Information Builders

FOCUS for Mainframe

Using Functions

Version 7.6

DN1001140.0308

Contents

1. Introducing Functions 17
Using Functions 18
Types of Functions 19
Character Functions 20
Maintain-specific Character Functions 22
Data Source and Decoding Functions 24
Date and Time Functions 24
Maintain-specific Date and Time Functions 27
Format Conversion Functions 28
Numeric Functions 29
System Functions 30
Input/Output Functions 31
Character Chart for ASCII and EBCDIC 32

2. Accessing and Calling a Function 39
Calling a Function 40
Supplying an Argument in a Function 42
Argument Types 42
Argument Formats 43
Argument Length 44
Number and Order of Arguments 44
Verifying Function Parameters 45
Calling a Function From a DEFINE, COMPUTE, or VALIDATE Command 48
Calling a Function From a Dialogue Manager Command 50
Assigning the Result of a Function to a Variable 50
Branching Based on a Function’s Result 52
Calling a Function From an Operating System RUN Command 54
Calling a Function From Another Function 55
Calling a Function in WHERE or IF Criteria 56
Using a Calculation or Compound IF Command 57
Calling a Function in WHEN Criteria 57
Calling a Function From a RECAP Command 59
Storing and Accessing an External Function 61
Storing and Accessing a Function on z/OS 61
Storing and Accessing a Function on CMS 63

3. Character Functions 67
ARGLEN: Measuring the Length of a Character String 68
ASIS: Distinguishing Between a Space and a Zero 69
BITSON: Determining If a Bit Is On or Off 70
BITVAL: Evaluating a Bit String as a Binary Integer 72
BYTVAL: Translating a Character to a Decimal Value 74
CHKFMT: Checking the Format of a Character String 76
CTRAN: Translating One Character to Another 79
CTRFLD: Centering a Character String 86
DCTRAN: Translating a Single-Byte or Double-Byte Character to Another 88
DSTRIP: Removing a Single-Byte or Double-Byte Character From a String 89
EDIT: Extracting or Adding Characters 90
GETTOK: Extracting a Substring (Token) 92
JPTRANS: Converting Japanese-Specific Characters 94
LCWORD: Converting a Character String to Mixed-Case 99
LJUST: Left-Justifying a Character String 100
LOCASE: Converting Text to Lowercase 102
OVRLAY: Overlaying a Character String 103
PARAG: Dividing Text Into Smaller Lines 106
POSIT: Finding the Beginning of a Substring 108
RJUST: Right-Justifying a Character String 110
SOUNDEX: Comparing Character Strings Phonetically 112
SPELLNM: Spelling Out a Dollar Amount 114
SQUEEZ: Reducing Multiple Spaces to a Single Space 115
STRIP: Removing a Character From a String 117
STRREP: Replacing Character Strings 120
SUBSTR: Extracting a Substring 122
TRIM: Removing Leading and Trailing Occurrences 124
UPCASE: Converting Text to Uppercase 126
Character Functions for AnV Fields 129
LENV: Returning the Length of an Alphanumeric Field 130
LOCASV: Creating a Variable Length Lowercase String 131
POSITV: Finding the Beginning of a Variable Length Substring 133
SUBSTV: Extracting a Variable Length Substring 135
TRIMV: Removing Characters From a String 137
UPCASV: Creating a Variable Length Uppercase String 139

4. Maintain-specific Character Functions 141
CHAR2INT: Translating a Character Into an Integer Value 142
INT2CHAR: Translating an Integer Value Into a Character 143
LCWORD and LCWORD2: Converting a Character String to Mixed-Case 144
LENGTH: Determining the Length of a Character String 145
LJUST: Left-Justifying a Character String (Maintain) 146
LOWER: Converting a Character String to Lowercase 147
MASK: Extracting or Adding Characters 147
NLSCHR: Converting Characters From the Native English Code Page 149
OVRLAY: Overlaying a Character String (Maintain) 150
POSIT: Finding the Beginning of a Substring (Maintain) 152
RJUST: Right-Justifying a Character String (Maintain) 153
SELECTS: Decoding a Value From a Stack 154
STRAN: Substituting One Substring for Another 156
STRCMP: Comparing Character Strings 158
STRICMP: Comparing Character Strings and Ignoring Case .160
STRNCMP: Comparing Character Substrings 160
STRTOKEN: Extracting a Substring Based on Delimiters 161
SUBSTR: Extracting a Substring (Maintain) 162
TRIM: Removing Trailing Occurrences (Maintain) 164
TRIMLEN: Determining the Length of a String Excluding Trailing Spaces 164
UPCASE: Converting Text to Uppercase (Maintain) 165

5. Data Source and Decoding Functions 167
DB_LOOKUP: Retrieving a Value From a Lookup Data Source 168
DECODE: Decoding Values 172
FIND: Verifying the Existence of a Value in a Data Source 177
LAST: Retrieving the Preceding Value 182
LOOKUP: Retrieving a Value From a Cross-referenced Data Source 184
Using the Extended LOOKUP Function 190

6. Date and Time Functions 191
Date and Time Function Terminology 192
Using Standard Date and Time Functions 193
Specifying Work Days 194
Enabling Leading Zeros For Date and Time Functions in Dialogue Manager 197
Using Date and Time Formats 199
Assigning Date-Time Values 203
DATEADD: Adding or Subtracting a Date Unit to or From a Date 210
DATECVT: Converting the Format of a Date 214
DATEDIF: Finding the Difference Between Two Dates .216
DATEMOV: Moving a Date to a Significant Point 219
DATETRAN: Formatting Dates in International Formats 223
HADD: Incrementing a Date-Time Value 237
HCNVRT: Converting a Date-Time Value to Alphanumeric Format 239
HDATE: Converting the Date Portion of a Date-Time Value to a Date Format 241
HDIFF: Finding the Number of Units Between Two Date-Time Values 242
HDTTM: Converting a Date Value to a Date-Time Value 244
HEXTR: Extracting Components of a Date-Time Value and Setting Remaining Components to Zero 246
HGETC: Storing the Current Date and Time in a Date-Time Field 248
HHMMSS: Retrieving the Current Time 249
HINPUT: Converting an Alphanumeric String to a Date-Time Value 251
HMASK: Extracting Components of a Date-Time Field and Preserving Remaining Components 252
HMIDNT: Setting the Time Portion of a Date-Time Value to Midnight 255
HNAME: Retrieving a Date-Time Component in Alphanumeric Format 256
HPART: Retrieving a Date-Time Component in Numeric Format 259
HSETPT: Inserting a Component Into a Date-Time Value 261
HTIME: Converting the Time Portion of a Date-Time Value to a Number 263
TIMETOTS: Converting a Time to a Timestamp 265
TODAY: Returning the Current Date 266

Using Legacy Date Functions 268
Using Old Versions of Legacy Date Functions 268
Using Dates With Two- and Four-Digit Years 269
AYM: Adding or Subtracting Months to or From Dates 271
AYMD: Adding or Subtracting Days to or From a Date 273
CHGDAT: Changing How a Date String Displays 274
DA Functions: Converting a Date to an Integer 278
DMY, MDY, YMD: Calculating the Difference Between Two Dates 280
DOWK and DOWKL: Finding the Day of the Week 281
DT Functions: Converting an Integer to a Date 282
GREGDT: Converting From Julian to Gregorian Format 285
JULDAT: Converting From Gregorian to Julian Format 287
YM: Calculating Elapsed Months 289

7. Maintain-specific Date and Time Functions 291
Maintain-specific Standard Date and Time Functions 292
HHMMSS: Retrieving the Current Time (Maintain) 292
Initial_HHMMSS: Returning the Time the Application Was Started 293
Initial_TODAY: Returning the Date the Application Was Started 294
TODAY: Retrieving the Current Date (Maintain) 294
TODAY2: Returning the Current Date 295
ADD: Adding Days to a Date 296
DAY: Extracting the Day of the Month From a Date 297
JULIAN: Determining How Many Days Have Elapsed in the Year 298
MONTH: Extracting the Month From a Date 299
QUARTER: Determining the Quarter 300
SETMDY: Setting the Value to a Date 301
SUB: Subtracting a Value From a Date 302
WEEKDAY: Determining the Day of the Week for a Date 303
YEAR: Extracting the Year From a Date 304

8. Format Conversion Functions 305
ATODBL: Converting an Alphanumeric String to Double-Precision Format 306
EDIT: Converting the Format of a Field 309
FTOA: Converting a Number to Alphanumeric Format 311
HEXBYT: Converting a Decimal Integer to a Character 312
ITONUM: Converting a Large Binary Integer to Double-Precision Format 316
ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format 317
ITOZ: Converting a Number to Zoned Format 319
PCKOUT: Writing a Packed Number of Variable Length 321
PTOA: Converting a Packed-Decimal Number to Alphanumeric Format 322
UFMT: Converting an Alphanumeric String to Hexadecimal .325

9. Numeric Functions 327
ABS: Calculating Absolute Value 328
ASIS: Distinguishing Between a Blank and a Zero 329
BAR: Producing a Bar Chart 329
CHKPCK: Validating a Packed Field 332
DMOD, FMOD, and IMOD: Calculating the Remainder From a Division 335
EXP: Raising “e” to the Nth Power 336
FMLINFO: Returning FOR Values 337
FMLLIST: Returning an FML Tag List 340
FMLFOR: Retrieving FML Tag Values 341
FMLCAP: Retrieving FML Hierarchy Captions 342
INT: Finding the Greatest Integer 344
LOG: Calculating the Natural Logarithm 345
MAX and MIN: Finding the Maximum or Minimum Value 346
MIRR: Calculating the Modified Internal Return Rate 347
NORMSDST and NORMSINV: Calculating Cumulative Normal Distribution 351
PRDNOR and PRDUNI: Generating Reproducible Random Numbers 355
RDNORM and RDUNIF: Generating Random Numbers 358
SQRT: Calculating the Square Root 360
XIRR: Calculating the Modified Internal Return Rate (Periodic or Non-Periodic) 361

10. System Functions 365
FEXERR: Retrieving an Error Message 366
FINDMEM: Finding a Member of a Partitioned Data Set 367
GETPDS: Determining If a Member of a Partitioned Data Set Exists 369
GETUSER: Retrieving a User ID 374
HHMMSS: Retrieving the Current Time 375
MVSDYNAM: Passing a DYNAM Command to the Command Processor 376
SLEEP: Suspending Execution for a Given Number of Seconds 379
TODAY: Returning the Current Date 380

11. Input/Output Functions 381
PUTDDREC and CLSDDREC: Opening, Closing, and Writing Files 382

A. Creating a Subroutine 387
Writing a Subroutine 388
Naming a Subroutine 389
Creating Arguments 389
MSO Considerations 390
z/VM Considerations With LE 391
Language Considerations 392
Programming a Subroutine 395
Compiling and Storing a Subroutine 401
Compiling and Storing a Subroutine on CMS 401
Compiling and Storing a Subroutine on z/OS 402
Testing the Subroutine 403
Using a Custom Subroutine: The MTHNAM Subroutine .403
Writing the MTHNAM Subroutine 404
Calling the MTHNAM Subroutine From a Request .423
Subroutines Written in REXX 424
Formats and REXX Subroutines 430
Compiling FUSREXX Macros in CMS 435

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 z/VM and z/OS (MVS, OS/390) environments.

The users' documentation for FOCUS Version 7.6 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	Introducing Functions	Offers an introduction to functions and explains the different types of functions available.
2	Accessing and Calling a Function	Describes considerations for supplying arguments in a function, explains how to use a function in a command, and how to access externally-stored functions.
3	Character Functions	Describes character functions, which enable you to manipulate alphanumeric fields and character strings.
4	Maintain-specific Character Functions	Describes Maintain-specific character functions which manipulate alphanumeric fields and character strings.
5	Data Source and Decoding Functions	Describes data source functions, which enable you to search for or retrieve data source records or values.
6	Date and Time Functions	Describes date and time functions, which enable you to manipulate date and time values.
7	Maintain-specific Date and Time Functions	Describes Maintain-specific date and time functions which manipulate date and time values.
8	Format Conversion Functions	Describes format conversion functions, which convert fields from one format to another.
9	Numeric Functions	Describes numeric functions, which enable you to perform calculations on numeric constants and fields.
10	System Functions	Describes system functions, which enable you to make calls to the operating system to get information about the operating environment or to use a system service.
11	Input/Output Functions	Describes input/output functions, which use your operating environment's input/output routines to open, close, and write records to sequential files
A	Creating a Subroutine	Describes how to create and store site-specific functions.

Documentation Conventions

This manual describes the following conventions apply throughout this manual:

Convention	Description
`THIS TYPEFACE` or `this typeface`	Denotes syntax that you must enter exactly as shown.
`this typeface`	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
`underscore`	Indicates a default setting.
this typeface	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.
this typeface	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 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.

1 Introducing Functions

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

Topics:

❑ Using Functions

❑ Types of Functions

❑ Character Chart for ASCII and EBCDIC

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 2, *Accessing and Calling a Function*.

Types of Functions

In this section:

Character Functions

Maintain-specific Character Functions

Data Source and Decoding Functions

Date and Time Functions

Maintain-specific Date and Time Functions

Format Conversion Functions

Numeric Functions

System Functions

Input/Output 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 20.
- ❑ **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 22.
- ❑ **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 24.
- ❑ **Date and time functions.** Manipulate dates and times. For details, see *Date and Time Functions* on page 24.
- ❑ **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 27.
- ❑ **Format conversion functions.** Convert fields from one format to another. For details, see *Format Conversion Functions* on page 28.
- ❑ **Numeric functions.** Perform calculations on numeric constants and fields. For details, see *Numeric Functions* on page 29.
- ❑ **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 30.

- **Input/Output functions.** Call the operating system's input/output routines to open, close, and write records to flat files. For details, see *Input/Output Functions* on page 31.

Character Functions

The following functions manipulate alphanumeric fields or character strings. For details, see Chapter 3, *Character Functions*.

ARGLEN

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

ASIS

Distinguishes between a blank and a zero in Dialogue Manager.

BITSON

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

BITVAL

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

BYTVAL

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

CHKFMT

Checks a character string for incorrect characters or character types.

CTRAN

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

CTRFLD

Centers a character string within a field.

DCTRAN

Translates a single-byte or double-byte character to another.

DSTRIP

Removes a single-byte or double-byte character from a string.

EDIT

Extracts characters from or adds characters to a character string.

GETTOK

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

JPTRANS

Converts Japanese-specific characters.

LCWORD

Converts the letters in a character string to mixed case.

LJUST

Left-justifies a character string within a field.

LOCASE

Converts alphanumeric text to lowercase.

OVRLAY

Overlays a base character string with a substring.

PARAG

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

POSIT

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

RJUST

Right-justifies a character string.

SOUNDEX

Searches for a character string phonetically without regard to spelling.

SPELLNM

Takes an alphanumeric string or a numeric value with two decimal places and spells it out with dollars and cents.

SQUEEZ

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

STRIP

Removes all occurrences of a specific character from a string.

STRREP

Replaces all occurrences of a specific character string.

SUBSTR

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

TRIM

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

UPCASE

Converts a character string to uppercase.

Character Functions for AnV Fields

LENV

Returns the actual length of an A*n*V field or the size of an A*n* field.

LOCASV

Converts alphanumeric text to lowercase in an A*n*V field.

POSITV

Finds the starting position of a substring in an A*n*V field.

SUBSTV

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

TRIMV

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

UPCASV

Converts a character string to uppercase in an A*n*V field.

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 4, *Maintain-specific Character Functions*.

CHAR2INT

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

INT2CHAR

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

LCWORD and LCWORD2

Converts the letters in a character string to mixed case.

LENGTH

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

LJUST

Left-justifies a character string within a field.

