned, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation.



**Example Finding the Greatest Integer**

INT finds the greatest integer in the DED\_AMT field and stores it in INT\_DED\_AMT:

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND COMPUTE
INT_DED_AMT/I9 = INT(DED_AMT);
BY LAST_NAME BY FIRST_NAME
WHERE (DEPARTMENT EQ 'MIS') AND (PAY_DATE EQ 820730);
END
```

The output is:

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

**LOG: Calculating the Natural Logarithm**

---

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, VM/CMS,

Available Languages: reporting, Maintain

The LOG function returns the natural logarithm of a number.

**Syntax How to Calculate the Natural Logarithm**

`LOG(argument)`

where:

*argument*

Numeric

Is the value for which the natural logarithm is calculated, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation. If *argument* is less than or equal to 0, LOG returns 0.

### Example **Calculating the Natural Logarithm**

LOG calculates the logarithm of the CURR\_SAL field:

```
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 output is:

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

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The MAX and MIN functions return the maximum or minimum value, respectively, from a list of values.

### Syntax **How to Find the Maximum or Minimum Value**

```
{MAX|MIN}(argument1, argument2, ...)
```

where:

**MAX**

Returns the maximum value.

**MIN**

Returns the minimum value.

*argument1, argument2*

Numeric

Are the values of which the maximum or minimum value is returned, the name of a field that contains the values, or an expression that returns the values. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation.

**Example Determining the Minimum Value**

MIN returns either the value of the ED\_HRS field or the constant 30, whichever is lower:

```
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
```

The output is:

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

## **NORMSDST and NORMSINV: Calculating Cumulative Normal Distribution**

---

Available Operating Systems: All

Available Languages: reporting

The NORMSDST and NORMSINV functions perform calculations on a standard normal distribution curve:

- The NORMSDST function calculates the percentage of data values that are less than or equal to a normalized value. A normalized value is a point on the x-axis of a standard normal distribution curve in standard deviations from the mean. This is useful for determining percentiles in normally distributed data.
- The NORMSINV function finds the normalized value that forms the upper boundary of a percentile in a standard normal distribution curve. This is the inverse of NORMSDST.

The results of NORMSDST and NORMSINV are returned as double-precision and are accurate to 6 significant digits.

A standard normal distribution curve is a normal distribution that has a mean of 0 and a standard deviation of 1. The total area under this curve is 1. A point on the x-axis of the standard normal distribution is called normalized value. Assuming that your data is normally distributed, you can convert a data point to a normalized value in order to find the percentage of scores that are less than or equal to the raw score.

You can convert a value (raw score) from your normally distributed data to the equivalent normalized value (z-score) as follows:

$$z = (\text{raw\_score} - \text{mean}) / \text{standard\_deviation}$$

To convert from a z-score back to a raw score, use the following formula:

$$\text{raw\_score} = z * \text{standard\_deviation} + \text{mean}$$

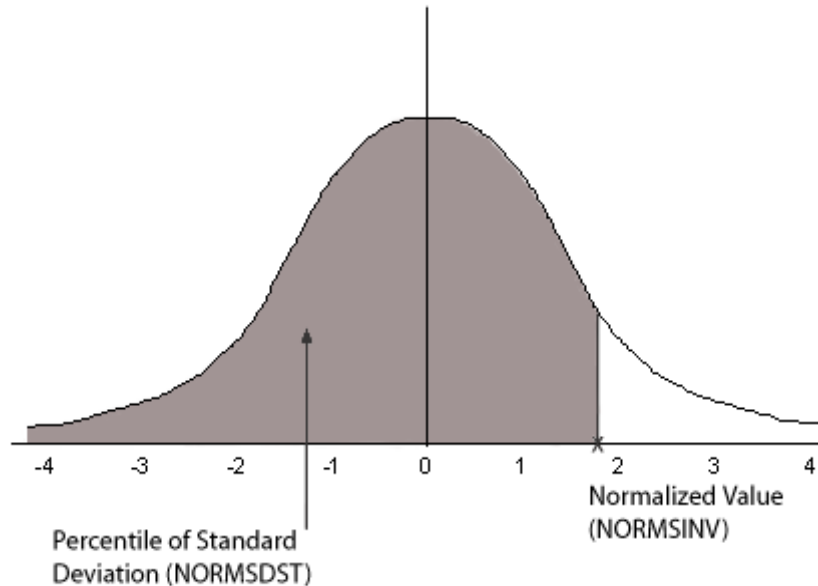
The mean of data points  $x_i$ , where  $i$  is from 1 to  $n$  is:

$$(\sum x_i) / n$$

The standard deviation of data points  $x_i$ , where  $i$  is from 1 to  $n$  is:

$$\text{SQRT} ( ((\sum x_i^2 - (\sum x_i)^2 / n) / (n - 1)))$$

The result of the NORMSDST and NORMSINV functions are illustrated by the following diagram:



## Reference Characteristics of the Normal Distribution

Many common measurements tend to be normally distributed. A plot of normally distributed data values approximates a bell-shaped curve. The two measures needed to describe any normal distribution are the mean and the standard deviation:

- The *mean* is the point at the center of the curve.
- The *standard deviation* describes the spread of the curve. It is the distance from the mean to the point of inflection (point where the curve changes direction).

**Syntax**    **How to Calculate the Cumulative Standard Normal Distribution Function**

```
NORMSDST(value, 'D8');
```

where:

*value*

Is a normalized value.

D8

Is the required format for the result. The value returned by the function is double precision. You can assign it to a field with any valid numeric format.

**Syntax**    **How to Calculate the Inverse Cumulative Standard Normal Distribution Function**

```
NORMSINV(value, 'D8');
```

where:

*value*

Is a number between 0 and 1 which represents the a percentile in a standard normal distribution).

D8

Is the required format for the result. The value returned by the function is double precision. You can assign it to a field with any valid numeric format.

**Example Using the NORMSDST and NORMSINV Functions**

NORMSDST finds the percentile for the Z field. NORMSINV then returns this percentile to a normalized value.

```

DEFINE FILE GGPRODS
  -* CONVERT SIZE FIELD TO DOUBLE PRECISION
  X/D12.5 = SIZE;
  END

TABLE FILE GGPRODS
  SUM X NOPRINT CNT.X NOPRINT
  -* CALCULATE MEAN AND STANDARD DEVIATION
  COMPUTE NUM/D12.5 = CNT.X; NOPRINT
  COMPUTE MEAN/D12.5 = AVE.X; NOPRINT
  COMPUTE VARIANCE/D12.5 = ((NUM*ASQ.X) - (X*X/NUM))/(NUM-1); NOPRINT
  COMPUTE STDEV/D12.5 = SQRT(VARIANCE); NOPRINT

  PRINT SIZE X NOPRINT
  -* COMPUTE NORMALIZED VALUES AND USE AS INPUT TO NORMSDST FUNCTION
  -* THEN USE RETURNED VALUES AS INPUT TO NORMSINV FUNCTION
  -* AND CONVERT BACK TO DATA VALUES
  COMPUTE Z/D12.5 = (X - MEAN)/STDEV;
  COMPUTE NORMSD/D12.5 = NORMSDST(Z, 'D8');
  COMPUTE NORMSI/D12.5 = NORMSINV(NORMSD, 'D8');
  COMPUTE DSIZE/D12 = NORMSI * STDEV + MEAN;
  BY PRODUCT_ID NOPRINT
  END
    
```

The output is:

Size	Z	NORMSD	NORMSI	DSIZE
12	-.80273	.21106	-.80273	12
12	-.80273	.21106	-.80273	12
16	-.07298	.47091	-.07298	16
20	.65678	.74434	.65678	20
24	1.38654	.91721	1.38654	24
20	.65678	.74434	.65678	20
24	1.38654	.91721	1.38654	24
16	-.07298	.47091	-.07298	16
12	-.80273	.21106	-.80273	12
8	-1.53249	.06270	-1.53249	8

## PRDNOR and PRDUNI: Generating Reproducible Random Numbers

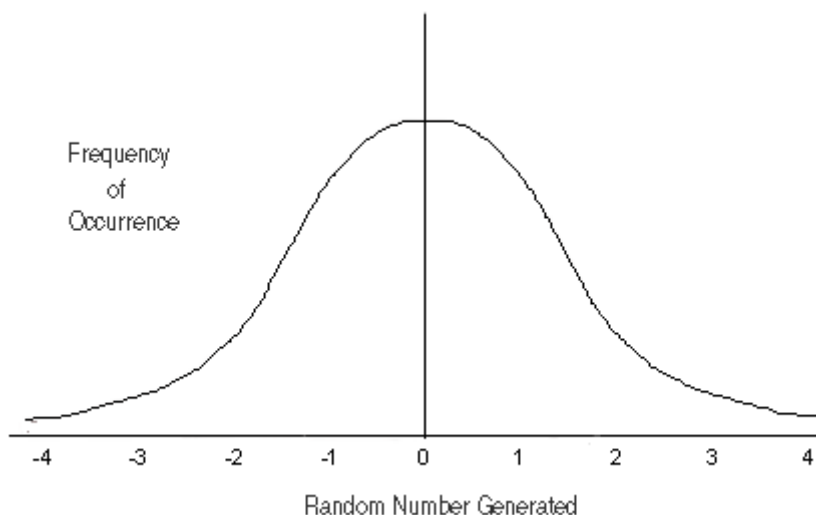
---

Available Operating Systems: All

Available Languages: reporting, Maintain

The PRDNOR and PRDUNI functions generate reproducible random numbers:

- PRDNOR generates reproducible double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1. If PRDNOR generates a large set of numbers, they have the following properties:
  - The numbers lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, meaning that there are more numbers closer to 0 than farther away.