LOWER

Converts a character string to lowercase.

MASK

Extracts characters from or adds characters to a character string.

NLSCHR

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

OVRLAY

Overlays a base character string with a substring.

POSIT

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

RJUST

Right-justifies a character string.

SELECTS

Decodes a value from a stack.

STRAN

Substitutes a substring for another substring in a character string.

STRCMP

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

STRICMP

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

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.

SUBSTR

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

TRIM

Removes trailing occurrences of a pattern within a character string.

TRIMLEN

Determines the length of a character string excluding trailing spaces.

UPCASE

Converts a character string to uppercase.

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 5, *Data Source and Decoding Functions*.

DECODE

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

FIND

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

LAST

Retrieves the preceding value for a field.

LOOKUP

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

DBLOOKUP

Retrieves a data value from a lookup data source.

Date and Time Functions

In this section:

Standard Date and Time Functions

Legacy Date Functions

The following functions manipulate dates and times. For details see Chapter 6, *Date and Time Functions*.

Standard Date and Time Functions

DATEADD

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

DATECVT

Converts date formats.

DATEDIF

Returns the difference between two dates in units.

DATEMOV

Moves a date to a significant point on the calendar.

DATETRAN

Formats dates in international formats.

HADD

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

HCNVRT

Converts a date-time field to a character string.

HDATE

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

HDIFF

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

HDTTM

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

HEXTR

Extracts components from a date-time value and moves them to a target date-time field with all other components set to zero.

HGETC

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

HMASK

Extracts components from a date-time value and moves them to a target date-time field with all other components of the target field preserved.

HHMMSS

Retrieves the current time from the system.

HINPUT

Converts an alphanumeric string to a date-time value.

HMIDNT

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

HNAME

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

HPART

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

HSETPT

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

HTIME

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

TIMETOTS

Converts a time to a timestamp.

TODAY

Retrieves the current date from the system.

Legacy Date Functions

AYM

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

AYMD

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

CHGDAT

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

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.

DMY, MDY, and YMD

Calculate the difference between two dates.

DOWK and DOWKL

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

DT

Converts 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.

GREGDT

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

JULDAT

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

YM

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

Maintain-specific Date and Time Functions

In this section:

Maintain-specific Standard Date and Time Functions

Maintain-specific Legacy Date Functions

The following functions manipulate dates and times. They are available only in the Maintain language. For details, see Chapter 7, *Maintain-specific Date and Time Functions*.

Maintain-specific Standard Date and Time Functions

HHMMSS

Retrieves the current time from the system.

Initial_HHMMSS

Retrieves the time that the Maintain module was started.

Initial_TODAY

Retrieves the date that the Maintain module was started.

TODAY

Retrieves the current date from the system.

TODAY2

Retrieves the current date from the system.

Maintain-specific Legacy Date Functions

ADD

Adds a given number of days to a date.

DAY

Extracts the day of the month from a date.

JULIAN

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

MONTH

Extracts the month from a date.

QUARTER

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

SETMDY

Sets a value to a date.

SUB

Subtracts a given number of days from a date.

WEEKDAY

Determines the day of the week for a date.

YEAR

Extracts the year from a date.

Format Conversion Functions

The following functions convert fields from one format to another. For details, see Chapter 8, *Format Conversion Functions*.

ATODBL

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

EDIT

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

FTOA

Converts a number in a numeric format to alphanumeric format.

HEXBYT

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

ITONUM

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

ITOPACK

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

ITOZ

Converts a number in numeric format to zoned format.

PCKOUT

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

PTOA

Converts a packed decimal number from numeric format to alphanumeric format.

UFMT

Converts characters in alphanumeric field values to hexadecimal representation.

Numeric Functions

The following functions perform calculations on numeric constants or fields. For details, see Chapter 9, *Numeric Functions*.

ABS

Returns the absolute value of a number.

ASIS

Distinguishes between a blank and a zero in Dialogue Manager.

BAR

Produces a horizontal bar chart.

CHKPCK

Validates the data in a field described as packed format.

DMOD, FMOD, and IMOD

Calculate the remainder from a division.

EXP

Raises the number “e” to a specified power.

EXPN

Is an operator that evaluates a number expressed in scientific notation. For information, see *Using Expressions* in the *Creating Reports* manual.

FMLINFO

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

FMLLIST

Returns a string containing the complete tag list for each row in an FML request.

FMLFOR

Retrieves the tag value associated with each row in an FML request.

FMLCAP

Returns the caption value for each row in an FML hierarchy request.

INT

Returns the integer component of a number.

LOG

Returns the natural logarithm of a number.

MAX and MIN

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

MIRR

Calculates the modified internal rate of return for a series of periodic cash flows.

NORMSDST and NORMSINV

Perform calculations on a standard normal distribution curve.

PRDNOR and PRDUNI

Generate reproducible random numbers.

RDNORM, and RDUNIF

Generate random numbers.

SQRT

Calculates the square root of a number.

XIRR

Calculates the internal rate of return for a series of cash flows that can be periodic or non-periodic.

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 10, *System Functions*.

FEXERR

Retrieves an Information Builders error message.

FINDMEM

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

Available Operating Systems: z/OS

GETPDS

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

Available Operating Systems: z/OS

GETUSER

Retrieves the ID of the connected user.

HHMMSS

Retrieves the current time from the system.

MVSDYNAM

Transfers a FOCUS DYNAM command to the DYNAM command processor.

Available Operating Systems: z/OS

SLEEP

Suspends execution for a specified number of seconds.

TODAY

Retrieves the current date from the system.

Input/Output Functions

The following functions call the operating system's input/output routines to open, close, and write records to flat files. For details, see Chapter 11, *Input/Output Functions*.

PUTDDREC

Writes a character string as a record in a flat file. Opens the file if it is closed.

CLSDDREC

Closes a file and frees the memory used to store information about open files.

Character Chart for ASCII and EBCDIC

This chart shows the primary printable characters in the ASCII and EBCDIC character sets and their decimal equivalents. Extended ASCII codes (above 127) are not included.

Decimal	ASCII		EBCDIC	
33	!	exclamation point		
34	"	quotation mark		
35	#	number sign		
36	$	dollar sign		
37	%	percent		
38	&	ampersand		
39	'	apostrophe		
40	(	left parenthesis		
41	)	right parenthesis		
42	*	asterisk		
43	+	plus sign		
44	,	comma		
45	-	hyphen		
46	.	period		
47	/	slash		
48	0	0		
49	1	1		
50	2	2		
51	3	3		
52	4	4		
53	5	5		
54	6	6		
55	7	7		

Decimal	ASCII		EBCDIC	
56	8	8		
57	9	9		
58	:	colon		
59	;	semicolon		
60	<	less-than sign		
61	=	equal sign		
62	>	greater-than sign		
63	?	question mark		
64	@	at sign		
65	A	A		
66	B	B		
67	C	C		
68	D	D		
69	E	E		
70	F	F		
71	G	G		
72	H	H		
73	I	I		
74	J	J	¢	cent sign
75	K	K	.	period
76	L	L	<	less-than sign
77	M	M	(	left parenthesis
78	N	N	+	plus sign
79	O	O	|	logical or
80	P	P	&	ampersand

Decimal	ASCII		EBCDIC	
81	Q	Q		
82	R	R		
83	S	S		
84	T	T		
85	U	U		
86	V	V		
87	W	W		
88	X	X		
89	Y	Y		
90	Z	Z	!	exclamation point
91	[	opening bracket	$	dollar sign
92	\	back slant	*	asterisk
93	]	closing bracket	)	right parenthesis
94	^	caret	;	semicolon
95	_	underscore	¬	logical not
96	`	grave accent	-	hyphen
97	a	a	/	slash
98	b	b		
99	c	c		
100	d	d		
101	e	e		
102	f	f		
103	g	g		
104	h	h		
105	i	i		

Decimal	ASCII		EBCDIC	
106	j	j		
107	k	k	,	comma
108	l	l	%	percent
109	m	m	_	underscore
110	n	n	>	greater-than sign
111	o	o	?	question mark
112	p	p		
113	q	q		
114	r	r		
115	s	s		
116	t	t		
117	u	u		
118	v	v		
119	w	w		
120	x	x		
121	y	y		
122	z	z	:	colon
123	{	opening brace	#	number sign
124	|	vertical line	@	at sign
125	}	closing brace	'	apostrophe
126	~	tilde	=	equal sign
127			"	quotation mark
129			a	a
130			b	b
131			c	c

Decimal	ASCII		EBCDIC	
132			d	d
133			e	e
134			f	f
135			g	g
136			h	h
137			i	i
145			j	j
146			k	k
147			l	l
148			m	m
149			n	n
150			o	o
151			p	p
152			q	q
153			r	r
162			s	s
163			t	t
164			u	u
165			v	v
166			w	w
167			x	x
168			y	y
169			z	z
185			`	grave accent
193			A	A

Decimal	ASCII		EBCDIC	
194			B	B
195			C	C
196			D	D
197			E	E
198			F	F
199			G	G
200			H	H
201			I	I
209			J	J
210			K	K
211			L	L
212			M	M
213			N	N
214			O	O
215			P	P
216			Q	Q
217			R	R
226			S	S
227			T	T
228			U	U
229			V	V
230			W	W
231			X	X
232			Y	Y
233			Z	Z

Decimal	ASCII		EBCDIC	
240			0	0
241			1	1
242			2	2
243			3	3
244			4	4
245			5	5
246			6	6
247			7	7
248			8	8
249			9	9

2 Accessing and Calling a Function

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.

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

Calling a Function

How to:

Call a Function

Store Output in a Field

Access the Maintain MNTUWS Function Library

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 more information on external functions, see *Types of Functions* in Chapter 1, *Introducing Functions*.

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 more information on retrieving the MNTUWS library, see *How to Access the Maintain MNTUWS Function Library* on page 42.

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 42.

`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

In this section:

Argument Types

Argument Formats

Argument Length

Number and Order of Arguments

Verifying Function Parameters

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 A*n*V 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

How to:

Enable Parameter Verification

Control Function Parameter Verification

Example:

Verifying Parameters With Correctable Errors

Verifying Parameters With Uncorrectable Errors

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 is used.

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

where:

<u>SYSTEM</u>

Gives precedence to functions supplied by Information Builders. SYSTEM is the default value. 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.

Note: When USERFNS is set to LOCAL, DT functions only display a six-digit date.

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.

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.

`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: 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

How to:

Call a Function From a COMPUTE, DEFINE, 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

In this section:

Assigning the Result of a Function to a Variable

Branching Based on a Function's Result

Calling a Function From an Operating System RUN 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 more information, see *Assigning the Result of a Function to a Variable* on page 50.
- ❑ From an -IF command. For more information, see *Calling a Function in WHERE or IF Criteria* on page 56.
- ❑ From an operating system -RUN command. For more information, see *Calling a Function From an Operating System RUN Command* on page 54.

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

How to:

Assign the Result of a Function to a Variable

Example:

Calling a Function From a -SET Command

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

How to:

Branch Based on a Function’s Result

Example:

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

How to:

Call a Function From an Operating System -RUN Command

Example:

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 example 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

How to:

Call a Function From Another Function

Example:

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 example, 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

In this section:

Using a Calculation or Compound IF Command

How to:

Call a Function in WHERE Criteria

Call a Function in IF Criteria

Example:

Calling a Function in WHERE 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

```
IF 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

How to:

Call a Function in WHEN Criteria

Example:

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

How to:

Call a Function From a RECAP Command

Example:

Calling a Function in a RECAP Command

You can call a function from 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 appears 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

In this section:

Storing and Accessing a Function on z/OS

Storing and Accessing a Function on CMS

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 more information on creating a subroutine, see Appendix A, *Creating a Subroutine*.

Storing and Accessing a Function on z/OS

How to:

Allocate a Load Library in z/OS Batch

Allocate a Load Library in TSO

Allocate a Load Library

Example:

Allocating the Load Library BIGLIB.LOAD in z/OS Batch (JCL)

Allocating the FUSELIB.LOAD Load Library

Concatenating a Load Library to USERLIB in TSO

Concatenating a Load Library to STEPLIB in Batch (JCL)

On z/OS, 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 must allocate those libraries.

Procedure: How to Allocate a Load Library in z/OS 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 z/OS 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 STEPLIB 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 CMS

In this section:

Searching a Function Library in CMS

Adding or Deleting a Function Library

How to:

Search a Function Library

List Function Libraries Specified by the GLOBAL Command

Add or Delete a Function Library

Reference:

Search Sequence on CMS

Example:

Searching a Function Library

Searching Multiple Function Libraries

On 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 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 CMS

The standard 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 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, 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 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.

3 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 4, *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
- DCTRAN: Translating a Single-Byte or Double-Byte Character to Another
- DSTRIP: Removing a Single-Byte or Double-Byte Character From a String
- EDIT: Extracting or Adding Characters
- GETTOK: Extracting a Substring (Token)
- JPTRANS: Converting Japanese-Specific Characters
- LCWORD: Converting a Character String to Mixed-Case
- LJUST: Left-Justifying a Character String
- LOCASE: Converting Text to Lowercase
- OVRLAY: 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
- SPELLNM: Spelling Out a Dollar AmountSQUEEZ: Reducing Multiple Spaces to a Single Space
- STRIP: Removing a Character From a String
- STRREP: Replacing Character Strings
- 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

How to:

Measure the Length of a Character String

Example:

Measuring the Length of a Character String

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

ARGLEN 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 = ARGLEN(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

How to:

Distinguish Between a Space and a Zero

Example:

Distinguishing Between a Space and a Zero

The ASIS function prevents the source character string from being converted into numeric format when it is used in Dialog Manager and consists of numeric characters and blanks.

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

How to:

Determine If a Bit Is On or Off

Example:

Evaluating a Bit in a Field

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 to be evaluated, 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

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: 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

How to:

Evaluate a Bit String

Example:

Evaluating a Bit String

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 to be evaluated, 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.

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

How to:

Translate a Character

Example:

Translating the First Character of a Field

Returning the EBCDIC Value With Dialogue Manager

The BYTVAL function translates a character to the ASCII, EBCDIC, or Unicode 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 is:

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

THE EQUIVALENT VALUE IS 33
```

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 z/OS 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

How to:

Check the Format of a Character String

Example:

Checking the Format of a Field

Checking the Format of a Field With MODIFY on z/OS

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, by 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 z/OS

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. he 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

How to:

Translate One Character to Another

Example:

Translating Spaces to Underscores

Inserting Accented Letter E's With MODIFY)

Inserting Commas With MODIFY

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 must know the decimal equivalent of the characters in internal machine representation. For printable EBCDIC or ASCII characters and their decimal equivalents see the *Character Chart for ASCII and EBCDIC* in Chapter 1, *Introducing Functions*. Note that the coding chart for conversion is platform dependent, hence your platform and configuration option determines whether ASCII, EBCDIC, or Unicode coding is used.

In Unicode configurations, this function uses values in the range:

- 0 to 255 for 1-byte characters.
- 256 to 65535 for 2-byte characters.
- 65536 to 16777215 for 3-byte characters.
- 16777216 to 4294967295 for 4-byte characters (primarily for EBCDIC).

Syntax: How to Translate One Character to Another

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

where:

charlen

Integer

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

string

Alphanumeric

Is the character string to be translated 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

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

How to:

Center a Character String

Example:

Centering a Field

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 number of characters in *string* and *outfield*, 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/A12 = CTRFLD(LAST_NAME, 12, 'A12');
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

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

DCTRAN: Translating a Single-Byte or Double-Byte Character to Another

How to:

Translate a Single-Byte or Double-Byte Character to Another

Example:

Using DCTRAN to Translate Double-Byte Characters

The DCTRAN function translates a single-byte or double-byte character within a character string to another character based on its decimal value. To use DCTRAN, you need to know the decimal equivalent of the characters in internal machine representation.

Syntax: How to Translate a Single-Byte or Double-Byte Character to Another

```
DCTRAN(length, source_string, inhexchar, outhexchar, outfield)
```

where:

length

Double

Is the length of the character string in which the character is to be replaced.

source_string

Alphanumeric

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

inhexchar

Double

Is the decimal equivalent of the hexadecimal value of the character to be translated.

outhexchar

Double

Is the decimal equivalent of the hexadecimal value of the character to be used as a substitute for *inhexchar*.

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: Using DCTRAN to Translate Double-Byte Characters

In the following:

```
DCTRAN(8, 'AｱA本B語', 177, 70, A8)
```

For AｱA本B語, the result is AFA本B語.

DSTRIP: Removing a Single-Byte or Double-Byte Character From a String

How to:

Remove a Single-Byte or Double-Byte Character From a String

Example:

Removing a Double-Byte Character From a String

The DSTRIP function removes all occurrences of a specific single-byte or double-byte 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 Single-Byte or Double-Byte Character From a String

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

where:

length

Double

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.

`outfield`

Alphanumeric

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

Example: Removing a Double-Byte Character From a String

In the following:

```
DSTRIP(9, 'A日A本B語', '日', A9)
```

For A日A本B語, the result is AA本B語.

EDIT: Extracting or Adding Characters

How to:

Extract or Add Characters

Example:

Extracting and Adding a Character to a Field

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 does not require an outfield argument because the result is obviously alphanumeric and its size is determined from the mask value.

EDIT can also convert the format of a field. For information on converting a field with EDIT, see *EDIT: Converting the Format of a Field* in Chapter 8, *Format Conversion Functions*.

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 $, determines the length of the output field.

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)

How to:

Extract a Substring (Token)

Example:

Extracting a Token From a Field

The GETTOK function divides a character string into substrings, called tokens. A specific character, called a delimiter, occurs in the string and separates the string into tokens. GETTOK returns the token specified by the token_number. GETTOK ignores leading and trailing blanks in the parent character string.

For example, suppose you want to extract the fourth word from a sentence. Use the space character for a delimiter and four for the token_number. GETTOK divides the sentence into words using this delimiter, then extracts the fourth word. If the string is not divided by the delimiter, use the PARAG function for this purpose.

Syntax: How to Extract a Substring (Token)

```
GETTOK(infield, inlen, token_number, '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_number

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:

```
                      LAST TOKEN
ADDRESS_LN3           (ZIP CODE)
-----------           ----------
RUTHERFORD NJ 07073   07073
NEW YORK NY 10039     10039
FREEPORT NY 11520     11520
NEW YORK NY 10001     10001
FREEPORT NY 11520     11520
ROSELAND NJ 07068     07068
JERSEY CITY NJ 07300  07300
FLUSHING NY 11354     11354
```

JPTRANS: Converting Japanese-Specific Characters

How to:

Convert Japanese Specific Characters

Reference:

Usage Notes for the JPTRANS Function

Example:

Using the JPTRANS Function

The JPTRANS function converts Japanese specific characters.

Syntax: How to Convert Japanese Specific Characters

```
JPTRANS('conversion_type', length, source_string, outfield)
```

where:

conversion_type

Is one of the following options indicating the type of conversion you want to apply to Japanese specific characters. These are the single component input types:

Conversion Type	Description
'UPCASE'	Converts Zenkaku(Fullwidth) alphabets to Zenkaku uppercase.
'LOCASE'	Converts Zenkaku alphabets to Zenkaku lowercase.
'HNZNALPHA'	Converts alphanumerics from Hankaku (Halfwidth) to Zenkaku.
'HNZNSIGN'	Converts ASCII symbols from Hankaku to Zenkaku.
'HNZNKANA'	Converts Katakana from Hankaku to Zenkaku.
'HNZNSPACE'	Converts space (blank) from Hankaku to Zenkaku.
'ZNHNALPHA'	Converts alphanumerics from Zenkaku to Hankaku.
'ZNHNSIGN'	Converts ASCII symbols from Zenkaku to Hankaku.
'ZNHNKANA'	Converts Katakana from Zenkaku to Hankaku.
'ZNHNSPACE'	Converts space from Zenkaku to Hankaku.
'HIRAKATA'	Converts Hiragana to Zenkaku Katakana.
'KATAHIRA'	Converts Zenkaku Katakana to Hiragana.

Conversion Type	Description
`'930TO939'`	Converts codepage from 930 to 939.
`'939TO930'`	Converts codepage from 939 to 930.

`length`

Integer

Is the number of characters in the source string.

`source_string`

Alphanumeric

Is the string to convert.

`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: Using the JPTRANS Function

In the following:

```
JPTRANS('UPCASE', 20, Alpha_DBCS_Field, 'A20')
```

For ａｂｃ, the result is ＡＢＣ.

```
JPTRANS('LOCASE', 20, Alpha_DBCS_Field, 'A20')
```

For ＡＢＣ, the result is ａｂｃ.

```
JPTRANS('HNZNALPHA', 20, Alpha_SBCS_Field, 'A20')
```

For AaBbCc123, the result is ＡａＢｂＣｃ１２３.

```
JPTRANS('HNZNSIGN', 20, Symbol_SBCS_Field, 'A20')
```

For !@$%,.?, the result is ！＠＄％、。？

```
JPTRANS('HNZNKANA', 20, Hankaku_Katakana_Field, 'A20')
```

For ｢ﾍﾞｰｽﾎﾞｰﾙ｡｣, the result is 「ベースボール。」

```
JPTRANS('HNZNSPACE', 20, Hankaku_Katakana_Field, 'A20')
```

For ｱｲｳ, the result is ｱ　ｲ　ｳ

```
JPTRANS('ZNHNALPHA', 20, Alpha_DBCS_Field, 'A20')
```

For Ａａ Ｂｂ Ｃｃ １２３, the result is AaBbCc123.

```
JPTRANS('ZNHNSIGN', 20, Symbol_DBCS_Field, 'A20')
```

For ！＠＄％、。？, the result is !@$%,.?

```
JPTRANS('ZNHNKANA', 20, Zenkaku_Katakana_Field, 'A20')
```

For 「ベースボール。」, the result is ｢ﾍﾞｰｽﾎﾞｰﾙ｡｣

```
PTRANS('ZNHNSPACE', 20, Zenkaku_Katakana_Field, 'A20')
```

For ア　イ　ウ, the result is ア イ ウ

```
JPTRANS('HIRAKATA', 20, Hiragana_Field, 'A20')
```

For あいう, the result is アイウ

```
JPTRANS('KATAHIRA', 20, Zenkaku_Katakana_Field, 'A20')
```

For アイウ, the result is あいう

In the following, codepoints 0x62 0x63 0x64 are converted to 0x81 0x82 0x83, respectively:

```
JPTRANS('930TO939', 20, CP930_Field, 'A20')
```

In the following, codepoints 0x59 0x62 0x63 are converted to 0x81 0x82 0x83, respectively:

```
JPTRANS('939TO930', 20, CP939_Field, 'A20')
```

Reference: Usage Notes for the JPTRANS Function

- ❑ HNZNSIGN and ZNHNSIGN focus on the conversion of symbols.

  Many symbols have a one to one relation between Japanese Fullwidth characters and ASCII symbols, whereas some characters have one to many relations. For example, the Japanese punctuation character (U+3001) and Fullwidth comma ,(U+FF0C) will be converted to the same comma ,(U+002C). We have the following EXTRA rule for those special cases.

  HNZNSIGN:

  - ❑ Double Quote " (U+0022) -> Fullwidth Right Double Quote ” (U+201D)
  - ❑ Single Quote ' (U+0027) -> Fullwidth Right Single Quote ’ (U+2019)
  - ❑ Comma , (U+002C) -> Fullwidth Ideographic Comma (U+3001)
  - ❑ Full Stop . (U+002E) -> Fullwidth Ideographic Full Stop ? (U+3002)
  - ❑ Backslash \ (U+005C) -> Fullwidth Backslash \ (U+FF3C)
  - ❑ Halfwidth Left Corner Bracket (U+FF62) ->Fullwidth Left Corner Bracket (U+300C)
  - ❑ Halfwidth Right Corner Bracket (U+FF63) -> Fullwidth Right Corner Braket (U+300D)
  - ❑ Halfwidth Katakana Middle Dot ? (U+FF65) -> Fullwidth Middle Dot · (U+30FB)

  ZNHNSIGN:

  - ❑ Fullwidth Right Double Quote ” (U+201D) -> Double Quote " (U+0022)
  - ❑ Fullwidth Left Double Quote “ (U+201C) -> Double Quote " (U+0022)
  - ❑ Fullwidth Quotation " (U+FF02) -> Double Quote " (U+0022)
  - ❑ Fullwidth Right Single Quote ’ (U+2019) -> Single Quote ' (U+0027)
  - ❑ Fullwidth Left Single Quote ‘ (U+2018) -> Single Quote ' (U+0027)
  - ❑ Fullwidth Single Quote ' (U+FF07) -> Single Quote ' (U+0027)
  - ❑ Fullwidth Ideographic Comma (U+3001) -> Comma , (U+002C)
  - ❑ Fullwidth Comma , (U+FF0C) -> Comma , (U+002C
  - ❑ Fullwidth Ideographic Full Stop ? (U+3002) -> Full Stop . (U+002E)
  - ❑ Fullwidth Full Stop . (U+FF0E) -> Full Stop . (U+002E)
  - ❑ Fullwidth Yen Sign ¥ (U+FFE5) -> Yen Sign ¥ (U+00A5)
  - ❑ Fullwidth Backslash \ (U+FF3C) -> Backslash \ (U+005C)
  - ❑ Fullwidth Left Corner Bracket (U+300C) -> Halfwidth Left Corner Bracket (U+FF62)

- ❑ Fullwidth Right Corner Bracket (U+300D) -> Halfwidth Right Corner Bracket (U+FF63)
- ❑ Fullwidth Middle Dot · (U+30FB) -> Halfwidth Katakana Middle Dot · (U+FF65)

❑ HNZNKANA and ZNHNKANA focus on the conversion of Katakana

They convert not only letters but also punctuation symbols on the following list:

- ❑ Fullwidth Ideographic Comma (U+3001) <-> Halfwidth Ideographic Comma (U+FF64)
- ❑ Fullwidth Ideographic Full Stop (U+3002) <-> Halfwidth Ideographic Full Stop (U+FF61)
- ❑ Fullwidth Left Corner Bracket (U+300C) <-> Halfwidth Left Corner Braket (U+FF62)
- ❑ Fullwidth Right Corner Bracket (U+300D) <-> Halfwidth Right Corner Bracket (U+FF63)
- ❑ Fullwidth Middle Dot · (U+30FB) <-> Halfwidth Katakana Middle Dot · (U+FF65)
- ❑ Fullwidth Prolonged Sound (U+30FC) <-> Halfwidth Prolonged Sound (U+FF70)

❑ JPTRANS can be nested for multiple conversions.

For example, text data may contain fullwidth numbers and fullwidth symbols. In some situations, they should be cleaned up for ASCII numbers and symbols.

For バンゴウ＃１２３, the result is バンゴウ#123

```
JPTRANS('ZNHNALPHA', 20, JPTRANS('ZNHNSIGN', 20, Symbol_DBCS_Field,
'A20'), 'A20')
```

❑ HNZNSPACE and ZNHNSPACE focus on the conversion of a space (blank character).

Currently only conversion between U+0020 and U+3000 is supported.

LCWORD: Converting a Character String to Mixed-Case

How to:

Convert a Character String to Mixed-Case

Example:

Converting a Character String to Mixed-Case

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 4, *Maintain-specific Character Functions*.

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

How to:

Left-Justify a Character String

Example:

Left-Justifying a Field

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 4, *Maintain-specific Character Functions*.

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

The following request creates the XNAME field in which the last names are not left-justified. Then, 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

How to:

Convert Text to Lowercase

Example:

Converting a Field to Lowercase

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

How to:

Overlay a Character String

Example:

Replacing Characters in a Character String

Overlaying a Character in a String With MODIFY

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 4, *Maintain-specific Character Functions*.

Syntax: How to Overlay a Character String

```
OVRLAY(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

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

The COMPUTE 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

How to:

Divide Text Into Smaller Lines

Example:

Dividing Text Into Smaller Lines

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

How to:

Find the Beginning of a Substring

Example:

Finding the Position of a Letter

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 4, *Maintain-specific Character Functions*.

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

How to:

Right-Justify a Character String

Example:

Right-Justifying a Field

The RJUST function right-justifies a character string. All trailing blacks 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 4, *Maintain-specific Character Functions*.

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 enclosed in single quotation marks or a field 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.

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

How to:

Compare Character Strings Phonetically

Example:

Comparing Character Strings Phonetically

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

2-byte Alphanumeric

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 01 to 99, expressed with two digits (for example '01'); 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
```

SPELLNM: Spelling Out a Dollar Amount

How to:

Spell Out a Dollar Amount

Example:

Spelling Out a Dollar Amount

The SPELLNM function spells out an alphanumeric string or numeric value containing two decimal places as dollars and cents. For example, the value 32.50 is THIRTY TWO DOLLARS AND FIFTY CENTS. This function is available only for WebFOCUS.

Syntax: How to Spell Out a Dollar Amount

```
SPELLNM(outlength, number, outfield)
```

where:

outlength

Integer

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

If you know the maximum value of *number*, use the following table to determine the value of *outlength*:

If number is less than...	...outlength should be
$10	37
$100	45
$1,000	59
$10,000	74
$100,000	82
$1,000,000	96

number

Alphanumeric or Numeric (9.2)

Is the number to be spelled out. This value must contain two decimal places.

`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: Spelling Out a Dollar Amount

SPELLNM spells out the values in CURR_SAL and stores the result in AMT_IN_WORDS:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
AMT_IN_WORDS/A82 = SPELLNM(82, CURR_SAL, AMT_IN_WORDS);
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

```
CURR_SAL     AMT_IN_WORDS

$13,200.00   THIRTEEN THOUSAND TWO HUNDRED DOLLARS AND NO CENTS
$18,480.00   EIGHTEEN THOUSAND FOUR HUNDRED EIGHTY DOLLARS AND NO CENTS
$18,480.00   EIGHTEEN THOUSAND FOUR HUNDRED EIGHTY DOLLARS AND NO CENTS
$21,780.00   TWENTY-ONE THOUSAND SEVEN HUNDRED EIGHTY DOLLARS AND NO CENTS
$9,000.00    NINE THOUSAND DOLLARS AND NO CENTS
$27,062.00   TWENTY-SEVEN THOUSAND SIXTY-TWO DOLLARS AND NO CENTS
```

SQUEEZ: Reducing Multiple Spaces to a Single Space

How to:

Reduce Multiple Spaces to a Single Space

Example:

Reducing Multiple Spaces to a Single Space

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

How to:

Remove a Character From a String

Example:

Removing Occurrences of a Character From a String (Reporting)

Removing Single Quotation Marks From a String (Reporting)

Removing Commas From a String (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        SDIR
---------        ----
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 A39:

```
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
```

STRREP: Replacing Character Strings

How to:

Replace Character Strings

Reference:

Usage Notes for STRREP Function

Example:

Replacing Commas and Dollar Signs

The STRREP function enables you to replace all instances of a specified string within a given input string. It also supports replacement by null strings.

Syntax: How to Replace Character Strings

```
STRREP (inlength, instring, searchlength, searchstring, replength,
repstring, outlength, outstring)
```

where:

inlength

Numeric

Is the number of characters in the input string.

instring

Alphanumeric

Is the input string.

searchlength

Numeric

Is the number of characters in the (shorter length) string to be replaced.

searchstring

Alphanumeric

Is the character string to be replaced.

replength

Numeric

Is the number of characters in the replacement string. Must be zero (0) or greater.

repstring

Alphanumeric

Is the replacement string (alphanumeric). Ignored if replength is zero (0).

`outlength`

Numeric

Is the number of characters in the resulting output string. Must be 1 or greater.

`outstring`

Alphanumeric

Is the resulting output string after all replacements and padding.

Reference: Usage Notes for STRREP Function

The maximum string length is 4095.

Example: Replacing Commas and Dollar Signs

In the following example, STRREP finds and replaces commas and dollar signs that appear in the CS_ALPHA field, first replacing commas with null strings to produce CS_NOCOMMAS (removing the commas) and then replacing the dollar signs ($) with (USD) in the right-most CURR_SAL column:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL NOPRINT
COMPUTE CS_ALPHA/A15=FTOA(CURR_SAL,'(D12.2M)',CS_ALPHA);
        CS_NOCOMMAS/A14=STRREP(15,CS_ALPHA,1,',',0,'X',14,CS_NOCOMMAS);
        CS_USD/A17=STRREP(14,CS_NOCOMMAS,1,'$',4,'USD ',17,CS_USD);
        NOPRINT
        CS_USD/R AS CURR_SAL
BY LAST_NAME
END
```

Here is the output:

```
LAST_NAME           CS_ALPHA           CS_NOCOMMAS                CURR_SAL
---------           --------           -----------         -----------------
BANNING           $29,700.00             $29700.00               USD 29700.00
BLACKWOOD         $21,780.00             $21780.00               USD 21780.00
CROSS             $27,062.00             $27062.00               USD 27062.00
GREENSPAN          $9,000.00              $9000.00               USD  9000.00
IRVING            $26,862.00             $26862.00               USD 26862.00
JONES             $18,480.00             $18480.00               USD 18480.00
MCCOY             $18,480.00             $18480.00               USD 18480.00
MCKNIGHT          $16,100.00             $16100.00               USD 16100.00
ROMANS            $21,120.00             $21120.00               USD 21120.00
SMITH             $22,700.00             $22700.00               USD 22700.00
STEVENS           $11,000.00             $11000.00               USD 11000.00
```

SUBSTR: Extracting a Substring

How to:

Extract a Substring

Example:

Extracting a String

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 4, *Maintain-specific Character Functions*.