- The average of the numbers is close to 0.
- The numbers can be any size, but most are between 3 and -3.
- PRDUNI generates reproducible 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).

VM/CMS behavior differs from OS/390 behavior. In VM/CMS, the seed changes upon multiple executions as the function is reloaded unless you reissue the DEFINE for each execution. In OS/390, the numbers do not reproduce.

**Syntax**      **How to Generate Reproducible Random Numbers**

`{PRDNOR|PRDUNI}(seed, outfield)`

where:

*PRDNOR*

Generates reproducible double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

*PRDUNI*

Generates reproducible double-precision random numbers uniformly distributed between 0 and 1.

*seed*

Numeric

Is the seed or the field that contains the seed, up to 9 digits. The seed is truncated to an integer.

On MVS, the numbers do not reproduce.

On CMS, the numbers reproduce only if the DEFINE that calls the function is reissued each time you run the request.

*outfield*

Double-precision

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.



**Example**    **Generating Reproducible Random Numbers**

PRDNOR assigns random numbers and stores them in RAND. These values are then used to randomly pick five employee records identified by the values in the LAST NAME and FIRST NAME fields. The seed is 40. To produce a different set of numbers, change the seed.

```
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 output is:

RAND	LAST_NAME	FIRST_NAME
----	-----	-----
1.38	STEVENS	ALFRED
1.12	MCCOY	JOHN
.55	SMITH	RICHARD
.21	JONES	DIANE
.01	IRVING	JOAN

## **RDNORM and RDUNIF: Generating Random Numbers**

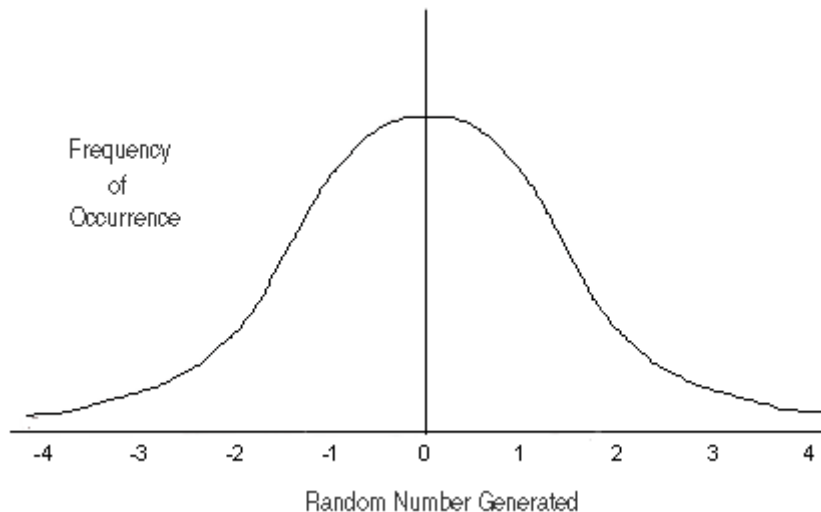
---

Available Operating Systems: All

Available Languages: reporting, Maintain

The RDNORM and RDUNIF functions generate random numbers:

- RDNORM generates double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1. If RDNORM generates a large set of numbers (between 1 and 32768), they have the following properties:
  - The numbers lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, meaning that there are more numbers closer to 0 than farther away.



- The average of the numbers is close to 0.
- The numbers can be any size, but most are between 3 and -3.
- 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).

**Syntax**    **How to Generate Random Numbers**

```
{RDNORM|RDUNIF} (outfield)
```

where:

**RDNORM**

Generates double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

**RDUNIF**

Generates double-precision random numbers uniformly distributed between 0 and 1.

*outfield*

Double-precision

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

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Generating Random Numbers**

RDNORM assigns random numbers and stores them in RAND. These numbers are then used to randomly choose five employee records identified by the values in the LAST NAME and FIRST NAME fields.

```
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
```

The request produces output similar to the following:

RAND	LAST_NAME	FIRST_NAME
----	-----	-----
.65	CROSS	BARBARA
.20	BANNING	JOHN
.19	IRVING	JOAN
.00	BLACKWOOD	ROSEMARIE
-.14	GREENSPAN	MARY

## SQRT: Calculating the Square Root

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The SQRT function calculates the square root of a number.

### **Syntax**    **How to Calculate the Square Root**

`SQRT(argument)`

where:

*argument*

Numeric

Is the value for which the square root is calculated, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation. If you supply a negative number, the result is zero.

### **Example**    **Calculating the Square Root**

SQRT calculates the square root of LISTPR:

```
TABLE FILE MOVIES
PRINT LISTPR AND COMPUTE
SQRT_LISTPR/D12.2 = SQRT(LISTPR);
BY TITLE
WHERE CATEGORY EQ 'MUSICALS';
END
```

The output is:

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

---

---

## CHAPTER 11

# System Functions

### Topics:

- FEXERR: Retrieving an Error Message
- FGETENV: Retrieving the Value of an Environment Variable
- FINDMEM: Finding a Member of a Partitioned Data Set
- FPUTENV: Assigning a Value to an Environment Variable
- GETPDS: Determining If a Member of a Partitioned Data Set Exists
- GETUSER: Retrieving a User ID
- HHMMSS: Retrieving the Current Time
- MVSDYNAM: Passing a DYNAM Command to the Command Processor
- TODAY: Returning the Current Date

System functions call the operating system to obtain information about the operating environment or to use a system service.

## FEXERR: Retrieving an Error Message

---

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS,

Available Languages: reporting, Maintain

The FEXERR function retrieves an Information Builders error message. It is especially useful in a procedure using a command that suppresses the display of output messages.

An error message consists of up to four lines of text; the first line contains the message and the remaining three contain a detailed explanation if one exists. FEXERR retrieves the first line of the error message.

### **Syntax**    **How to Retrieve an Error Message**

`FEXERR (error, 'A72')`

where:

`error`

Numeric

Is the error number, up to 5 digits long.

`'A72'`

Is the format of the output value enclosed in single quotation marks. The format is A72 because the maximum length of an Information Builders error message is 72 characters.

In Maintain, you must supply the field name instead.

### **Example**    **Retrieving an Error Message**

FEXERR retrieves the error message whose number is contained in the &ERR variable, in this case 650. The result is returned to the variable &&MSGVAR and has the format A72.

```
-SET &ERR = 650;  
-SET &&MSGVAR = FEXERR(&ERR, 'A72');  
-TYPE &&MSGVAR
```

The output is:

```
(FOC650) THE DISK IS NOT ACCESSED
```

## **FGETENV: Retrieving the Value of an Environment Variable**

---

Available Operating Systems: AS/400, OpenVMS, UNIX

Available Languages: reporting

The FGETENV function retrieves the value of an environment variable and returns it as an alphanumeric string.

### ***Syntax* How to Retrieve the Value of an Environment Variable**

`FGETENV(varlength, 'varname', outfieldlen, outfldformat)`

where:

*varlength*

Integer

Is the length of the environment variable name.

*varname*

Alphanumeric

Is the name of the environment variable.

*outfieldlen*

Integer

Is the length of the field in which the environment variable's value is stored.

*outfldformat*

Alphanumeric

Is the format of the field in which the environment variable's value is stored.

## FINDMEM: Finding a Member of a Partitioned Data Set

---

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

The FINDMEM function, used on OS/390, determines if a specific member of a partitioned data set (PDS) exists. This function is used primarily in Dialogue Manager procedures.

To use this function, allocate the PDS to a ddname because the ddname is required in the function call. You can search multiple PDSs with one function call if they are concatenated to one ddname.

### 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 value must be an 8-character literal enclosed in single quotation marks, or a variable that contains the ddname. If you supply a literal less than 8 characters long, pad it with trailing spaces.

*member*

A8

Is the member you are searching for. This value must be 8 characters long. If you supply a literal that has less than 8 characters, pad it with trailing spaces.

*outfield*

A1

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The result is one of the following:

Y indicates the member exists in the PDS.

N indicates the member does not exist in the PDS.

E indicates an error occurred. Either the data set is not allocated to the ddname, or the data set allocated to the ddname is not a PDS (and may be a sequential file).

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.



**Example Finding a Member of a Partitioned Data Set**

FINDMEM searches for the EMPLOYEE Master File in the PDS allocated to ddname MASTER, and returns the result to the variable &FINDCODE. The result has the format A1:

```
-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 RECDLIMIT 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
```

The output is:

```
MEMBER EXISTS, RETURN CODE = Y
> NUMBER OF RECORDS IN TABLE=          4  LINES=          4

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
```

## FPUTENV: Assigning a Value to an Environment Variable

---

Available Operating Systems: AS/400, OpenVMS, Tandem, UNIX

Available Languages: reporting

The FPUTENV function assigns a character string to an environment variable. Use FPUTENV to set values that are used elsewhere in the system.

**Limit:** You cannot use FPUTENV to set or change FOCPRINT, FOCPATH, or USERPATH; once started, these variables are held in memory and not reread from the environment.

### Syntax

### How to Assign a Value to an Environment Variable

```
FPUTENV (namelength, 'name', valuelength, 'value', outfield)
```

where:

*namelength*

Integer

Is the maximum length of the name of the environment variable.

*name*

Alphanumeric

Is the name of the environment variable enclosed in single quotation marks. The name must be right-justified and padded with blanks to the maximum length specified by *namelength*.

*valuelength*

Is the maximum length of the environment variable value.

**Note:** The sum of *namelength* and *valuelength* cannot exceed 64.

*value*

Is the value you wish to assign to the environment variable. The string must be right-justified and contain no embedded blanks. Strings that contain embedded blanks are truncated at the first blank.

*outfield*

Integer

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. If the variable is set successfully, the return code is 0; any other value indicates a failure occurred.

In Dialogue Manager, you must specify the format.

**Example Assigning a Value to an Environment Variable**

FPUTENV assigns the value FOCUS/Shell to the PS1 variable and stores it in a field with the format A12:

```
-SET &RC = FPUTENV(3,'PS1', 12 'FOCUS/Shell:', 'A12');
```

The request displays the following prompt when the user issues the UNIX shell command SH:

```
FOCUS/Shell:
```

---

**GETPDS: Determining If a Member of a Partitioned Data Set Exists**

---

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

The GETPDS function determines if a specific member of a partitioned data set (PDS) exists, and if it does, returns the PDS name. This function is used primarily in Dialogue Manager procedures.

To use this function, allocate the PDS to a ddname because the ddname is required in the function call. You can search multiple PDSs with one function call if they are concatenated to one ddname.

GETPDS is almost identical to FINDMEM, except that GETPDS provides either the PDS name or returns a different set of status codes.

**Syntax**      **How to Determine if a PDS Member Exists**

`GETPDS(ddname, member, outfield)`

where:

*ddname*

A8

Is the ddname to which the PDS is allocated. This value must be an 8-character literal enclosed in single quotation marks, or a variable that contains the ddname. If you supply a literal less than 8 characters long, pad it with trailing spaces.

*member*

A8

Is the member the function searches for. This value must be 8 characters long. If you supply a literal with less than 8 characters, pad it with trailing spaces.

*outfield*

A44

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The maximum length of a PDS name is 44. The result is one of the following:

PDS name is the name of the PDS that contains the member, if it exists.

\*D indicates the ddname is not allocated to a data set.

\*M indicates the member does not exist in the PDS.

\*E indicates an error occurred. For example, the data set allocated to the ddname is not a PDS (and may be a sequential file).

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example Determining if a PDS Member Exists**

GETPDS searches for the member specified by &MEMBER in the PDS allocated to &DDNAME, and returns the result to &PNAME. The result has the format A44.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '          ';
-SET &PNAME = GETPDS(&DDNAME, &MEMBER, 'A44');
-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
```

Output similar to the following is produced:

```
MEMBER EMPLOYEE IS FOUND IN
THE PDS USER1.MASTER.DATA
ALLOCATED TO MASTER
```

### Example Copying a Member for Editing in TED

GETPDS searches for the member specified by &MEMBER in the PDS allocated to &DDNAME, and returns the result to &PNAME. The DYNAM commands copy the member from the production PDS to the 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; the local PDS to ddname MYMASTER. 05sub12.fex.

```
-* 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 File 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, 'A44');
-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 session, allocate the ddnames:

```
> > tso alloc f(master) da('prod720.master.data') shr
> > tso alloc f(mymaster) da('user1.master.data') shr
```

Run the procedure, and specify the EMPLOYEE member. It is copied to your local PDS, and you access 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** Displaying the Attributes of a PDS

To view the attributes of the PDS that contains a specific member, this Dialogue Manager procedure can search for the EMPLOYEE member in the PDS allocated to the ddname MASTER and, based on its existence, allocate the PDS to the ddname TEMPMAST. Dialogue Manager system variables are used to display the attributes.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '                ';
-SET &PNAME = GETPDS(&DDNAME, &MEMBER, 'A44');
-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
```

Sample output is:

```
> THE DATA SET ATTRIBUTES INCLUDE:  
DATA SET NAME IS: USER1.MASTER.DATA  
VOLUME IS: USERMO  
DISPOSITION IS: SHR  
>
```

## GETUSER: Retrieving a User ID

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The GETUSER function retrieves the ID of the connected user. GETUSER can also retrieve the name of an S/390 batch job if you run the function from the batch job. To retrieve a logon ID for MSO, use the MSOINFO function described in the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*.

### **Syntax**

#### **How to Retrieve a User ID**

```
GETUSER(outfield)
```

where:

*outfield*

A8

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The field must be 8 bytes long.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.



**Example Retrieving a User ID**

GETUSER retrieves the user ID of the person running 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 output is:

```
SALARY REPORT RUN FROM USERID: USER1

DEPARTMENT    TOTAL SALARIES
-----
MIS            $108,002.00
PRODUCTION    $114,282.00
```

**HHMMSS: Retrieving the Current Time**

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The HHMMSS function retrieves the current time from the operating system as an 8-character string, separating the hours, minutes, and seconds with periods for reporting and colons for Maintain.

For details on how to use HHMMSS in reporting, see *HHMMSS: Retrieving the Current Time* in Chapter 7. For details on how to use HHMMSS in Maintain, see *HHMMSS: Retrieving the Current Time (Maintain)* in Chapter 8.

## MVSDYNAM: Passing a DYNAM Command to the Command Processor

---

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

The MVSDYNAM function transfers a FOCUS DYNAM command to the DYNAM command processor. It is useful in passing allocation commands to the processor in a compiled MODIFY procedure after the CASE AT START command.

### **Syntax**

#### **How to Pass a DYNAM Command to the Command Processor**

`MVSDYNAM(command, length, outfield)`

where:

*command*

Alphanumeric

Is the DYNAM command enclosed in single quotation marks, or a field or variable that contains the command. The function converts lowercase input to uppercase.

*length*

Numeric

Is the maximum length of the command in characters, between 1 and 256.

*outfield*

I4

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

MVSDYNAM returns one of the following codes:

0 indicates the DYNAM command transferred and executed successfully.

*positive number* is the error number corresponding to a FOCUS error.

*negative number* is the FOCUS error number corresponding to a DYNAM failure.

In Dialogue Manager, you must specify the format.

**Example** **Passing a DYNAM Command to the Command Processor**

MVSDYNAM passes the DYNAM command contained in LINE to the processor. The return code is stored in RES.

```

-* 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 ALLOC command:

```
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
BLKSWRITTEN = 96
FOCUSPAGES  = 8
ENTER A SPACE TO CONTINUE >
```

Type one space and press *Enter* to continue. Then enter the DYNAM FREE command (the DYNAM keyword is assumed):

```
ENTER DYNAM COMMAND BELOW:
free file car
```

The function successfully passes the DYNAM FREE command to the processor and the return code displays:

```
THE RESULT OF DYNAM WAS 0
```

Press *Enter* to continue. The second query command indicates that the allocation was 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
BLKSWRITTEN =          0
>
```

## TODAY: Returning the Current Date

---

Available Operating Systems: All

Available Languages: reporting, Maintain

The TODAY function retrieves the current date from the system.

For details on using TODAY in reporting, see *TODAY: Returning the Current Date* in Chapter 7. For details on using TODAY in Maintain, see *TODAY: Retrieving the Current Date (Maintain)* in Chapter 8.