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 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

How to:

Remove Leading and Trailing Occurrences

Example:

Removing Leading Occurrences

Removing Trailing Occurrences

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 4, *Maintain-specific Character Functions*.

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

How to:

Convert Text to Uppercase

Example:

Converting a Mixed-Case Field to Uppercase

Converting a Lowercase Field to Uppercase With MODIFY

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 4, *Maintain-specific Character Functions*.

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 of type A*n*v or A*n*

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

Reference:

Usage Notes for Using an AnV Field in a Function

A*n*V 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 A*n*V 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 A*n*V argument in a function, the input parameter is treated as an A*n* parameter and is padded with blanks to its declared size (*n*). If the last parameter specifies an A*n*V format, the function result is converted to type A*n*V 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 A*n*V 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 A*n*V 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

How to:

Find the Length of an Alphanumeric Field

Example:

Finding the Length of an AnV Field

LENV returns the actual length of an A*n*V field or the size of an A*n* field.

Syntax: How to Find the Length of an Alphanumeric Field

```
LENV(string, outfield)
```

where:

string

Alphanumeric of type A*n* or A*n*V

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

`outfield`

Integer

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

Example: Finding the Length of an A*n*V Field

TRIMV creates an AnV field named TITLEV by removing trailing blanks from the TITLE value. Then LENV 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

How to:

Create a Variable Length Lowercase String

Example:

Creating a Variable Length Lowercase String

LOCASV converts alphabetic characters to lowercase. This is similar to LOCASE, but LOCASV can return A*n*V output whose actual length is the lesser of the actual length of the A*n*V 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

Integer

Is the maximum length of the input string.

string

Alphanumeric of type A*n* or A*n*V

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 A*n* or A*n*V format. If it is a field of type A*n*V, 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 of type A*n* or A*n*V

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 A*n*V or A*n* format. Is the field has A*n*V or A*n* 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/A15V = 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

How to:

Find the Beginning of a Variable Length Substring

Example:

Finding the Starting Position of a Variable Length Pattern

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 A*n*V 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 of type A*n* or A*n*V

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 A*n* or A*n*V format. If it is a field of type A*n*V, 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 of type A*n* or A*n*V

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 A*n* or A*n*V format. If it is a field of type A*n*V, 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

Integer

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:

```
                          Pattern  Pattern
 TITLE                      Start   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

How to:

Extract a Variable Length Substring

Example:

Extracting a Variable Length Substring

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 A*n*V 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`

Integer

Is the maximum length of the input string.

`parent`

Alphanumeric of type A*n* or A*n*V

Is the parent string enclosed in single quotation marks, or the field containing the parent string. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, 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 of type A*n* or A*n*V

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 A*n* or A*n*V format.

If the format of *outfield* is A*n*V, 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:

```
                        Pattern Pattern  Trimmed
TITLE                     Start  Length  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

How to:

Remove Characters From a String

Example:

Creating an AnV Field by Removing Trailing Blanks

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 A*n*V format.

TRIMV is useful for converting an A*n* field to an A*n*V 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, slength_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 of type A*n* or A*n*V

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

slength_limit

Integer

Is the maximum length of the input string.

pattern

Alphanumeric of type A*n* or A*n*V

Is the pattern to remove enclosed in single quotation marks. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, 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 of type A*n* or A*n*V

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 A*n*V or A*n* format.

If the format of *outfield* is A*n*V, the actual length is equal to the number of characters left after trimming.

Example: Creating an A*n*V Field by Removing Trailing Blanks

TRIMV creates an A*n*V 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:

```
CATEGORY  DIRECTOR        TITLEV
--------  --------        ------

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

How to:

Create a Variable Length Uppercase String

Example:

Creating a Variable Length Uppercase String

UPCASV converts alphabetic characters to uppercase like UPCASE. However, UPCASV can return A*n*V output whose actual length is the lesser of the actual length of the A*n*V 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

Integer

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 of type A*n* or A*n*V

Is the character string enclosed in single quotation marks, or the field containing the character string. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, 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 of type A*n* or A*n*V

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 A*n*V, 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/A15V=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
```

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
```

4 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 3, *Character Functions*.

Topics:

- ❏ CHAR2INT: Translating a Character Into an Integer Value
- ❏ INT2CHAR: Translating an Integer Value Into 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
- ❏ OVRLAY: 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 Substring 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 Into an Integer Value

How to:

Translate a Character Into an Integer Value

Example:

Translating a Character Into an Integer Value

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 Into a Character

How to:

Translate an Integer Value Into a Character

Example:

Translating an Integer Value Into a Character

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

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

How to:

Convert a Character String to Mixed-Case

Example:

Converting a Character String to Mixed-Case

The LCWORDand 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 *Accessing and Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

There is also an LCWORD function available for both the reporting and Maintain languages. For information on this function, see *Character Functions* in Chapter 3, *Character Functions*.

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

How to:

Determine the Length of a Character String

Example:

Determining the Length of a Character String

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)

How to:

Left-Justify a Character String

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 *Accessing and Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

There is also an LJUST function available for the reporting language. For information on this function, see *Character Functions* in Chapter 3, *Character Functions*.

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

How to:

Convert a Character String to Lowercase

The LOWER function converts a character string to lowercase.

To use this function, you must import the function library MNTUWS. For more information on importing this library see *Accessing and Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

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

How to:

Extract or Add Characters

Example:

Extracting a Character From a Field

Adding Dashes to a Field

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:

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

How to:

Convert Characters From the Native English Code Page

Example:

Converting Characters From the Native English Code Page

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 appear whenever the variable ADOLLAR is used, regardless of the code page being run.

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

OVRLAY: Overlaying a Character String (Maintain)

How to:

Overlay a Character String

Example:

Overlaying a Character String

The OVRLAY 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 *Accessing and Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

There is also an OVRLAY function available for the reporting language. For information on this function, see *Character Functions* in Chapter 3, *Character Functions*.

Syntax: How to Overlay a Character String

```
OVRLAY(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)

How to:

Find the Beginning of a Substring

Example:

Finding the Beginning of a Substring

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 *Accessing and Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

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 3, *Character Functions*.

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)

How to:

Right-Justify a Character String

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 3, *Character Functions*.

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 with which to pad the character string and right-justify it. RJUST uses *char* only when *length* is greater than the length of *string*.

SELECTS: Decoding a Value From a Stack

How to:

Decode a Value From a Stack

Example:

Decoding Values With SELECTS

Decoding a Value From a Stack

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 to extract 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

How to:

Substitute a Substring

Example:

Substituting One String for Another

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, import the function library MNTUWS. For more information on importing this library see *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

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

How to:

Compare Character Strings

Example:

Comparing Character Strings

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

How to:

Compare Character Strings and Ignore Case

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

How to:

Compare Character Substrings

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*.

STRTOKEN: Extracting a Substring Based on Delimiters

How to:

Extract a Substring

Example:

Extracting a Substring

The STROKEN 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 more information on importing this library see *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

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)

How to:

Extract a Substring

Example:

Extracting the First Character of a String in 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 3, *Character Functions*.

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)

How to:

Remove Trailing Occurrences

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 3, *Character Functions*.

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

How to:

Determine the Length of a String Excluding Trailing Spaces

Example:

Determining the Length of a String Excluding Trailing Spaces

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)

How to:

Convert Text to Uppercase

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 *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

There is also an UPCASE function available for the reporting language. For information on this function, see *UPCASE: Converting Text to Uppercase* in Chapter 3, *Character Functions*.

Syntax: How to Convert Text to Uppercase

```
UPCASE(string)
```

where:

string

Alphanumeric

Is the character string to be converted to uppercase.

5 Data Source and Decoding Functions

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.

Topics:

- ❏ DB_LOOKUP: Retrieving a Value From a Lookup Data Source
- ❏ DECODE: Decoding Values
- ❏ FIND: Verifying the Existence of a Value in a Data Source
- ❏ LAST: Retrieving the Preceding Value
- ❏ LOOKUP: Retrieving a Value From a Cross-referenced Data Source

DB_LOOKUP: Retrieving a Value From a Lookup Data Source

How to:

Retrieve a Value From a Lookup Data Source

Reference:

Usage Notes for DB_LOOKUP

Example:

Retrieving a Value From a Fixed Format Sequential File in a TABLE Request

The DB_LOOKUP function enables you to retrieve a value from one data source when running a request against another data source, without joining or combining the two data sources.

DB_LOOKUP compares pairs of fields from the source and lookup data sources to locate matching records and retrieve the value to return to the request. You can specify as many pairs as needed to get to the lookup record that has the value you want to retrieve. If your field list pairs do not lead to a unique lookup record, the first matching lookup record retrieved is used.

DB_LOOKUP can be called in a DEFINE command, TABLE COMPUTE command, or MODIFY COMPUTE command.

There are no restrictions on the source file. The lookup file can be any non-FOCUS data source that is supported as the cross referenced file in a cluster join. The lookup fields used to find the matching record are subject to the rules regarding cross-referenced join fields for the lookup data source. A fixed format sequential file can be the lookup file if it is sorted in the same order as the source file.

Syntax: How to Retrieve a Value From a Lookup Data Source

```
DB_LOOKUP(look_mf, srcfld1, lookfld1, srcfld2, lookfld2, ..., returnfld);
```

where:

look_mf

Is the lookup Master File.

srcfld1, srcfld2 ...

Are fields from the source file used to locate a matching record in the lookup file.

lookfld1, lookfld2 ...

Are columns from the lookup file that share values with the source fields. Only columns in the table or file can be used; columns created with DEFINE cannot be used. For multi-segment synonyms only columns in the top segment can be used.

returnfld

Is the name of a column in the lookup file whose value is returned from the matching lookup record. Only columns in the table or file can be used; columns created with DEFINE cannot be used.

Reference: Usage Notes for DB_LOOKUP

- The maximum number of pairs that can be used to match records is 63.
- If the lookup file is a fixed format sequential file, it must be sorted and retrieved in the same order as the source file. The sequential file's key field must be the first lookup field specified in the DB_LOOKUP request. If it is not, no records will match.

  In addition, if a DB_LOOKUP request against a sequential file is issued in a DEFINE FILE command, you must clear the DEFINE FILE command at the end of the TABLE request that references it or the lookup file will remain open. It will not be reusable until closed and may cause problems when you exit FOCUS. Other types of lookup files can be reused without clearing the DEFINE. They will be cleared automatically when all DEFINE fields are cleared.
- If the lookup field has the MISSING=ON attribute in its Master File and the DEFINE or COMPUTE command specifies MISSING ON, the missing value is returned when the lookup field is missing. Without MISSING ON in both places, the missing value is converted to a default value (blank for an alphanumeric field, zero for a numeric field).
- Source records display on the report output even if they lack a matching record in the lookup file.
- Only real fields in the lookup Master File are valid as lookup and return fields.
- If there are multiple rows in the lookup table where the source field is equal to the lookup field, the first value of the return field is returned.

Example: Retrieving a Value From a Fixed Format Sequential File in a TABLE Request

The following procedure creates a fixed format sequential file named GSALE from the GGSALES data source. The fields in this file are PRODUCT (product description), CATEGORY (product category), and PCD (product code). The file is sorted on the PCD field:

```
SET ASNAMES = ON
TABLE FILE GGSALES
SUM PRODUCT CATEGORY
BY PCD
ON TABLE HOLD AS GSALE FORMAT ALPHA
END
```

The following Master File is generated as a result of the HOLD command:

```
FILENAME=GSALE, SUFFIX=FIX     , $
  SEGMENT=GSALE, SEGTYPE=S1, $
    FIELDNAME=PCD, ALIAS=E01, USAGE=A04, ACTUAL=A04, $
    FIELDNAME=PRODUCT, ALIAS=E02, USAGE=A16, ACTUAL=A16, $
    FIELDNAME=CATEGORY, ALIAS=E03, USAGE=A11, ACTUAL=A11, $
```

The following TABLE request against the GGPRODS data source, sorts the report on the field that matches the key field in the lookup file. It retrieves the value of the CATEGORY field from the GSALE lookup file by matching on the product code and product description fields. Note that the DEFINE FILE command is cleared at the end of the request:

```
DEFINE FILE GGPRODS
PCAT/A11 MISSING ON = DB_LOOKUP(GSALE,  PRODUCT_ID, PCD,
         PRODUCT_DESCRIPTION, PRODUCT, CATEGORY);
END
TABLE FILE GGPRODS
PRINT PRODUCT_DESCRIPTION PCAT
BY PRODUCT_ID
END
DEFINE FILE GGPRODS CLEAR
END
```

Because the GSALE Master File does not define the CATEGORY field with the MISSING=ON attribute, the PCAT column displays a blank in those rows that have no matching record in the lookup file:

```
Product
Code     Product           PCAT
-------  -------           ----
B141     Hazelnut
B142     French Roast
B144     Kona
F101     Scone             Food
F102     Biscotti          Food
F103     Croissant         Food
G100     Mug               Gifts
G104     Thermos           Gifts
G110     Coffee Grinder    Gifts
G121     Coffee Pot        Gifts
```

If you add the MISSING=ON attribute to the CATEGORY field in the GSALE Master File, the PCAT column displays a missing data symbol in rows that do not have a matching record in the lookup file:

```
Product
Code     Product           PCAT
-------  -------           ----
B141     Hazelnut           .
B142     French Roast       .
B144     Kona               .
F101     Scone             Food
F102     Biscotti          Food
F103     Croissant         Food
G100     Mug               Gifts
G104     Thermos           Gifts
G110     Coffee Grinder    Gifts
G121     Coffee Pot        Gifts
```

DECODE: Decoding Values

How to:

Supply Values in the Function

Read Values From a File

Example:

Supplying Values in the Function

Reading Values From a File

Reference:

Guidelines for Reading Values From a File

The DECODE function assigns values based on the coded value of an input field. DECODE is useful for giving a more meaningful value to a coded value in a field. 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 4, *Maintain-specific Character Functions*.

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

Alphanumeric or Numeric

Is the coded value for which DECODE searches. 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

Alphanumeric or Numeric

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

Alphanumeric or Numeric

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

Alphanumeric

Is a logical name or a shorthand name that points to the physical file containing the decoded values.

default

Alphanumeric or Numeric

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 interpretedin EBCDIC format 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 applies 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. 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 a Data Source

How to:

Verify the Existence of a Value in a Data Source

Example:

Verifying the Existence of a Value in another Data Source (Maintain)

Verifying the Existence of a Value in the same Data Source (Maintain)

Verifying the Existence of a Value in an Indexed Field (MODIFY)

Rejecting a Transaction When a Value Is Not Found (MODIFY)

Available Languages: MODIFY, Maintain

The FIND function determines if a data value is in a data source field being searched. The function sets a temporary field to 1 (a non-zero value for MODIFY) if the data value is found in the data source field, and to 0 if it is not. FIND does not change the searched file's current database position. A value greater than zero confirms the presence of the data value, not the number of instances in the data source field.

Note: For MODIFY only, the FIND function verifies the existence of a incoming data value in an indexed FOCUS 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 data value is not found in the data source, and 0 if the data value is in the data source.

Syntax: How to Verify the Existence of a Value in a Data Source

```
FIND(fieldname [AS dbfield] IN file);
```

where:

fieldname

Is the name of the search field that contains the data value being searched for in the data source.

dbfield

Is the name of the data source field whose values are compared to the values in the search field.

For Maintain - the AS field is required and the name must be qualified.

For MODIFY - the AS field must be indexed. If the search field and the data source field have the same name, omit this phrase.

file

Is the name of the FOCUS data source.

For Maintain - the IN file is unnecessary since the AS field name is required and must be qualified.

For MODIFY - the IN field must be indexed.

Do not include a space between FIND and the left parenthesis.

Example: Verifying the Existence of a Value in another Data Source (Maintain)

In the following example, FIND determines if a data value is found in another data source.

```
MAINTAIN FILE MOVIES AND VIDEOTRK
FOR ALL NEXT MOVIES.MOVIECODE INTO FILMSTK
TYPE "RC SHOULD BE 1 WHERE MOVIECODE EXISTS IN BOTH FILES";
TYPE " "
COMPUTE RC/I1;
COMPUTE I/I1=1;
REPEAT FILMSTK.FOCCOUNT
  COMPUTE RC= FIND(FILMSTK(I).MOVIECODE AS VIDEOTRK.MOVIECODE)
  TYPE "FOR MOVIECODE = <<FILMSTK(I).MOVIECODE , RC = <<RC"
  COMPUTE I=I+1;
ENDREPEAT
END
```

The output is:

```
RC SHOULD BE 1 WHERE MOVIECODE EXISTS IN BOTH FILES
 FOR MOVIECODE = 001MCA, RC = 1
 .
 .
 .
 FOR MOVIECODE = 387PLA, RC = 0
 .
 .
 .
 FOR MOVIECODE = 963CBS, RC = 1
 TRANSACTIONS: COMMITS   =     1 ROLLBACKS =     0
 SEGMENTS     : INCLUDED =     0 UPDATED    =     0 DELETED    =     0
```

Example: Verifying the Existence of a Value in the same Data Source (Maintain)

In the following example, FIND determines if a data value is found in the same data source.

```
MAINTAIN FILE CAR
COMPUTE RETAIL_COST=31500;
COMPUTE CHECK/I1;
COMPUTE CHECK= FIND (RETAIL_COST);
   IF CHECK = 1 THEN GOTO FOUND1
      ELSE GOTO NOT1;
CASE FOUND1
TYPE "THERE IS A CAR WITH A RETAIL_COST OF <<RETAIL_COST"
-* ....
ENDCASE
CASE NOT1
TYPE "THERE IS NO CAR WITH A RETAIL_COST OF <<RETAIL_COST"
-*....
ENDCASE
-*….
END
```

The output is:

```
THERE IS A CAR WITH A RETAIL_COST OF    31,500
 TRANSACTIONS: COMMITS    =     1 ROLLBACKS =     0
 SEGMENTS     : INCLUDED  =     0 UPDATED    =     0 DELETED    =     0
```

Example: Verifying the Existence of a Value in an Indexed Field (MODIFY)

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 it 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 it prints STUDENT NOT LISTED IN EDUCATION FILE.

Example: Rejecting a Transaction When a Value Is Not Found (MODIFY)

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

How to:

Retrieve the Preceding Value

Example:

Retrieving the Preceding Value

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:

```
                                                     RUNNING
                                                       TOTAL
DEPARTMENT  LAST_NAME                CURR_SAL         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

In this section:

Using the Extended LOOKUP Function

How to:

Retrieve a Value From a Cross-referenced Data Source

Use the Extended LOOKUP Function

Example:

Reading a Value From a Cross-referenced Data Source

Using a Value in a Host Segment to Search a Data Source

Using the LOOKUP Function With a VALIDATE Command

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:

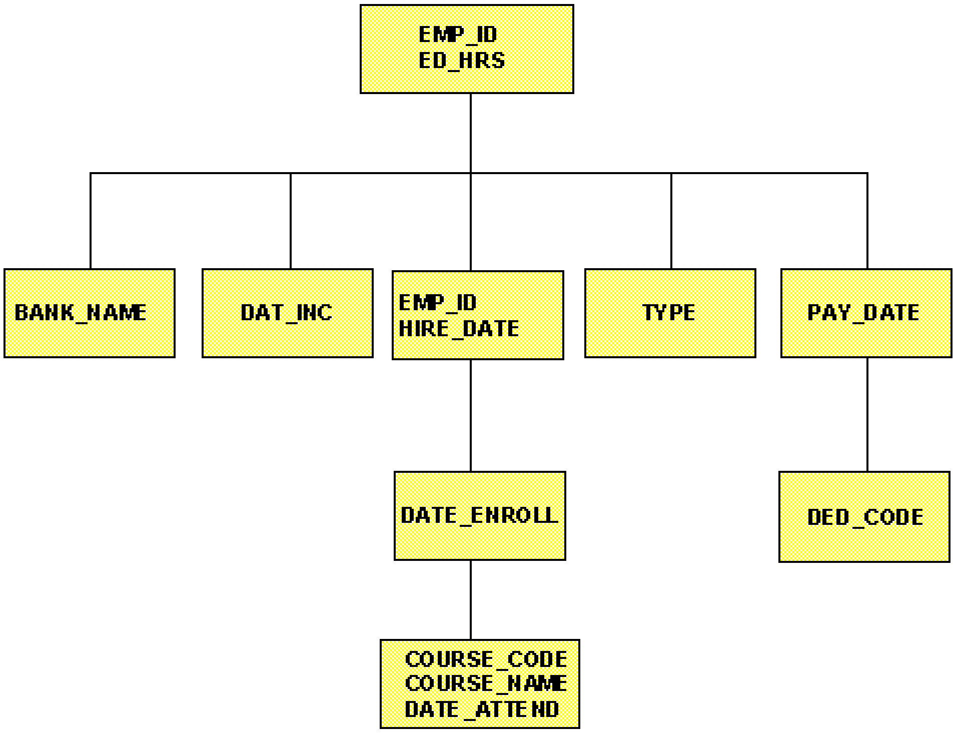
field

  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 following diagram shows the file structure:



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. The transaction is rejected as an error.

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.

6 Date and Time 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 193.
- Legacy date functions for use with legacy dates. For more information, see *Using Legacy Date Functions* on page 268.

  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 7, *Maintain-specific Date and Time Functions*.

Topics:

- Date and Time Function Terminology
- Using Standard Date and Time Functions
- Using Legacy Date 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

In this section:

Specifying Work Days

Enabling Leading Zeros For Date and Time Functions in Dialogue Manager

Using Date and Time Formats

Assigning Date-Time Values

DATEADD: Adding or Subtracting a Date Unit to or From a Date

DATECVT: Converting the Format of a Date

DATEDIF: Finding the Difference Between Two Dates

DATEMOV: Moving a Date to a Significant Point

DATETRAN: Formatting Dates in International Formats

HADD: Incrementing a Date-Time Value

HCNVRT: Converting a Date-Time Value to Alphanumeric Format

HDATE: Converting the Date Portion of a Date-Time Value to a Date Format

HDIFF: Finding the Number of Units Between Two Date-Time Values

HDTTM: Converting a Date Value to a Date-Time Value

HEXTR: Extracting Components of a Date-Time Value and Setting Remaining Components to Zero

HGETC: Storing the Current Date and Time in a Date-Time Field

HHMMSS: Retrieving the Current Time

HINPUT: Converting an Alphanumeric String to a Date-Time Value

HMIDNT: Setting the Time Portion of a Date-Time Value to Midnight

HMASK: Extracting Components of a Date-Time Field and Preserving Remaining Components

HNAME: Retrieving a Date-Time Component in Alphanumeric Format

HPART: Retrieving a Date-Time Component in Numeric Format

HSETPT: Inserting a Component Into a Date-Time Value

HTIME: Converting the Time Portion of a Date-Time Value to a Number

TODAY: Returning the Current Date

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 194.

❏ 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 197.

Specifying Work Days

In this section:

Specifying Business Days

Specifying Holidays

How to:

Set Business Days

View the Current Setting of Business Days

Select a Holiday File

Create a Holiday File

Example:

Setting Business Days to Reflect Your Work Week

Creating and Selecting a Holiday File

Reference:

Rules for Creating a Holiday File

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 as business days or holidays. You can also specify which day is the start of the week. This is used in week computations by the HDIFF, HNAME, HPART, and HSETPT functions.

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 196.
2. Save the file:

   In z/OS, the file must be a member of ERRORS named HDAY*xxxx*.

   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

How to:

Set the Display of Leading Zeros

Example:

Displaying Leading Zeros

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. OFF is the default value.

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

In this section:

Numeric String Format

Formatted-string Format

Translated-string Format

Time Format

How to:

Specify the Order of Date Components in a Date-Time Field

Example:

Using Numeric String Format

Using Formatted-string Format

Using Translated-string Format

Using Time Formats

Using Universal Date-Time Input Values

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.

The DATEFORMAT parameter specifies the order of the date components (month/day/year) when date-time values are entered in the formatted string and translated string formats. This setting is only applied when the meaning of numeric date components cannot be determined by any other means, for example when used as parameters in a date-time function.

Syntax: How to Specify the Order of Date Components in a Date-Time Field

```
SET DATEFORMAT = option
```

where:

option

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

Example: Using the DATEFORMAT Parameter

The following request uses a natural date literal with ambiguous numeric day and month components (APR 04 05) as input to the HINPUT function:

```
SET DATEFORMAT = MYD
DEFINE FILE EMPLOYEE
DTFLDYYMD/HYYMDI =  HINPUT(9,'APR 04 05', 8, DTFLDYYMD);
END

TABLE FILE EMPLOYEE
SUM   CURR_SAL NOPRINT DTFLDYYMD
END
```

With DATEFORMAT set to MYD, the value is interpreted as April 5, 1904:

```
DTFLDYYMD
---------
1904-04-05 00:00
```

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 H*nn* 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 Formats

The following are examples of acceptable time formats:

```
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
```

Example: Using Universal Date-Time Input Values

With DTSTANDARD settings of STANDARD and STANDARDU, the following date-time values can be read as input:

Input Value	Description
`14:30[:20,99]`	Comma separates time components instead of period
`14:30[:20.99]Z`	Universal time

`15:30[:20,99]+01`
`15:30[:20,99]+0100`
`15:30[:20,99]+01:00`
	Each of these is the same as above in Central European Time

`09:30[:20.99]-05`	Same as above in Eastern Standard Time

Note that these values are stored identically internally with the STANDARDU setting. With the STANDARD setting, everything following the Z, +, or - is ignored.

Assigning Date-Time Values

In this section:

Specifying the First Day of the Week for Use in Date-Time Functions

How to:

Assign Date-Time Values

Set a Day as the Start of the Week

View the Current Setting of WEEKFIRST

Enable Strict Processing of Date-Time Values

Example:

Assigning a Date-Time Value in a COMPUTE Command

Assigning a Date-Time Value in WHERE Criteria

Assigning a Date-Time Value in IF Criteria

Setting Sunday as the Start of the Week

Reference:

Arguments for Use With Date and Time Functions

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
```

Specifying the First Day of the Week for Use in Date-Time Functions

The WEEKFIRST parameter specifies a day of the week as the start of the week. This is used in week computations by the HDIFF, HNAME, HPART, and HSETPT functions. The WEEKFIRST parameter does not change the day of the month that corresponds to each day of the week, but only specifies which day is considered the start of the week.

Syntax: How to Set a Day as the Start of the Week

```
SET WEEKFIRST = {n|7}
```

where:

n

Is a number from 1 to 7, where 1 represents Sunday and 7 represents Saturday. The default value is 7. It is consistent with the Microsoft SQL Server convention.

Example: Setting Sunday as the Start of the Week

The following designates Sunday as the start of the week:

```
SET WEEKFIRST = 1
```

Syntax: How to View the Current Setting of WEEKFIRST

```
? SET WEEKFIRST
```

This returns the integer value of the first day of the week. For example, the integer 1 represents Sunday.

Reference: Arguments for Use With Date and Time Functions

The following component names, valid abbreviations, and values are supported as arguments for the date-time functions that require them:

Component Name	Abbreviation	Valid Values
`year`	`yy`	0001-9999
`quarter`	`qq`	1-4
`month`	`mm`	1-12
`day-of-year`	`dy`	1-366
`day` or `day-of-month`	`dd`	1-31 (The two names for the component are equivalent.)
`week`	`wk`	1-53
`weekday`	`dw`	1-7 (Sunday-Saturday)
`hour`	`hh`	0-23
`minute`	`mi`	0-59
`second`	`ss`	0-59
`millisecond`	`ms`	0-999
`microsecond`	`mc`	0-999999

Note:

- 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).

Syntax: How to Enable Strict Processing of Date-Time Values

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

where:

`ON`

Invokes strict processing. ON is the default value.

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

If DTSTRICT is ON and the result would be an invalid date-time value, the function returns the value zero (0).

`OFF`

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

DATEADD: Adding or Subtracting a Date Unit to or From a Date

How to:

Add or Subtract a Date Unit to or From a Date

Example:

Truncation With DATEADD

Using the Weekday Unit

Adding Weekdays to a Date (Reporting)

Determining If a Date Is a Work Day (Reporting)

Adding Months to a Date (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, first DATEADD moves to the next weekday, Monday, then it adds a day. The result is Tuesday.

❑ **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 196 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.

DATEADD works only with full component dates.

Syntax: How to Add or Subtract a Date Unit to or From a Date

`DATEADD(`*`date`*`, '`*`unit`*`', `*`#units`*`, `*`outfield`*`)`

where:

`date`

Date

Is a full component date.

`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`

Date (numeric date offset)

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. In Maintain, you must specify the name of the field.

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.

Note that the single quotes around the number in the first argument, ‘910623’, causes it to be treated as a natural date literal.

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

How to:

Convert a Date Format

Example:

Converting a YYMD Date to DMY

Converting a Legacy Date to Date Format (Reporting)

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.

DATECVT turns off optimization and compilation.

Note: You can use simple assignment instead of calling this function.

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

Format must match the format specified for the output format (*outfmt*).