*TODAY: Returning the Current Date*

---

---

## APPENDIX A

### Creating a Subroutine

#### Topics:

- Writing a Subroutine
- Compiling and Storing a Subroutine
- Testing the Subroutine
- Using a Custom Subroutine: The MTHNAM Subroutine
- Subroutines Written in REXX

You can create custom subroutines to use in addition to the functions provided by Information Builders. The process of creating a subroutine consists of the following steps:

- Writing a subroutine using any language that supports subroutine calls. Some of the most common languages are FORTRAN, COBOL, PL/I, Assembler, and C. For details, see *Writing a Subroutine* on page A-2.
- Compiling the subroutine. For details, see *Compiling and Storing a Subroutine* on page A-11.
- Storing the subroutine in a separate file; do not include it in the main program. For details, *Compiling and Storing a Subroutine* on page A-11.
- Testing the subroutine. For details, see *Testing the Subroutine* on page A-12.

## Writing a Subroutine

---

You can write a subroutine in any language that supports subroutines. 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 subroutine call.

When you write a subroutine you need to consider the requirements and limits that affect it. These are:

- Naming conventions. For details, see *Naming a Subroutine* on page A-3.
- Argument considerations. For details, see *Creating Arguments* on page A-3.
- Language considerations. For details, see *Language Considerations* on page A-4.
- Programming considerations. For details, see *Programming a Subroutine* on page A-6.

If 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. The following report request shows how much money employees would accrue if they invested 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 the current date because a subroutine can return only a single value each time it is called.

## Naming a Subroutine

A subroutine name can be up to eight characters long unless the language you are using to write the subroutine requires a shorter name. A name must start with a letter and can consist of a combination of letters and/or numbers. Special symbols are not permitted.

## Creating Arguments

When you create arguments for a subroutine, you must consider the following issues:

- **Maximum number of arguments.** A subroutine may contain up to 28 arguments. You can bypass this restriction by creating a subroutine that accepts multiple calls as described in *Including More Than 28 Arguments in a Subroutine Call* on page A-8.
- **Argument types.** You can use the same types of arguments in a subroutine as in a function. For details on these argument types, see *Argument Types* in Chapter 3.
- **Input arguments.** Input arguments are passed to a subroutine using standard conventions. Register one point to the list of argument addresses. Each address is a full word.
- **Output arguments.** A subroutine returns only one output argument. This argument must be the last in the subroutine. You can choose any format for the output argument except in Dialogue Manager which requires the argument to have the format of the output field.
- **Internal processing.** A subroutine's arguments are processed as follows:
  - An alphanumeric argument is not changed.
  - A numeric argument is converted to floating-point double-precision format except in an operating system RUN command or when storing the output in a variable.
- **Dialogue Manager requirements.** If you are writing a subroutine specifically for Dialogue Manager, the subroutine may need to perform a conversion. For details on using a subroutine with Dialogue Manager, see *Calling a Function From a Dialogue Manager Command* in Chapter 3.

## Language Considerations

When writing a subroutine, you must consider the following language issues:

**Language and memory.** If you write a 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.** In VM/CMS, FORTRAN input/output operations are not supported. If a subroutine written in FORTRAN must read or write data, write the I/O portions in a separate subroutine in another language. However, TSO supports FORTRAN input/output operations.

**PL/I.** When writing a subroutine in PL/I:

- The RETURNS attribute cannot be used.
- The following attribute must be in the procedure (PROC) statement:

`OPTIONS (COBOL)`

- Alphanumeric arguments received from a request must be declared as

`CHARACTER (n)`

where:

*n*

Is the field length as defined by the request. Do not use the VARYING attribute.

- Numeric arguments received from a request must be declared as

`DECIMAL FLOAT (16)`

or

`BINARY FLOAT (53)`

- The format described in the DEFINE or COMPUTE command determines the format of the output argument:

FOCUS Format	PL/I Declaration for Output
<i>An</i>	CHARACTER ( <i>n</i> )
I	BINARY FIXED (31)
F	DECIMAL FLOAT (6) or BINARY FLOAT (21)
D	DECIMAL FLOAT (16) or BINARY FLOAT (53)
P	DECIMAL FIXED (15) (for small packed numbers, 8 bytes) DECIMAL FIXED (31) (for large packed numbers, 16 bytes)

- Variables that are not arguments with the STATIC attribute must be declared. This avoids dynamically allocating these variables every time the subroutine is executed.

**C language.** When writing a subroutine in C:

- Do not return a value with the return statement.
- Declare double-precision fields as Double.
- The format defined in the DEFINE or COMPUTE command determines the format of the output argument:

FOCUS Format	C Declaration for Output
<i>An</i>	char *xxx <i>n</i> Alphanumeric fields are not terminated with a null byte and cannot be processed by many of the string manipulation subroutines in the run-time library.
I	long *xxx
F	float *xxx
D	double *xxx
P	No equivalent in C.

## Programming a Subroutine

Consider the following when planning your programming requirements:

- Write the subroutine to include an argument that specifies the output field.
- If the subroutine initializes a variable, it must initialize it each time it is executed (serial reusability).
- Since a single request may execute a subroutine numerous times, code the subroutine as efficiently as possible.
- If you create your subroutine in a text file or text library, the subroutine must be 31-bit addressable.
- The last argument, normally used for returning the result of the subroutine to FOCUS, can also be used to provide input from FOCUS to the subroutine.

You can add flexibility to your subroutine by using a programming technique. A programming technique can be one of the following:

- Executing a subroutine at an entry point. An entry point enables you to use one algorithm to produce different results. For details, see *Executing a Subroutine at an Entry Point* on page A-6.
- Creating a subroutine with multiple subroutine calls. Multiple calls enable the subroutine to process more than 28 arguments. For details, see *Including More Than 28 Arguments in a Subroutine Call* on page A-8.

### Executing a Subroutine at an Entry Point

A subroutine is usually executed starting from the first statement. However, a subroutine can be executed starting from any place in the code designated as an *entry point*. This enables a subroutine to use one basic algorithm to produce different results. For example, the DOWK subroutine calculates the day of the week on which a date falls. By specifying 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.

Each entry point has a name. To execute a subroutine at an entry point, specify the entry point name in the subroutine call instead of the subroutine name. How you designate an entry point depends on the language you are using.

In VM/CMS, a subroutine can be executed from its entry points only if the subroutine is stored in a library.

**Syntax**    **How to Execute a Subroutine at an Entry Point**

```
{subroutine|entrypoint} (input1, input2,...outfield)
```

where:

*subroutine*

Is the name of the subroutine.

*entrypoint*

Is the name of the entry point to execute the subroutine at.

*input1, input2,...*

Are the subroutine's arguments.

*outfield*

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

**Example**    **Executing a Subroutine at an Entry Point**

The FTOC subroutine, written in pseudocode below, converts Fahrenheit temperature to Centigrade. The entry point FTOK (designated by the Entry command) sets a flag that causes 273 to be subtracted from the Centigrade temperature to find the 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.
```

The following 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. The subroutine processes as:

```
CENTIGRADE/D6.2 = FTOC (TEMPERATURE, CENTIGRADE);
```

To obtain the Kelvin temperature, specify the entry name FTOK in the subroutine call. The subroutine processes as:

```
KELVIN/D6.2 = FTOK (TEMPERATURE, KELVIN);
```

## Including More Than 28 Arguments in a Subroutine Call

A subroutine can specify a maximum of 28 arguments including the output argument. To process more than 28 arguments, the subroutine must specify two or more call statements to pass the arguments to the subroutine.

Use the following technique for writing a subroutine with multiple calls:

1. Divide the subroutine into segments. Each segment receives 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.

Each 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. 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, each segment processes six arguments; the sixth segment processes two arguments and four dummy arguments.

Subroutines may require additional arguments as determined by the programmer who creates the subroutine.

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 one branches to the first segment, number two 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 command returning control back to the request (RETURN command).

4. The last segment returns the final output value to the request.

You can also use the entry point technique to write subroutines that process more than 28 arguments. For details, see *Executing a Subroutine at an Entry Point* on page A-6.

**Syntax**    **How to Create a Subroutine With Multiple Call Statements**

```

field = subroutine (1, group1, field);
field = subroutine (2, group2, field);
.
.
.
outfield = subroutine (n, groupn, outfield);

```

where:

*field*

Is the name of the field that contains the result of the segment or the format of the field enclosed in single quotation marks. This field must have the same format as *outfield*.

Do not specify *field* for the last call statement; use *outfield*.

*subroutine*

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

*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*, *group2*, ...

Are lists of input arguments passed by each subroutine call. Each group contains the same number of arguments, and no more than 26 arguments each.

The final group may contain dummy arguments.

*outfield*

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format. In Maintain, you must specify the name of the field.

### **Example**    **Creating a Subroutine With 32 Arguments**

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 result. 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. 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
```

## Compiling and Storing a Subroutine

---

After you write a subroutine, you need to compile and store it. This topic discusses compiling and storing your subroutine for VM/CMS, and OS/390.