Is the name of the field that contains the result. In Maintain, you must specify the name of the field.

Example: Converting a YYMD Date to DMY

DATECVT converts 19991231 to 311299 and stores the result in CONV_FIELD:

```
CONV_FIELD/DMY = DATECVT(19991231, 'I8YYMD', 'DMY');
```

or

```
CONV_FIELD/DMY = DATECVT('19991231', 'A8YYMD', '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

How to:

Find the Difference Between Two Dates

Example:

Truncation With DATEDIF

Using Month Calculations

Finding the Number of Weekdays Between Two Dates (Reporting)

Finding the Number of Years Between Two Dates (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 196 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.

DATEDIF works only with full component dates.

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. Is a full component date.

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

Numeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. In Maintain, you must specify the name of the field.

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('March 31 2001', 'May 31 2001', 'M')` yields 2.

`DATEDIF('March 31 2001', 'May 30 2001', 'M')` yields 1 (because May 30 is not the end of the month).

`DATEDIF('March 31 2001', 'April 30 2001', '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/yymd = '20020717';
compute DATE2/yymd = '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

How to:

Move a Date to a Significant Point

Example:

Determining Significant Points for a Date (Reporting)

Determining the End of the Week (Reporting)

Determining the End of the Week (Maintain)

The DATEMOV function moves a date to a significant point on the calendar.

DATEMOV works only with full component dates.

Syntax: How to Move a Date to a Significant Point

```
DATEMOV(date, 'move-point')
```

where:

date

Date

Is a full component date. Is the date to be moved.

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.

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  THU  FRI  WED  THU  THU  MON  WED  THU  THU
1981/11/02  MON  TUE  FRI  MON  MON  TUE  THU  MON  MON
1982/04/01  THU  FRI  WED  THU  THU  MON  WED  THU  THU
1982/05/01  SAT  TUE  THU  MON  FRI  TUE  WED  MON  THU
1981/07/01  WED  THU  TUE  WED  WED  THU  TUE  WED  WED
1981/07/01  WED  THU  TUE  WED  WED  THU  TUE  WED  WED
```

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
```

DATETRAN: Formatting Dates in International Formats

How to:

Format Dates in International Formats

Example:

Using the DATETRAN Function

Reference:

Usage Notes for the DATETRAN Function

The DATETRAN function formats dates in international formats.

Syntax: How to Format Dates in International Formats

```
DATETRAN (indate, '(intype)', '([formatops])', 'lang', outlen, output)
```

where:

indate

Is the input date (in date format) to be formatted. Note that the date format cannot be an alphanumeric or numeric format with date display options (legacy date format).

intype

Is one of the following character strings indicating the input date components and the order in which you want them to display, enclosed in parentheses and single quotation marks:

Single Component Input Type	Description
`'(W)'`	Day of week component only (original format must have only W component).
`'(M)'`	Month component only (original format must have only M component).

Two-Component Input Type	**Description**
`'(YYM)'`	Four-digit year followed by month.
`'(YM)'`	Two-digit year followed by month.
`'(MYY)'`	Month component followed by four-digit year.
`'(MY)'`	Month component followed by two-digit year.

Three- Component Input Type	**Description**
`'(YYMD)'`	Four-digit year followed by month followed by day.
`'(YMD)'`	Two-digit year followed by month followed by day.
`'(DMYY)'`	Day component followed by month followed by four-digit year.
`'(DMY)'`	Day component followed by month followed by two-digit year.
`'(MDYY)'`	Month component followed by day followed by four-digit year.
`'(MDY)'`	Month component followed by day followed by two-digit year.
`'(MD)'`	Month component followed by day (derived from three-component date by ignoring year component).
`'(DM)'`	Day component followed by month (derived from three-component date by ignoring year component).

formatops

Is a string of zero or more formatting options enclosed in parentheses and single quotation marks. The parentheses and quotation marks are required even if you do not specify formatting options. Formatting options fall into the following categories:

- Options for suppressing initial zeros in month or day numbers.
- Options for translating month or day components to full or abbreviated uppercase or default case (mixed case or lowercase depending on the language) names.
- Date delimiter options and options for punctuating a date with commas.

Valid options for suppressing initial zeros in month or day numbers are:

Format Option	Description
`m`	Zero-suppresses months (displays numeric months before October as 1 through 9 rather than 01 through 09).
`d`	Displays days before the tenth of the month as 1 through 9 rather than 01 through 09.
`dp`	Displays days before the tenth of the month as 1 through 9 rather than 01 through 09 with a period after the number.
`do`	Displays days before the tenth of the month as 1 through 9. For English (langcode EN) only, displays an ordinal suffix (st, nd, rd, or th) after the number.

Valid month and day name translation options are:

Format Option	Description
`T`	Displays month as an abbreviated name with no punctuation, all uppercase.
`TR`	Displays month as a full name, all uppercase.
`Tp`	Displays month as an abbreviated name followed by a period, all uppercase.
`t`	Displays month as an abbreviated name with no punctuation. The name is all lowercase or initial uppercase, depending on language code.
`tr`	Displays month as a full name. The name is all lowercase or initial uppercase, depending on language code.
`tp`	Displays month as an abbreviated name followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`W`	Includes an abbreviated day of the week name at the start of the displayed date, all uppercase with no punctuation.
`WR`	Includes a full day of the week name at the start of the displayed date, all uppercase.

Format Option	Description
`Wp`	Includes an abbreviated day of the week name at the start of the displayed date, all uppercase, followed by a period.
`w`	Includes an abbreviated day of the week name at the start of the displayed date with no punctuation. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`wr`	Includes a full day of the week name at the start of the displayed date. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`wp`	Includes an abbreviated day of the week name at the start of the displayed date followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`X`	Includes an abbreviated day of the week name at the end of the displayed date, all uppercase with no punctuation.
`XR`	Includes a full day of the week name at the end of the displayed date, all uppercase.
`Xp`	Includes an abbreviated day of the week name at the end of the displayed date, all uppercase, followed by a period.
`x`	Includes an abbreviated day of the week name at the end of the displayed date with no punctuation. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`xr`	Includes a full day of the week name at the end of the displayed date. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`xp`	Includes an abbreviated day of the week name at the end of the displayed date followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).

Valid date delimiter options are:

Format Option	Description
`B`	Uses a blank as the component delimiter. This is the default if the month or day of week is translated or if comma is used.
`.`	Uses a period as the component delimiter.
`-`	Uses a minus sign as the component delimiter. This is the default when the conditions for a blank default delimiter are not satisfied.
`/`	Uses a slash as the component delimiter.
`|`	Omits component delimiters.
`K`	Uses appropriate Asian characters as component delimiters.
`c`	Places a comma after the month name (following T, Tp, TR, t, tp, or tr). Places a comma and blank after the day name (following W, Wp, WR, w, wp, or wr). Places a comma and blank before the day name (following X, XR, x, or xr).
`e`	Displays the Spanish or Portuguese word de or DE between the day and month and between the month and year. The case of the word de is determined by the case of the month name. If the month is displayed in uppercase, DE is displayed; otherwise de is displayed. Useful for formats DMY, DMYY, MY, and MYY.
`D`	Inserts a comma after the day number and before the general delimiter character specified.
`Y`	Inserts a comma after the year and before the general delimiter character specified.

`lang`

Is the two-character standard ISO code for the language into which the date should be translated, enclosed in single quotation marks. Valid language codes are:

'AR'	Arabic
'CS'	Czech
'DA'	Danish
'DE'	German
'EN'	English
'ES'	Spanish
'FI'	Finnish
'FR'	French
'EL'	Greek
'IW'	Hebrew
'IT'	Italian
'JA'	Japanese
'KO'	Korean
'LT'	Lithuanian
'NL'	Dutch
'NO'	Norwegian
'PO'	Polish
'PT'	Portuguese
'RU'	Russian
'SV'	Swedish
'TH'	Thai
'TR'	Turkish
'TW'	Chinese (Traditional)
'ZH'	Chinese (Simplified)

`outlen`

Numeric

Is the length of the output field in bytes. If the length is insufficient, an all blank result is returned. If the length is greater than required, the field is padded with blanks on the right.

`output`

Alphanumeric

Is the name of the field that contains the translated date, or its format enclosed in single quotation marks.

Reference: Usage Notes for the DATETRAN Function

- ❏ The output field, though it must be type A and not AnV, may in fact contain variable length information, since the lengths of month names and day names can vary, and also month and day numbers may be either one or two bytes long if a zero-suppression option is chosen. Unused bytes are filled with blanks.
- ❏ All invalid and inconsistent inputs result in all blank output strings. Missing data also results in blank output.
- ❏ The base dates (1900-12-31 and 1900-12 or 1901-01) are treated as though the DATEDISPLAY setting were ON (that is, not automatically shown as blanks). To suppress the printing of base dates, which have an internal integer value of 0, test for 0 before calling DATETRAN. For example:

```
RESULT/A40 = IF DATE EQ 0 THEN ' ' ELSE
                DATETRAN (DATE, '(YYMD)', '(.t)', 'FR', 40, 'A40');
```

- ❏ Valid translated date components are contained in files named DTLNG*lng* where *lng* is a three-character code that specifies the language. These files must be accessible for each language into which you want to translate dates.
- ❏ If you use a terminal emulator program, it must be set to use a code page that can display the accent marks and characters in the translated dates. You may not be able to display dates translated into European and Asian characters at the same time. Similarly, if you want to print the translated dates, your printer must be capable of printing the required characters.
- ❏ The DATETRAN function is not supported in Dialogue Manager.

Example: Using the DATETRAN Function

The following request prints the day of the week in the default case of the specific language:

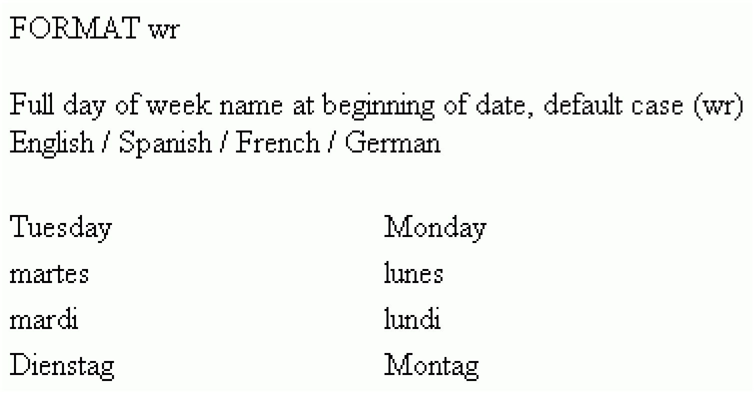
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20051003;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;
```

```
OUT1A/A8=DATETRAN(DATEW, '(W)', '(wr)', 'EN', 8 , 'A8') ;
OUT1B/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'EN', 8 , 'A8') ;
OUT1C/A8=DATETRAN(DATEW, '(W)', '(wr)', 'ES', 8 , 'A8') ;
OUT1D/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'ES', 8 , 'A8') ;
OUT1E/A8=DATETRAN(DATEW, '(W)', '(wr)', 'FR', 8 , 'A8') ;
OUT1F/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'FR', 8 , 'A8') ;
OUT1G/A8=DATETRAN(DATEW, '(W)', '(wr)', 'DE', 8 , 'A8') ;
OUT1H/A8=DATETRAN(DATEW2, '(W)', '(wr)', 'DE', 8 , 'A8') ;
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT wr"
""
"Full day of week name at beginning of date, default case (wr)"
"English / Spanish / French / German"
""
SUM OUT1A AS '' OUT1B AS '' TRANSDATE NOPRINT
OVER OUT1C AS '' OUT1D AS ''
OVER OUT1E AS '' OUT1F AS ''
OVER OUT1G AS '' OUT1H AS ''
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
ON TABLE SET STYLE *
GRID=OFF, $
END
```

The output is:



```
FORMAT wr

Full day of week name at beginning of date, default case (wr)
English / Spanish / French / German

Tuesday             Monday
martes              lunes
mardi               lundi
Dienstag            Montag
```

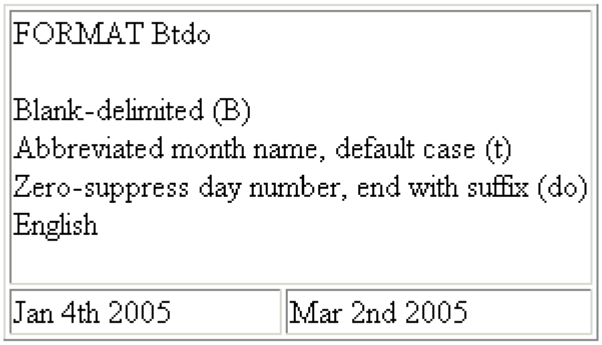
The following request prints a blank delimited date with an abbreviated month name in English. Initial zeros in the day number are suppressed, and a suffix is added to the end of the number:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT2A/A15=DATETRAN(DATEYYMD,  '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
OUT2B/A15=DATETRAN(DATEYYMD2, '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
END
TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdo"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with suffix (do)"
"English"
""
SUM OUT2A AS '' OUT2B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



```
FORMAT Btdo

Blank-delimited (B)
Abbreviated month name, default case (t)
Zero-suppress day number, end with suffix (do)
English

Jan 4th 2005      Mar 2nd 2005
```

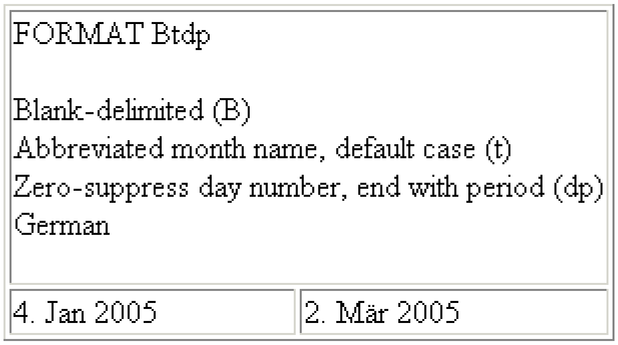
The following request prints a blank delimited date with an abbreviated month name in German. Initial zeros in the day number are suppressed, and a period is added to the end of the number:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT3A/A12=DATETRAN(DATEYYMD,  '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
OUT3B/A12=DATETRAN(DATEYYMD2, '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
END
TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdp"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with period (dp)"
"German"
""
SUM OUT3A AS '' OUT3B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



```
FORMAT Btdp

Blank-delimited (B)
Abbreviated month name, default case (t)
Zero-suppress day number, end with period (dp)
German

4. Jan 2005    2. Mär 2005
```

The following request prints a blank delimited date in French with a full day name at the beginning and a full month name, in lower case (the default for French):

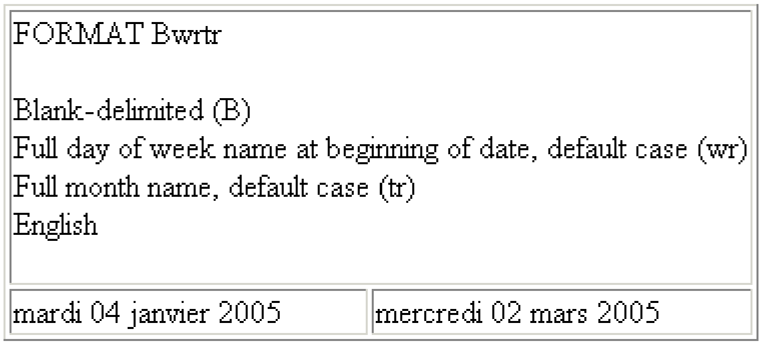
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT4A/A30 = DATETRAN(DATEYYMD,  '(DMYY)', '(Bwrtr)', 'FR', 30, 'A30');
OUT4B/A30 = DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrtr)', 'FR', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrtr"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Full month name, default case (tr)"
"English"
""
SUM OUT4A AS '' OUT4B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



```
FORMAT Bwrtr

Blank-delimited (B)
Full day of week name at beginning of date, default case (wr)
Full month name, default case (tr)
English

mardi 04 janvier 2005     mercredi 02 mars 2005
```

The following request prints a blank delimited date in Spanish with a full day name at the beginning in lowercase (the default for Spanish) followed by a comma, and with the word de between the day number and month and between the month and year:

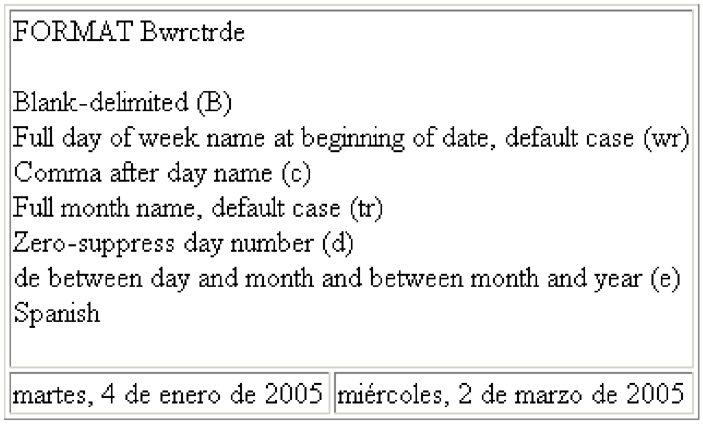
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1     ;
DATEW2/W=TRANS2    ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT5A/A30=DATETRAN(DATEYYMD,  '(DMYY)', '(Bwrctrde)', 'ES', 30, 'A30');
OUT5B/A30=DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrctrde)', 'ES', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrctrde"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Comma after day name (c)"
"Full month name, default case (tr)"
"Zero-suppress day number (d)"
"de between day and month and between month and year (e)"
"Spanish"
""
SUM OUT5A AS '' OUT5B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Bwrctrde

Blank-delimited (B)
Full day of week name at beginning of date, default case (wr)
Comma after day name (c)
Full month name, default case (tr)
Zero-suppress day number (d)
de between day and month and between month and year (e)
Spanish

martes, 4 de enero de 2005	miércoles, 2 de marzo de 2005

The following request prints a date in Japanese characters with a full month name at the beginning, in the default case and with zero suppression:

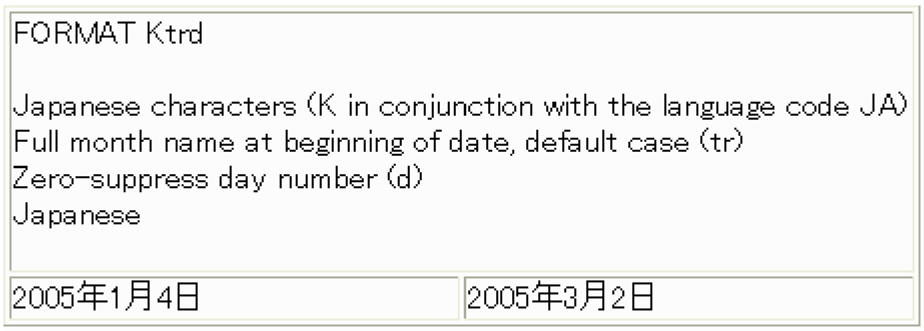
```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1      ;
DATEYYMD2/YYMDW=TRANS2     ;

OUT6A/A30=DATETRAN(DATEYYMD , '(YYMD)', '(Ktrd)', 'JA', 30, 'A30');
OUT6B/A30=DATETRAN(DATEYYMD2, '(YYMD)', '(Ktrd)', 'JA', 30, 'A30');
END

TABLE FILE VIDEOTRK
HEADING
"FORMAT Ktrd"
""
"Japanese characters (K in conjunction with the language code JA)"
"Full month name at beginning of date, default case (tr)"
"Zero-suppress day number (d)"
"Japanese"
""
SUM OUT6A AS '' OUT6B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Ktrd

Japanese characters (K in conjunction with the language code JA)
Full month name at beginning of date, default case (tr)
Zero-suppress day number (d)
Japanese

2005年1月4日	2005年3月2日

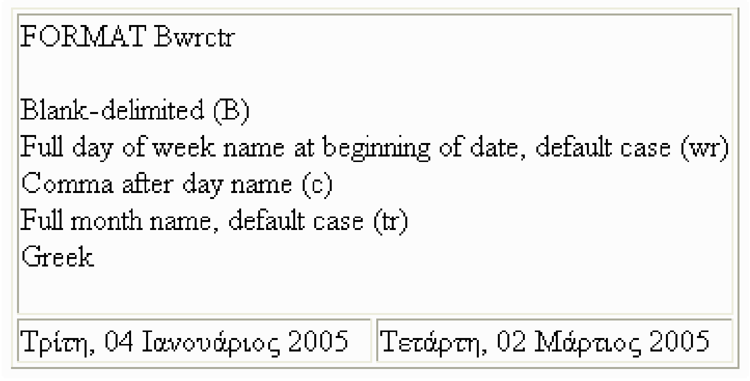
The following request prints a blank delimited date in Greek with a full day name at the beginning in the default case followed by a comma, and with a full month name in the default case:

```
DEFINE FILE VIDEOTRK
TRANS1/YYMD=20050104;
TRANS2/YYMD=20050302;

DATEW/W=TRANS1      ;
DATEW2/W=TRANS2     ;
DATEYYMD/YYMDW=TRANS1     ;
DATEYYMD2/YYMDW=TRANS2    ;

OUT7A/A30=DATETRAN(DATEYYMD , '(DMYY)', '(Bwrctr)', 'GR', 30, 'A30');
OUT7B/A30=DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrctr)', 'GR', 30, 'A30');
END
TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrctrde"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Comma after day name (c)"
"Full month name, default case (tr)"
"Greek"
""
SUM OUT7A AS '' OUT7B AS '' TRANSDATE NOPRINT
ON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:



FORMAT Bwrctr

Blank-delimited (B)
Full day of week name at beginning of date, default case (wr)
Comma after day name (c)
Full month name, default case (tr)
Greek

Τρίτη, 04 Ιανουάριος 2005	Τετάρτη, 02 Μάρτιος 2005

HADD: Incrementing a Date-Time Value

How to:

Increment a Date-Time Value

Example:

Incrementing the Month Component of a Date-Time Field (Reporting)

Incrementing the Month Component of a Date-Time Field (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

Date-time

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

Alphanumeric

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 208.

Note: WEEKDAY is not a valid component for HADD.

increment

Integer

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

Integer

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

Date-time

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 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
------  ---------         ---------
1237    2000/02/05 03:30  2000/04/05 03:30:00
1118    2000/06/26 05:45  2000/08/26 05:45:00
```

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

How to:

Convert a Date-Time Value to Alphanumeric Format

Example:

Converting a Date-Time Field to Alphanumeric Format (Reporting)

Converting a Date-Time Field to Alphanumeric Format (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

Date-time

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

Alphanumeric

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

Integer

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 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, '(HYYMDS)', 20, 'A20');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ALPHA_DATE_TIME1      ALPHA_DATE_TIME2
------  ---------         ----------------      ----------------
1237    2000/02/05 03:30  20000205033000000     2000/02/05 03:30:00
1118    2000/06/26 05:45  20000626054500000     2000/06/26 05:45:00
```

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,'(HYYMDH)',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

How to:

Convert the Date Portion of a Date-Time Value to a Date Format

Example:

Converting the Date Portion of a Date-Time Field to a Date Format (Reporting)

Converting the Date Portion of a Date-Time Field to a Date Format (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

Date-time

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

Date

Is the output format. The value must be YYMD. YYMD is a constant value and can not be changed in this syntax, although you can change the format in subsequent DEFINEs or COMPUTEs.

outfield

YYMD

Is the field that contains the result. This form of the output parameter is required 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
------  ---------         --------------
1237    2000/02/05 03:30  2000/02/05
1118    2000/06/26 05:45  2000/06/26
```

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

How to:

Find the Number of Units Between Two Date-Time Values

Example:

Finding the Number of Days Between Two Date-Time Fields (Reporting)

Finding the Number of Days Between Two Date-Time Fields (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

Date-time

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

Date-time

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

Alphanumeric

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

Floating-point double-precision

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 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
------  ---------         ---------                ---------
1237    2000/02/05 03:30  2000/04/05 03:30:00      60.00
1118    2000/06/26 05:45  2000/08/26 05:45:00      61.00
```

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

How to:

Convert a Date Value to a Date-Time Value

Example:

Converting a Date Field to a Date-Time Field (Reporting)

Converting a Date Field to a Date-Time Field (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

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

Integer

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

Date-time

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 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
------  ---------         --------------  ---
1237    2000/02/05 03:30  2000/02/05      2000/02/05 12:00AM
1118    2000/06/26 05:45  2000/06/26      2000/06/26 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
```

HEXTR: Extracting Components of a Date-Time Value and Setting Remaining Components to Zero

How to:

Extract Multiple Components From a Date-Time Value

Example:

Extracting Hour and Minute Components Using HEXTR

The HEXTR function extracts one or more components from a date-time value and moves them to a target date-time field with all other components set to zero.

Syntax: How to Extract Multiple Components From a Date-Time Value

```
HEXTR(source, 'componentstring', length, outfield)
```

where:

source

Is the a date-time value from which to extract the specified components.

componentstring

Is a string of codes, in any order, that indicates which components are to be extracted and moved to the output date-time field. The following table shows the valid values. The string is considered to be terminated by any character not in this list:

Code	Description
C	century (the two high-order digits only of the four-digit year)
Y	year (the two low-order digits only of the four-digit year)

Code	Description
YY	Four digit year.
M	month
D	day
H	hour
I	minutes
S	seconds
s	milliseconds (the three high-order digits of the six-digit microseconds value)
u	microseconds (the three low-order digits of the six-digit microseconds value)
m	All six digits of the microseconds 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 the includes microseconds.

outfield

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).

Example: Extracting Hour and Minute Components Using HEXTR

The VIDEOTR2 data source has a date-time field named TRANSDATE of type HYYMDI. The following request selects all records containing the time 09:18AM, regardless of the value of the remaining components:

```
TABLE FILE VIDEOTR2
PRINT TRANSDATE
BY LASTNAME
BY FIRSTNAME
WHERE HEXTR(TRANSDATE, 'HI', 8, 'HYYMDI') EQ DT(09:18AM)
END
```

The output is:

```
LASTNAME         FIRSTNAME   TRANSDATE
--------         ---------   ---------
DIZON            JANET       1999/11/05 09:18
PETERSON         GLEN        1999/09/09 09:18
```

HGETC: Storing the Current Date and Time in a Date-Time Field

How to:

Store the Current Date and Time in a Date-Time Field

Example:

Storing the Current Date and Time in a Date-Time Field (Reporting)

Storing the Current Date and Time in a Date-Time Field (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

Integer

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

Date-time

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 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
------  ---------         ---
1237    2000/02/05 03:30  2000/10/03 15:34:24.000000
1118    2000/06/26 05:45  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

How to:

Retrieve the Current Time

Example:

Retrieving the Current Time

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 7, *Maintain-specific Date and Time Functions*.

Syntax: How to Retrieve the Current Time

```
HHMMSS(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.

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

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

How to:

Convert an Alphanumeric String to a Date-Time Value

Example:

Converting an Alphanumeric String to a Date-Time Value (Reporting)

Converting an Alphanumeric String to a Date-Time Value (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`

Integer

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`

Alphanumeric

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.

`length`

Integer

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`

Date-time

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 Maintain, you must specify the name of the field.

Example: Converting an Alphanumeric String to a Date-Time Value (Reporting)

HCNVRT 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 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20');
DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
WHERE DATE EQ 2000;
END
```

The output is:

```
CUSTID  DATE-TIME         ALPHA_DATE_TIME      DT_FROM_ALPHA
------  ---------         ---------------      -------------
1237    2000/02/05 03:30  20000205033000000    2000/02/05 03:30:00
1118    2000/06/26 05:45  20000626054500000    2000/06/26 05:45:00
```

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
```

HMASK: Extracting Components of a Date-Time Field and Preserving Remaining Components

How to:

Move Multiple Date-Time Components to a Target Date-Time Field

Reference:

Usage Notes for the HMASK Function

Example:

Changing a Date-Time Field Using HMASK

The HMASK function extracts one or more components from a date-time value and moves them to a target date-time field with all other components of the target field preserved.

Syntax: How to Move Multiple Date-Time Components to a Target Date-Time Field

`HMASK(source, 'componentstring', input, length, outfield)`

where:

`source`

Is the date-time value from which the specified components are extracted.

`componentstring`

Is a string of codes, in any order, that indicates which components are to be extracted and moved to the output date-time field. The following table shows the valid values. The string is considered to be terminated by any character not in this list:

Code	Description
C	century (the two high-order digits only of the four-digit year)
Y	year (the two low-order digits only of the four-digit year)
YY	Four digit year.
M	month
D	day
H	hour
I	minutes
S	seconds
s	milliseconds (the three high-order digits of the six-digit microseconds value)
u	microseconds (the three low-order digits of the six-digit microseconds value)
m	All six digits of the microseconds value.

`input`

Is the date-time value that provides all the components for the output that are not specified in the component string.

`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 the includes microseconds.

outfield

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).

Reference: Usage Notes for the HMASK Function

HMASK processing is subject to the DTSTRICT setting. Moving the day (D) component without the month (M) component could lead to an invalid result, which is not permitted if the DTSTRICT setting is ON. Invalid date-time values cause any date-time function to return zeroes.