### Compiling and Storing a Subroutine on VM/CMS

Compile the subroutine, then use the GENSUBLL command to add the compiled object code to a load library (file type LOADLIB). Do not store the subroutine in the FUSELIB load library (FUSELIB LOADLIB), as it may be overwritten when you install the next release of your application. You may also store the compiled object code either as a text file (file type TEXT), or as a member in a text library (file type TXTLIB).

### **Reference** Storing a Subroutine in a Text File or Library

You can store a subroutine in two types of text files, a text file or a text library. Individual text files are easier to maintain and control, but text libraries enable you to build different entry points into the subroutine. For details, see *Executing a Subroutine at an Entry Point* on page A-6.

There are two VM/CMS commands for use with a text library:

- The TXTLIB command allows you to create, add to, and delete text libraries.
- The GLOBAL TXTLIB command allows you to specify text libraries to gain access to your 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 subroutine that your subroutine calls is also compiled and placed in text file or library.

**Syntax**    **How to Query the Location of a Compiled Object**

`GENSUBLL ?`

displays online information about the GENSUBLL command.

**Compiling and Storing a Subroutine on OS/390**

Compile the subroutine, then link-edit it 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. 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 your application.

If the subroutine is written in PL/I, include the following when link-editing the subroutine

`ENTRY subroutine`

where:

`subroutine`

Is the name of the subroutine.

**Compiling and Storing a Subroutine on UNIX**

Run the program GENPCPM, which creates a .DLL file. Then check the location of your dynamic link functions library file as specified by the IBICPG environment variable, and save the .DLL file to this location.

**Testing the Subroutine**

---

After compiling and storing a subroutine, you can test it in a report request. In order to access the subroutine, you need to issue the GLOBAL command for VM/CMS or the ALLOCATE command for OS/390.

If an error occurs during testing, check to see if the error is in the request or in the subroutine.

**Procedure**    **How to Determine the Location of Error**

You can determine the location of an error with the following:

1. Write a dummy subroutine that has the same arguments but 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.

## Using a Custom Subroutine: The MTHNAM Subroutine

---

This topic discusses the MTHNAM subroutine as an example. The MTHNAM subroutine converts a number representing a month to the full name of that month. The subroutine processes as follows:

1. Receives the input argument from the request as a double-precision number.
2. Adds .000001 to the number which compensates for rounding errors. Rounding errors can occur since floating-point numbers are approximations and may be inaccurate in the last significant digit.
3. Moves the number into an integer field.
4. If the number is less than one or greater than 12, it changes the number to 13.
5. Defines a list containing the names of months and an error message for the number 13.
6. Sets the index of the list 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. Returns the result as an output field.

### Writing the MTHNAM Subroutine

The MTHNAM subroutine can be written in FORTRAN, COBOL, PL/I, BAL Assembler, and C.

#### **Reference** MTHNAM Subroutine Written in FORTRAN

This is a FORTRAN version of the MTHNAM subroutine where:

**MTH**

Is the double-precision number in the input argument.

**MONTH**

Is the name of the month. Since the character string 'September' contains nine letters, MONTH is a three element array. The subroutine passes the three elements back to your application which concatenates them into one field.

**A**

Is a two dimensional, 13 by three array containing the names of the months. The last three elements contain the error message.

**IMTH**

Is the integer representing the month.

The subroutine 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
```

## Reference **MTHNAM Subroutine Written in COBOL**

This is a COBOL version of the MTHNAM subroutine where:

**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 in the input argument.

MONTH

Is the name of the month corresponding to the integer in IMTH.

The subroutine 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.

```

## Reference **MTHNAM Subroutine Written in PL/I**

This is a PL/I version of the MTHNAM subroutine where:

### MTHNUM

Is the double-precision number in the input argument.

### FULLMTH

Is the name of the month corresponding to the integer in MONTHNUM.

### MONTHNUM

Is the integer representing the month.

### MONTH\_TABLE

Is a 13-element array containing the names of the months. The last element contains the error message.

The subroutine 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;
```

**Reference 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

*

```

## Using a Custom Subroutine: The MTHNAM Subroutine

```

                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
```

### Reference **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];
}
```



## Calling the MTHNAM Subroutine From a Request

You can call the MTHNAM subroutine from a report request.

### *Example* Calling the MTHNAM Subroutine

The DEFINE command extracts the month portion of the pay date. The MTHNAM subroutine then converts it into the full name of the month, and stores the name in the PAY\_MONTH field. The report request prints the monthly pay of Alfred Stevens.

```
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
```

The output is:

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 request can call a subroutine coded in REXX. These subroutines, also called FUSREXX macros, provide a 4GL option to the languages supported for user-written subroutines.

REXX subroutines are supported in the VM/CMS and OS/390 environments:

- In VM/CMS, a REXX subroutine 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 REXX subroutine into memory using the EXECLOAD command. The compilation and load process reduces the CPU requirements and increases speed. Compilation is also a security tool, making private information difficult to read.

REXX subroutines are not supported in the -CMS RUN command.

- In OS/390, a REXX subroutine contains REXX source code. Compiled REXX code is not supported.

REXX subroutines 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.

Because of CPU requirements, the use of REXX subroutines in large production jobs should be monitored carefully.

For more information on REXX subroutines, see your REXX documentation.

### **Reference** Storing and Searching for a REXX Subroutine

Store a REXX subroutine as follows:

- On VM/CMS, the FILETYPE of a REXX subroutine is FUSREXX; it can be stored on any accessed disk.
- On OS/390, DDNAME FUSREXX must be allocated to a PDS. This library is searched before other OS/390 libraries.

The search order for a REXX subroutine is:

1. FUSREXX
2. Standard VM/CMS or OS/390 search order.

**Syntax**    **How to Call a REXX Subroutine**

```
DEFINE FILE filename
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
END
```

or

```
{DEFINE|COMPUTE} fieldname/{An|In} = subname(inlen1, inparm1, ...,
outlen, outparm);
```

or

```
-SET &var = subname(inlen1, inparm1, ..., outlen, outparm);
```

where:

*fieldname*

Is the field that contains the result.

*An*, *In*

Is the format of the field that contains the result.

*subname*

Is the name of the REXX subroutine.

*inlen1*, *inparm1* ...

Are the input parameters. Each parameter consists of a length and an alphanumeric parameter value. You can supply the value, the name of an alphanumeric field that contains the value, or an expression that returns the value. Up to 13 input parameter pairs are supported. Each parameter value can be up to 256 bytes long.

Dialogue Manager converts numeric arguments to floating-point double-precision format. Therefore, you can only pass alphanumeric input parameters to a REXX subroutine using -SET.

*outlen*, *outparm*

Is the output parameter pair, consisting of a length and a result. In most cases, the result should be alphanumeric, but integer results are also supported. The result can be a field or a Dialogue Manager variable that contains the value, or the format of the value 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 an integer, *outlen* must be 4 because FOCUS reserves four bytes for integer fields.

&*var*

Is the name of the Dialogue Manager variable that contains the result.

**Example**    **Returning the Day of the Week**

The REXX subroutine DOW returns the day of the week corresponding to the date an employee was hired. The routine contains 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
```

The procedure processes as follows:

1. The EDIT function converts HIRE\_DATE to alphanumeric format and stores the result in a field with the format A6.
2. The result is stored in the DAY\_OF\_THE\_WEEK field, and can be up to nine bytes long.

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 REXX subroutine is displayed below. It 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** **Passing Multiple Arguments to a REXX Subroutine**

The REXX subroutine INTEREST contains four input parameters.

```
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
```

The procedure processes as follows:

1. EDIT converts HIRE\_DATE to alphanumeric format and stores the result in AHDT.
2. EDIT converts CURR\_SAL to alphanumeric format and stores the result in ACSAL.
3. CURR\_SAL is converted to a floating-point double-precision field that includes commas, and the result is stored in DCSAL.
4. The second input field is six bytes long. Data is passed as a character variable &YMD in YYYYMMDD format.

The third input field is a character value of 6.5, which is three 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 REXX subroutine 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 a Parameter

A REXX subroutine can accept multiple tokens in a parameter. The following procedure passes employee information (PAY\_DATE and MO\_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
```

The procedure processes as follows:

1. COMPID is the concatenation of several character fields passed as the first parameter and stored in a field with the format A256. Each of the other parameters is a single argument.
2. EDIT converts PAY\_DATE to alphanumeric format.
3. EDIT converts MO\_PAY to alphanumeric format.
4. OK4RAISE executes, and the result is stored in OK4RAISE.

The output is:

EMP_ID	FIRST_NAME	LAST_NAME	DEPARTMENT
071382660	ALFRED	STEVENS	PRODUCTION

The REXX subroutine 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
```

REXX subroutines should use the REXX RETURN subroutine to return data. REXX EXIT is acceptable, but is generally used to end an EXEC, not a FUNCTION.

<pre>Correct /* Some FUSREXX function */ Arg input some rexx process ... Return data_to_FOCUS</pre>	<pre>Not as Clear /* Another FUSREXX function */ Arg input some rexx process ... Exit 0</pre>
---	---

## Formats and REXX Subroutines

A REXX subroutine requires input data to be in alphanumeric format. Most output is returned in alphanumeric format. If the format of an input argument is numeric, use the EDIT or FTOA functions to convert the argument to alphanumeric. You can then use the EDIT or ATODBL functions to convert the output back to numeric.

The output length in the subroutine call must be four. Character variables cannot be more than 256 bytes. This limit also applies to REXX subroutines. 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.

A REXX subroutine does not require any input parameters, but requires one return parameter, which must return at least one byte of data. It is possible for a REXX subroutine not to need input, such as a function that returns USERID.

A REXX subroutine does not support FOCUS date input arguments. When working with dates you can do one of the following:

- Pass an alphanumeric field with date display options and have the subroutine return a date value.

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. You must account for the number of days difference between the FOCUS base date and the REXX base date and convert the result to integer.

- Pass a date value converted to alphanumeric format. You must account for the difference in base dates for both the input and output arguments.

### Example Returning the Result in Alphanumeric Format

The NUMCNT subroutine returns the number of copies of each classic movie in alphanumeric 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
```

The procedure processes as follows:

1. The EDIT field converts COPIES to alphanumeric format, and stores the result in ACOPIES.
2. The result is stored in an 8-byte alphanumeric field 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 subroutine 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** Returning a Result in Integer Format

In the following example, the NUMDAYS subroutine finds the number of days between HIRE\_DATE and DAT\_INC and returns the result in integer format.

```
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
```

The procedure processes as follows:

1. EDIT converts HIRE\_DATE to alphanumeric format and stores the result in AHDT.
2. EDIT converts DAT\_INC to alphanumeric format and stores the result in ADI.
3. NUMDAYS finds the number of days between AHDT and ADI and stores the result in integer format.

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 subroutine 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** Passing a Date Value as an Alphanumeric Field With Date Options

In the following example, a date is used by passing an alphanumeric field with date options to the DATEREX1 subroutine. 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 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

### **Example** **Passing a Date as a Date Converted to Alphanumeric Format**

In the following example, a date is passed to the subroutine as a smart date converted to alphanumeric format. The DATEREX2 subroutine 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 DATEREX2 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 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

## Compiling FUSREXX Macros in VM/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 VM/CMS, issue the following command. Note that the file identifier must be in uppercase:

```
rexxcomp 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 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 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 issue the 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
```

*Subroutines Written in REXX*

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

## A

---

ABS function 10-2

accessing external functions 3-22  
    Developer Studio 3-27  
    OS/390 3-22  
    UNIX 3-24  
    VM/CMS 3-24

accessing functions 3-1, 3-3 to 3-4  
    OS/390 3-23

ADD function 8-5

alphanumeric strings 9-2

ARGLEN function 4-2 to 4-3

arguments 3-4

ASIS function 4-3 to 4-4, 10-3

assigning date-time values 7-9  
    COMPUTE command 7-9 to 7-10  
    DEFINE command 7-9  
    IF criteria 7-9, 7-11  
    WHERE criteria 7-9 to 7-10

ATODBL function 9-2 to 9-3

AYM function 7-50 to 7-51

AYMD function 7-51 to 7-52

## B

---

bar charts 10-3  
    scales 10-3

BAR function 10-3 to 10-5

batch allocation 3-22

bit strings 4-6 to 4-8

BITSON function 4-5 to 4-6

BITVAL function 4-6 to 4-8

branching in procedures 3-13  
    functions and 3-13, 3-15

BUSDAYS parameter 7-3 to 7-4

business days 7-3 to 7-4  
    BUSDAYS parameter 7-3

BYTVAL function 4-8 to 4-10

## C

---

calling functions 3-1 to 3-4  
    Dialogue Manager and 3-11  
    FOCUS commands and 3-10  
    from another function 3-16  
    IF criteria 3-17 to 3-18  
    in Maintain 3-4  
    RECAP command and 3-20 to 3-21  
    WHEN criteria 3-19  
    WHERE criteria 3-17 to 3-18

CHAR2INT function 5-2

character functions 2-3, 4-1  
    ARGLEN 4-2 to 4-3  
    ASIS 4-3 to 4-4  
    BITSON 4-5 to 4-6  
    BITVAL 4-6 to 4-8  
    BYTVAL 4-8 to 4-10  
    CHKFMT 4-10 to 4-12  
    CTRAN 4-14 to 4-15  
    CTRFLD 4-20  
    EDIT 4-21 to 4-22  
    GETTOK 4-22 to 4-24  
    LCWORD 4-24 to 4-25, 4-53, 5-4 to 5-5  
    LJUST 4-26 to 4-27  
    LOCASE 4-28 to 4-29  
    Maintain-specific 2-7, 5-1  
    OVLAY 4-29 to 4-31  
    PARAG 4-33 to 4-35  
    POSIT 4-35 to 4-37  
    RJUST 4-37 to 4-38

## Index

### character functions (*continued*)

- SOUNDEX 4-39 to 4-40
- SQUEEZ 4-40 to 4-41
- STRIP 4-42 to 4-44
- SUBSTR 4-44 to 4-46, 4-57
- TRIM 4-46 to 4-48
- TRIMV 4-60
- UPCASE 4-49 to 4-50

### character strings 4-1 to 4-2, 4-26

- adding 5-7
- bits 4-5 to 4-6
- centering 4-19 to 4-20
- comparing 4-39, 5-18, 5-20
- converting case 4-24, 4-28, 4-49, 4-53, 5-4, 5-6, 5-27
- determining length 5-26
- Dialogue Manager 4-3
- dividing 4-33
- extracting 5-7, 5-22
- extracting characters 4-21
- extracting substrings 4-22, 4-44, 4-46, 4-57, 5-24
- finding substrings 4-35, 5-11
- format 4-10
- justifying 4-26, 4-37, 5-6, 5-13
- measuring length 4-2, 5-5
- overlying 4-29, 5-9
- reducing spaces 4-40
- removing characters 4-42
- removing occurrences 4-46, 5-25
- right-justifying 4-37, 5-13
- substrings 5-16, 5-21
- translating characters 4-8, 4-14, 5-2 to 5-3

CHGDAT function 7-53 to 7-56

CHKFMT function 4-10 to 4-12

CHKPCK function 10-6 to 10-8

commands 11-14

- passing 11-14

compiling subroutines A-11

- OS/390 A-12

- UNIX A-12

- VM/CMS A-11

components 7-12

COMPUTE command

- assigning date-time values 7-10

- COMPUTE command 3-10

converting formats 9-1 to 9-2

creating subroutines A-1 to A-2

cross-referenced data sources 6-11

CTRAN function 4-14 to 4-15

CTRFLD function 4-19 to 4-20

custom subroutines A-13 to A-14, A-16 to A-19

## **D**

---

DA functions 7-56 to 7-57

DADMY function 7-56 to 7-57

DADYM function 7-56 to 7-57

DAMDY function 7-56 to 7-57

DAMYD function 7-56 to 7-57

data sets 11-4, 11-7

data source functions 2-10, 6-1

- FIND 6-6 to 6-8

- LAST 6-9 to 6-10

- LOOKUP 6-11 to 6-12, 6-14, 6-16 to 6-17

data source values 6-1

- decoding 6-2

- retrieving 6-9 to 6-10

- verifying 6-6 to 6-8



- data sources 6-1
  - cross-referenced 6-11, 6-17
  - retrieving values 6-9 to 6-11, 6-17
  - values 6-1
  - verifying values 6-6
- date and time functions 2-11, 7-1
  - arguments and 7-12
  - AYM 7-50 to 7-51
  - AYMD 7-51 to 7-52
  - CHGDAT 7-53 to 7-56
  - DA 7-56 to 7-57
  - DADMY 7-56 to 7-57
  - DADYM 7-56 to 7-57
  - DAMDY 7-56 to 7-57
  - DAMYD 7-56 to 7-57
  - DATEADD 7-13 to 7-16
  - DATECVT 7-16 to 7-18
  - DATEDIF 7-18 to 7-20
  - DATEMOV 7-21 to 7-24
  - DAYDM 7-56 to 7-57
  - DAYMD 7-56 to 7-58
  - Dialogue Manager and 7-5
  - DOWK 7-60 to 7-61
  - DOWKL 7-60
  - DTDMY 7-61 to 7-62
  - DTDYM 7-61 to 7-62
  - DTMDY 7-61 to 7-63
  - DTMYD 7-61 to 7-62
  - DTYDM 7-61 to 7-62
  - DTYMD 7-61 to 7-62
  - GREGDT 7-63 to 7-64
  - HADD 7-25 to 7-26
  - HCNVRT 7-27 to 7-28
  - HDATE 7-28 to 7-29
  - HDIFF 7-30 to 7-31
  - HDTTM 7-32 to 7-33
  - HGETC 7-33 to 7-34
  - HHMMSS 7-34 to 7-35
  - HINPUT 7-36 to 7-37
  - HMIDNT 7-38 to 7-39
  - HNAME 7-39 to 7-41
  - HPART 7-41 to 7-42
- date and time functions (*continued*)
  - HSETPT 7-42 to 7-44
  - HTIME 7-45 to 7-46
  - JULDAT 7-65 to 7-66
  - legacy 2-11, 2-14, 2-16, 7-1 to 7-2, 7-47
  - Maintain-specific 2-16, 8-1
  - standard 2-11, 2-16, 7-1 to 7-2
  - terminology 7-2
  - TODAY 7-46 to 7-47
  - work days 7-3
  - YM 7-67 to 7-68
  - YMD 7-59
- date formats 7-7
  - formatted-string format 7-7
  - numeric string format 7-7
  - translated-string format 7-8
- DATEADD function 7-13 to 7-16
- DATECVT function 7-16 to 7-18
- DATEDIF function 7-18 to 7-21
- DATEFNS parameter 7-47
  - GREGDT 7-64
  - JULDAT function 7-65
- DATEMOV function 7-21 to 7-24
- date-time values 7-7, 8-1
  - adding 7-13, 7-50 to 7-51, 8-5
  - assigning 7-9
  - converting 7-45
  - converting formats 7-16, 7-27 to 7-28, 7-32, 7-36, 7-53, 7-56, 7-61, 7-63, 7-65
  - determining day of week 8-11
  - determining quarter 8-8
  - elapsed time 7-67, 8-6
  - extracting 8-6 to 8-7, 8-12
  - finding day of week 7-60
  - finding difference 7-18, 7-30, 7-58
  - formats 7-7
  - incrementing 7-25
  - inserting components 7-42
  - moving dates 7-21

## Index

date-time values (*continued*)  
    retrieving 8-2 to 8-4  
    retrieving components 7-39, 7-41  
    retrieving time 7-34  
    returning dates 7-46  
    setting 8-8  
    setting time 7-38  
    storing 7-33  
    subtracting 7-13, 7-50 to 7-51, 8-10

DAY function 8-6

DAYDM function 7-56 to 7-57

DAYMD function 7-56 to 7-58

DECODE function 6-2 to 6-5

decoding functions 2-10, 6-1  
    DECODE function 6-2 to 6-5

decoding values 6-2  
    from files 6-2 to 6-5  
    in a function 6-2  
    in functions 6-3

DEFINE command 3-10  
    functions and 3-10

deleting function libraries 3-24

Dialogue Manager 3-11  
    date and time functions 7-5  
    functions and 3-11  
    LEADZERO parameter 7-5

Dialogue Manager commands 3-11  
    -IF 3-13  
    -RUN 3-15  
    -SET 3-11 to 3-12

DMOD function 10-9 to 10-11

DMY function 7-58 to 7-59

DOWK function 7-60 to 7-61

DOWKL function 7-60

DTDMY function 7-61 to 7-62

DTDYM function 7-61 to 7-62

DTMDY function 7-61 to 7-63

DTMYD function 7-61 to 7-62

DTYDM function 7-61 to 7-62

DTYMD function 7-61 to 7-62

Dynamic Language Environment (LE) 3-27

## E

---

EDIT function 4-21 to 4-22, 9-5 to 9-6

entry points A-6 to A-7

environment variables 11-3  
    assigning values 11-6  
    retrieving values 11-3

error messages 11-2

EXP function 10-11 to 10-12

EXPN function 10-13

external functions 2-2  
    accessing 3-22  
    storing 3-22

## F

---

FEXERR function 11-2

FGETENV function 11-3

FIND function 6-6 to 6-8

FINDMEM function 11-4 to 11-5

FMLINFO function 10-14 to 10-16

FMOD function 10-9 to 10-11

FOCUS commands 3-10

## format conversion functions 2-18, 9-1

- ATODBL 9-2 to 9-3
- EDIT 9-5 to 9-6
- FTOA 9-7 to 9-8
- HEXBYT 9-8 to 9-9
- ITONUM 9-11 to 9-12
- ITOPACK 9-12 to 9-13
- ITOA 9-14 to 9-15
- PCKOUT 9-16 to 9-17
- UFMT 9-18 to 9-19

## format conversions 9-5

- packed numbers 9-16
- to alphanumeric 9-7
- to characters 9-8
- to double-precision 9-11
- to hexadecimal 9-18
- to packed decimal 9-12
- to zoned format 9-14

## formats 9-2

- alphanumeric 9-6
- converting 9-2

## formatted-string format 7-7

## FPUTENV function 11-6 to 11-7

## FTOA function 9-7 to 9-8

## function arguments 3-4

- formats 3-5
- functions as 3-16
- in subroutines A-3
- length 3-6
- number 3-6
- supplying 3-4
- types 3-4

## function libraries 3-4, 3-24

- adding 3-24
- deleting 3-24

## function types 2-3

- character 2-3, 4-1, 5-1
- data source 2-10, 6-1
- date and time 2-11, 2-16, 7-1
- decoding 2-10, 6-1
- format conversion 2-18, 9-1
- Maintain-specific character 2-7
- numeric 2-19, 10-1
- system 2-21, 11-1

## functions 1-1, 2-1 to 2-2, 3-1 to 3-2

- accessing 3-1
- accessing external 3-22
- arguments 3-4
- branching in procedures 3-13, 3-18
- calling 3-1 to 3-4, 3-10, 3-16
- COMPUTE command and 3-10
- DEFINE command and 3-10
- Dialogue Manager and 3-11
- external 2-2
- FOCUS commands and 3-10
- IF command and 3-13
- internal 2-2
- invoking 3-3
- languages 1-2
- operating system commands and 3-15
- operating systems 1-2
- RUN command and 3-15
- storing external 3-22
- subroutines 2-2, A-1
- types 2-3
- VALIDATE command and 3-10
- variables and 3-11 to 3-12
- VM/CMS 3-24

**G**

## GETPDS function 11-7 to 11-9, 11-11

## GETTOK function 4-22 to 4-24

## GETUSER function 11-12

## GREGDT function 7-63 to 7-65

- DATEFNS parameter 7-64

## H

---

HADD function 7-25 to 7-26  
HCNVRT function 7-27 to 7-28  
HDATE function 7-28 to 7-29  
HDAY parameter 7-4 to 7-5  
HDIFF function 7-30 to 7-31  
HDTTM function 7-32 to 7-33  
HEXBYT function 9-8 to 9-9  
HGETC function 7-33 to 7-34  
HHMMSS function 7-34 to 7-35, 8-2, 11-13  
HINPUT function 7-36 to 7-37  
HMIDNT function 7-38 to 7-39  
HNAME function 7-39 to 7-41  
holidays 7-3 to 7-5  
    HDAY parameter 7-5  
    holiday files 7-4  
HPART function 7-41 to 7-42  
HSETPT function 7-42 to 7-44  
HTIME function 7-45 to 7-46

## I

---

-IF command 3-13  
IF criteria 3-17  
    assigning date-time values 7-11  
    functions and 3-17 to 3-18  
IMOD function 10-9 to 10-11  
Initial\_HHMMSS function 8-3  
Initial\_TODAY function 8-3  
INT function 10-16 to 10-17  
INT2CHAR function 5-3  
internal functions 2-2

invoking functions 3-2 to 3-3  
ITONUM function 9-11 to 9-12  
ITOPACK function 9-12 to 9-13  
ITOZ function 9-14 to 9-15

## J

---

JULDAT function 7-65 to 7-66  
    DATEFNS parameter 7-65  
JULIAN function 8-6 to 8-7

## L

---

languages 1-2  
LAST function 6-9 to 6-10  
LCWORD function 4-24 to 4-25, 4-53, 5-4 to 5-5  
LCWORD2 function 5-4 to 5-5  
LEADZERO parameter 7-5 to 7-6  
legacy date functions 2-11, 2-16, 7-2, 7-47, 8-5  
    DATEFNS parameter 7-47  
    DMY 7-58 to 7-59  
    legacy dates 7-48 to 7-49  
    legacy versions 7-47  
    MDY 7-58 to 7-59  
    YMD 7-58 to 7-59  
legacy dates 7-48 to 7-49  
LENGTH function 5-5  
LJUST function 4-26 to 4-27, 5-6  
load libraries 3-22  
LOCASE function 4-28 to 4-29  
LOG function 10-17 to 10-18  
LOOKUP function 6-11 to 6-12, 6-14, 6-16 to 6-17  
    extended function 6-17  
LOWER function 5-6

**M****Maintain-specific character functions 2-7, 5-1**

CHAR2INT 5-2  
 INT2CHAR 5-3  
 LWORD 5-4  
 LWORD2 5-4 to 5-5  
 LENGTH 5-5  
 LJUST 5-6  
 LOWER 5-6  
 MASK 5-7 to 5-8  
 NLSCHR 5-8  
 OVLAY 5-9 to 5-10  
 POSIT 5-11 to 5-12  
 RJUST 5-13  
 SELECTS 5-14 to 5-15  
 STRAN 5-16 to 5-17  
 STRCMP 5-18 to 5-19  
 STRICMP 5-20  
 STRTOKEN 5-22 to 5-23  
 SUBSTR 5-24 to 5-25  
 TRIM 5-25  
 TRIMLEN 5-26  
 UPCASE 5-27

**Maintain-specific date and time functions 2-16, 8-1**

ADD 8-5  
 DATEDIF 7-21  
 DATEMOV 7-24  
 DAY 8-6  
 HHMMSS 8-2  
 Initial\_HHMMSS 8-3  
 Initial\_TODAY 8-3  
 JULIAN 8-6 to 8-7  
 legacy 2-16, 8-1, 8-5  
 MONTH 8-7  
 QUARTER 8-8  
 SETMDY 8-8 to 8-9  
 standard 2-16, 8-1  
 SUB 8-10  
 TODAY 8-3 to 8-4  
 TODAY2 8-4  
 WEEKDAY 8-11  
 YEAR 8-12

Maintain-specific functions 3-4  
   MNTUWS function library 3-4  
   SELECTS 5-14  
   STRNCMP 5-21  
   STRTOKEN 5-22

MASK function 5-7 to 5-8

MAX function 10-18

MDY function 7-58 to 7-59

MIN function 10-18 to 10-19

MNTUWS function library 3-4

MONTH built-in function 8-7

MONTH function 8-7

MTHNAM subroutine A-13 to A-14, A-16 to A-19

MVSDYNAM function 11-14

**N**

naming subroutines A-3

National Language Support (NLS) 5-8

NLSCHR function 5-8

NORMSDST function 10-19, 10-21 to 10-22

NORMSINV function 10-19, 10-21 to 10-22

numbers 10-2

  absolute value 10-2

  bar charts 10-3

  calculating remainders 10-9

  generating random 10-23, 10-26

  greatest integer 10-16

  logarithms 10-17

  maximum 10-18

  minimum 10-18

  raising to a power 10-11

  scientific notation 10-13

  square root 10-28

  standard normal deviation 10-19 to 10-20

  validating packed fields 10-6

numeric functions 2-19, 10-1

- ABS 10-2
- ASIS 10-3
- BAR 10-3 to 10-5
- CHKPCK 10-6 to 10-8
- DMOD 10-9 to 10-11
- EXP 10-11 to 10-12
- EXPN 10-13
- FMLINFO 10-14 to 10-16
- FMOD 10-9 to 10-11
- IMOD 10-9 to 10-11
- INT 10-16 to 10-17
- LOG 10-17 to 10-18
- MAX 10-18
- MIN 10-18 to 10-19
- NORMSDST 10-19, 10-21 to 10-22
- NORMSINV 10-19, 10-21 to 10-22
- PRDNOR 10-23 to 10-25
- PRDUNI 10-23 to 10-24
- RDNORM 10-26 to 10-27
- RDUNIF 10-26 to 10-27
- SQRT 10-28

numeric string format 7-7

numeric values 10-1

**O**

---

operating system commands 3-15

- functions 3-15

operating systems 1-2

OS/390 3-22

- accessing functions 3-22
- compiling subroutines A-12
- storing functions 3-22 to 3-23
- storing subroutines A-12

OVLAY function 4-29 to 4-31, 5-9 to 5-10

**P**

---

PARAG function 4-33 to 4-35

PCKOUT function 9-16 to 9-17

POSIT function 4-35 to 4-37, 5-11 to 5-12

PRDNOR function 10-23 to 10-25

PRDUNI function 10-23 to 10-24

programming subroutines A-6

- arguments A-8 to A-10

**Q**

---

QUARTER function 8-8

**R**

---

RDNORM function 10-26 to 10-27

RDUNIF function 10-26 to 10-27

RECAP command 3-20 to 3-21

REXX subroutines A-20 to A-29

- formats A-25

RJUST function 4-37 to 4-38, 5-13

-RUN command 3-15

**S**

---

SELECTS function 5-14 to 5-15

-SET command 3-11 to 3-12

SET parameters 7-3

- BUSDAYS 7-3 to 7-4

- DATEFNS 7-47

- HDAY 7-4 to 7-5

- LEADZERO 7-5 to 7-6

SETMDY function 8-8 to 8-9

SOUNDEX function 4-39 to 4-40

SQRT function 10-28

SQUEEZ function 4-40 to 4-41

standard date and time functions 2-11, 2-16, 7-2

standard normal deviation 10-19 to 10-20

storing external functions 3-22
 

- OS/390 3-22 to 3-23
- UNIX 3-24
- VM/CMS 3-24

storing subroutines A-11 to A-12
 

- libraries A-11
- OS/390 A-12
- text files A-11
- UNIX A-12
- VM/CMS A-11

STRAN function 5-16 to 5-17

STRCMP function 5-18 to 5-19

STRICMP function 5-20

STRIP function 4-42 to 4-44

STRNCMP function 5-21

STRTOKEN function 5-22 to 5-23

SUB function 8-10

subroutines 2-2, A-1
 

- compiling A-11
- creating A-1
- custom A-13 to A-14, A-16 to A-19
- entry points A-6 to A-7
- MTHNAM A-13 to A-14, A-16 to A-19
- naming A-3
- programming A-6
- REXX A-20 to A-29
- storing A-11
- testing A-12
- writing A-2

SUBSTR function 4-44 to 4-46, 4-57, 5-24 to 5-25

substrings 4-21
 

- comparing 5-21
- extracting 4-21 to 4-22, 4-44, 4-46, 4-57, 5-24
- finding 4-35, 5-11
- overlying character strings 4-29, 5-9
- substituting 5-16

system functions 2-21, 11-1
 

- FEXERR 11-2
- FGETENV 11-3
- FINDMEM 11-4 to 11-5
- FPUTENV 11-6 to 11-7
- GETPDS 11-7 to 11-9, 11-11
- GETUSER 11-12
- HHMMSS 11-13
- MVSDYNAM 11-14
- TODAY 11-17

## T

---

testing subroutines A-12

time formats 7-7 to 7-8

TODAY function 7-46 to 7-47, 8-3 to 8-4, 11-17

TODAY2 function 8-4

translated-string format 7-8

TRIM function 4-46 to 4-48, 5-25

TRIMLEN function 5-26

TRIMV function 4-60

## U

---

UFMT function 9-18 to 9-19

UNIX 3-24
 

- accessing functions 3-24
- compiling subroutines A-12
- storing functions 3-24
- storing subroutines A-12

UPCASE function 4-49 to 4-50, 5-27

user IDs 11-12

## **V**

---

VALIDATE command 3-10

values 6-2

- decoding 6-2

- verifying 6-6 to 6-8

VM/CMS 3-24, A-11

- accessing external functions 3-24

- compiling subroutines A-11

- storing external functions 3-24

- storing subroutines A-11

## **W**

---

WEEKDAY function 8-11

WHEN criteria 3-19

WHERE criteria 3-17

- assigning date-time values 7-10

- functions and 3-17 to 3-18

work days 7-3

- business days 7-3

- holidays 7-3 to 7-4

writing subroutines A-2

- creating arguments A-3

- entry points A-6 to A-7

- languages A-4

- naming subroutines A-3

- programming A-6, A-8 to A-10

## **Y**

---

YEAR function 8-12

YM function 7-67 to 7-68

YMD function 7-58 to 7-59



---

---

## 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. You can contact us through the following methods:

**Mail:** Documentation Services - Customer Support  
Information Builders, Inc.  
Two Penn Plaza  
New York, NY 10121-2898

**Fax:** (212) 967-0460

**E-mail:** books\_info@ibi.com

**Web form:** <http://www.informationbuilders.com/bookstore/derf.html>

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

Telephone: \_\_\_\_\_ Date: \_\_\_\_\_

E-mail: \_\_\_\_\_

Comments:

---

---

## **Reader Comments**