Example: Changing a Date-Time Field Using HMASK

The VIDEOTRK data source has a date-time field named TRANSDATE of format HYYMDI. The following request changes any TRANSDATE value with a time component greater than 11:00 to 8:30 of the following day. First the HEXTR function extracts the hour and minutes portion of the value and compares it to 11:00. If it is greater than 11:00, the HADD function calls HMASK to change the time to 08:30 and adds one day to the date:

```
DEFINE FILE VIDEOTR2
ORIG_TRANSDATE/HYYMDI = TRANSDATE;
TRANSDATE =
IF HEXTR(TRANSDATE, 'HI', 8, 'HHI') GT DT(12:00)
   THEN HADD (HMASK(DT(08:30), 'HISs', TRANSDATE, 8, 'HYYMDI'), 'DAY',
    1,8, 'HYYMDI')
   ELSE TRANSDATE;
END

TABLE FILE VIDEOTR2
PRINT ORIG_TRANSDATE TRANSDATE
BY LASTNAME
BY FIRSTNAME
WHERE ORIG_TRANSDATE NE TRANSDATE
END
```

The output is:

```
LASTNAME          FIRSTNAME   ORIG_TRANSDATE    TRANSDATE
--------          ---------   --------------    ---------
BERTAL            MARCIA      1999/07/29 12:19  1999/07/30 08:30
GARCIA            JOANN       1998/05/08 12:48  1998/05/09 08:30
                              1999/11/30 12:12  1999/12/01 08:30
PARKER            GLENDA      1999/01/06 12:22  1999/01/07 08:30
RATHER            MICHAEL     1998/02/28 12:33  1998/03/01 08:30
WILSON            KELLY       1999/06/26 12:34  1999/06/27 08:30
```

HMIDNT: Setting the Time Portion of a Date-Time Value to Midnight

How to:

Set the Time Portion of a Date-Time Value to Midnight

Example:

Setting the Time to Midnight (Reporting)

Setting the Time to Midnight (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

Date-time

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

Integer

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

Date-time

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

How to:

Retrieve a Date-Time Component in Alphanumeric Format

Example:

Retrieving the Week Component in Alphanumeric Format (Reporting)

Retrieving the Day Component in Alphanumeric Format (Reporting)

Retrieving the Day Component in Alphanumeric Format (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

Date-time

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

Alphanumeric

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 208 for a list of valid components.

outfield

Alphanumeric, at least A2

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 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
------  ---------         --------------
1237    2000/02/05 03:30  06
1118    2000/06/26 05:45  26
```

When WEEKFIRST is set to three, the output is:

```
CUSTID  DATE-TIME         WEEK_COMPONENT
------  ---------         --------------
1237    2000/02/05 03:30  05
1118    2000/06/26 05:45  25
```

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
------  ---------         -------------
1237    2000/02/05 03:30  05
1118    2000/06/26 05:45  26
```

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

How to:

Retrieve a Date-Time Component in Numeric Format

Example:

Retrieving the Day Component in Numeric Format (Reporting)

Retrieving the Day Component in Numeric Format (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

Date-time

Is a date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

component

Alphanumeric

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 208 for a list of valid components.

outfield

Integer

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

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

How to:

Insert a Component Into a Date-Time Value

Example:

Inserting the Day Component Into a Date-Time Field (Reporting)

Inserting the Day Component Into a Date-Time Field (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

Date-time

Is a date-time value, the name of a date-time field that contains the value, or an expression that returns the value.

component

Alphanumeric

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 208 for a list of valid components.

value

Integer

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

Integer

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

Date-time

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 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

How to:

Convert the Time Portion of a Date-Time Field to a Number

Example:

Converting the Time Portion of a Date-Time Field to a Number (Reporting)

Converting the Time Portion of a Date-Time Field to a Number (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

Integer

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

Date-time

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

Floating-point double-precision

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 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
------  ---------                --------
1237    2000/02/05 03:30   12,600,000.00
1118    2000/06/26 05:45   20,700,000.00
```

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
```

TIMETOTS: Converting a Time to a Timestamp

How to:

Convert a Time to a Timestamp

Example:

Converting a Time to a Timestamp

The TIMETOTS function converts a time to a timestamp, using the current date to supply the date component of its value. The first argument must be in H (date-time) format. The DATE component will be set to the current date.

Syntax: How to Convert a Time to a Timestamp

```
TIMETOTS (time, length, outfield)
```

where:

time

Date-time

Is the time in a date-time format.

length

Integer

Is the length of the result. This can be one of the following:

`8` for time values including milliseconds.

`10` for input time values including microseconds.

outfield

Date-time

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 Time to a Timestamp

TIMETOTS converts a time argument to a timestamp:

```
DEFINE FILE T
TSTMPSEC/HYYMDS = TIMETOTS(TMSEC, 8, 'HYYMDS');
TSTMPMILLI/HYYMDm = TIMETOTS(TMMILLI, 10, 'HYYMDm');
END
TABLE FILE T
PRINT TMSEC TSTMPSEC TMMILLI TSTMPMILLI;
END
```

The output is similar to the following:

```
TMSEC      TSTMPSEC              TMMILLI         TSTMPMILLI
01:02:03   2000-10-22 01:02:03   01:02:03.456789 2000-10-22 01:02:03.456789
11:22:33   2000-10-22 11:22:33   11:22:33.444444 2000-10-22 11:22:33.444444
```

TODAY: Returning the Current Date

How to:

Retrieve the Current Date

Example:

Retrieving the Current Date

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 &DATE*fmt* 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 12/13/2006
DEPARTMENT       CURR_SAL
----------      --------
MIS            $108,002.00
PRODUCTION     $114,282.00
```

Using Legacy Date Functions

In this section:

Using Old Versions of Legacy Date Functions

Using Dates With Two- and Four-Digit Years

AYM: Adding or Subtracting Months to or From Dates

AYMD: Adding or Subtracting Days to or From a Date

CHGDAT: Changing How a Date String Displays

DA Functions: Converting a Date to an Integer

DMY, MDY, YMD: Calculating the Difference Between Two Dates

DOWK and DOWKL: Finding the Day of the Week

DT Functions: Converting an Integer to a Date

GREGDT: Converting From Julian to Gregorian Format

JULDAT: Converting From Gregorian to Julian Format

YM: Calculating Elapsed Months

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

How to:

Activate Old 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 Old Legacy Date Functions

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

where:

ON

Activates the function that supports dates for the year 2000 and later. ON is the default value.

OFF

Deactivates a function that supports dates for the year 2000 and later.

Using Dates With Two- and Four-Digit Years

Example:

Using Four-Digit Years

Using Two-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
      ----     -----      -----
1980/06/02  1980154  19800602
1981/07/01  1981182  19810701
1982/05/01  1982121  19820501
1982/01/04  1982004  19820104
1982/08/01  1982213  19820801
1982/01/04  1982004  19820104
1982/07/01  1982182  19820701
1981/07/01  1981182  19810701
1982/04/01  1982091  19820401
1982/02/02  1982033  19820202
1982/04/01  1982091  19820401
1981/11/02  1981306  19811102
1982/04/01  1982091  19820401
1982/05/15  1982135  19820515
```

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

How to:

Add or Subtract Months to or From a Date

Example:

Adding Months to a Date

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

I4, I4YM, I6, or I6YYM

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

I4YM or I6YYM

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

How to:

Add or Subtract Days to or From a Date

Example:

Adding Days to a Date

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

I6, I6YMD, I8, I8YYMD

Is the legacy date in year-month-day format. 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 How a Date String Displays

How to:

Change the Date Display String

Example:

Converting the Date Display From YMD to MDYYX

Reference:

Short to Long Conversion

Usage Notes for CHGDAT

The CHGDAT function rearranges the year, month, and day portions of an input character string representing a date. It may also convert the input string from long to short or short to long date representation. Long representation contains all three date components: year, month, and day; short representation omits one or two of the date components, such as year, month, or day. The input and output date strings are described by display options that specify both the order of date components (year, month, day) in the date string and whether two or four digits are used for the year (for example, 97 or 1997). CHGDAT reads an input date character string and creates an output date character string that represents the same date in a different way.

Note: CHGDAT requires a date character string as input, not a date itself. Whether the input is a standard or legacy date, convert it to a date character string (using the EDIT or DATECVT functions, for example) before applying CHGDAT.

The order of date components in the date character string is described by display options comprised of the following characters in your chosen order:

Character	Description
`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 in the resulting string, append one of the following characters to the display options for the resulting string:

Character	Description
`T`	Displays the month as a three-letter abbreviation.
`X`	Displays the full name of the month.

Display options can consist of up to five display characters. Characters other than those display options are ignored.

For example: The display options 'DMYY' specify that the date string starts with a two digit day, then two digit month, then four digit year.

Note: Display options are *not* date formats.

Reference: Short to Long Conversion

If you are converting a date from short to long representation (for example, from year-month to year-month-day), the function supplies the portion of the date missing in the short representation, 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)	If DATEFNS=ON, the century will be determined by the 100-year window defined by DEFCENT and YRTHRESH. See *Customizing Your Environment* in *Developing Applications* or *Working With Cross-Century Dates* in the iBase archive for details on DEFCENT and YRTHRESH. If DATEFNS=OFF, the year 19*xx* is supplied, where *xx* is the last two digits in the year.

Syntax: How to Change the Date Display String

```
CHGDAT('in_display_options','out_display_options',date_string,outfield)
```

where:

`'in_display_options'`

A1 to A5

Is a series of up to five display options that describe the layout of *date_string*. These options can be stored in an alphanumeric field or supplied as a literal enclosed in single quotation marks.

`'out_display_options'`

A1 to A5

Is a series of up to five display options that describe the layout of the converted date string. These options can be stored in an alphanumeric field or supplied as a literal enclosed in single quotation marks *date_string*

A2 to A8

Is the input date character string with date components in the order specified by *in_display_options*.

Note that if the original date is in numeric format, you must convert it to a date character string. If *date_string* does not correctly represent the date (the date is invalid), the function returns blank spaces.

outfield

A*xx*, where *xx* is a number of characters large enough to fit the date string specified by *out_display_options*. A17 is long enough to fit the longest date string.

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.

Reference: Usage Notes for CHGDAT

Since CHGDAT uses a date string (as opposed to a date) and returns a date string with up to 17 characters, use the EDIT or DATECVT functions or any other means to convert the date to or from a date character string.

Example: Converting the Date Display From YMD to MDYYX

The EDIT function changes HIRE_DATE from numeric to alphanumeric format. CHGDAT then converts each value in ALPHA_HIRE from displaying the components as YMD to MDYYX and stores the result in HIRE_MDY, which has the format A17. The option X in the output value 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

How to:

Convert a Date to an Integer

Example:

Converting Dates and Calculating the Difference Between Them

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 282.

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

I or P format with date display options.

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.

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. The format of the date returned depends on the function.

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

How to:

Calculate the Difference Between Two Dates

Example:

Calculating the Number of Days Between Two Dates

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

I, P, or A format with date display options.

Is the beginning date, or the name of a field that contains the date.

end

I, P, or A format with date display options.

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:

```
                                                 FIRST PAY   DAYS
LAST_NAME         FIRST_NAME   HIRE_DATE   INCREASE    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

How to:

Find the Day of the Week

Example:

Finding the Day of the Week

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

I6YMD or I8 YMD

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: A3

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

How to:

Convert an Integer to a Date

Example:

Converting an Integer to a Date

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 (for more information, see *DA Functions: Converting a Date to an Integer* on page 278). 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.

Note: When USERFNS is set to LOCAL, DT functions only display a six-digit date.

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`

Integer

Is the number of days since December 31, 1899. The number is truncated to an integer.

`outfield`

I6xxx, where xxx corresponds to the function DTxxx in the above list.

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

How to:

Convert From Julian to Gregorian Format

Example:

Converting From Julian to Gregorian Format

Reference:

DATEFNS Settings for GREGDT

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:

DATEFNS Setting	I6 or I7 Format	I8 Format or Greater
ON	YMD	YYMD
OFF	YMD	YMD

Syntax: How to Convert From Julian to Gregorian Format

```
GREGDT(indate, outfield)
```

where:

indate

I5 or I7

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, I8, I6YMD, or I8YYMD

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

How to:

Convert From Gregorian to Julian Format

Example:

Converting From Gregorian to Julian Format

Reference:

DATEFNS Settings for JULDAT

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	I6 or I7 Format	I8 Format or Greater
ON	YYNNN	YYYYNNN
OFF	YYNNN	YYNNN

Syntax: How to Convert From Gregorian to Julian Format

```
JULDAT(indate, outfield)
```

where:

indate

I6, I8, I6YMD, I8YYMD

Is the date or the name of the field that contains the date in year-month-day format (I6YMD or I8YYMD).

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

How to:

Calculate Elapsed Months

Example:

Calculating Elapsed Months

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

I4YM or I6YYM

Is the start date in year-month format (for example, I4YM). If the date is not valid, the function returns a 0.

todate

I4YM or I6YYM

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
```

7 Maintain-specific Date and Time 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 more information on these functions, see Chapter 6, *Date and Time Functions*.

Topics:

❑ Maintain-specific Standard Date and Time Functions

Maintain-specific Standard Date and Time Functions

In this section:

HHMMSS: Retrieving the Current Time (Maintain)

Initial_HHMMSS: Returning the Time the Application Was Started

Initial_TODAY: Returning the Date the Application Was Started

TODAY: Retrieving the Current Date (Maintain)

TODAY2: Returning the Current Date

ADD: Adding Days to a Date

DAY: Extracting the Day of the Month From a Date

JULIAN: Determining How Many Days Have Elapsed in the Year

MONTH: Extracting the Month From a Date

QUARTER: Determining the Quarter

SETMDY: Setting the Value to a Date

SUB: Subtracting a Value From a Date

WEEKDAY: Determining the Day of the Week for a Date

YEAR: Extracting the Year From a Date

Standard date and time functions are for use with non-legacy dates. For a definition of standard dates and times, see Chapter 6, *Date and Time Functions*.

HHMMSS: Retrieving the Current Time (Maintain)

How to:

Retrieve the Current Time

Example:

Retrieving the Current Time

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 *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

There is also an HHMMSS function available in the reporting language. For information on this function, see *HHMMSS: Retrieving the Current Time* in Chapter 6, *Date and Time Functions*.

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

How to:

Retrieve the Initial Time

The Initial_HHMMSS function returns the time when the Maintain application was started as an 8-character string, with embedded periods separating the hours, minutes, and seconds.

To use this function, you must import the function library MNTUWS. For details on importing this library, see *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

Syntax: How to Retrieve the Initial Time

```
Initial_HHMMSS()
```

Initial_TODAY: Returning the Date the Application Was Started

How to:

Retrieve the Initial Date

The Initial_TODAY function returns the date in MM/DD/YY format when the Maintain application was started as an 8-character string with embedded slashes.

To use this function, you must import the function library MNTUWS. For details on importing this library, see *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

Syntax: How to Retrieve the Initial Date

```
Initial_TODAY()
```

TODAY: Retrieving the Current Date (Maintain)

How to:

Retrieve the Current Date

Example:

Retrieving the Current Date

The TODAY function retrieves the current date from the system in the format MM/DD/YY. 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 *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

There is a version of the TODAY function that is available only in the reporting language. For information on this function, see *TIMETOTS: Converting a Time to a Timestamp* in Chapter 6, *Date and Time Functions*.

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

How to:

Retrieve the Current Date

Example:

Retrieving the Current Date

The TODAY2 function retrieves the current date from the operating system in the format MM/DD/YYYY. Use format A10 with the TODAY2 function to ensure proper results.

To use this function, you must import the function library MNTUWS. For information on importing this library, see *Calling a Function* in Chapter 2, *Accessing and Calling a Function*.

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
```

ADD: Adding Days to a Date

How to:

Add Days to a Date

Example:

Adding Days to a Date

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):

```
DateVar      ADD(DateVar, 10);
-------      -----------------
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

How to:

Extract the Day of the Month From a Date

Example:

Extracting the Day of the Month From a Date

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) from which to extract the day of the month, 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):

```
DATE         DAY(DATE)
----         ---------
01/01/2000   1
01/02/2000   2
01/03/2000   3
```

JULIAN: Determining How Many Days Have Elapsed in the Year

How to:

Determine How Many Days Have Elapsed in the Year

Example:

Determining How Many Days Have Elapsed in the Year

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):

```
DATE          JULIAN(DATE)
----          ------------
01/01/2000    1
02/01/2000    32
03/01/2000    61
```

MONTH: Extracting the Month From a Date

How to:

Extract the Month From a Date

Example:

Extracting the Month From a Date

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) from which to extract the month, 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):

```
DATE          MONTH(DATE)
----          -----------
01/01/2000    1
02/01/2000    2
03/01/2000    3
```

QUARTER: Determining the Quarter

How to:

Determine the Quarter for a Date

Example:

Determining the Quarter for a Date

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) for which to determine the quarter, 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):

```
DATE          QUARTER(DATE)
----          -------------
01/01/2000    1
04/01/2000    2
07/01/2000    3
```

SETMDY: Setting the Value to a Date

How to:

Set a Value to a Date

Example:

Setting a Value to a Date

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 appears 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

How to:

Subtract a Value From a Date

Example:

Subtracting Days From a Date

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 from which to subtract the value, 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):

```
DateVar      SUB(DateVar, 10);
-------      -----------------
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

How to:

Determine the Day of the Week for a Date

Example:

Determining the Day of the Week for a Date

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) for which to determine the weekday, 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):

```
DATE          WEEKDAY(DATE)
----          -------------
01/01/2000    6
01/02/2000    7
01/03/2000    1
```

YEAR: Extracting the Year From a Date

How to:

Extract the Year From a Date

Example:

Extracting a Year From a Date

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 from which to extract the year, 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):

```
DATE           YEAR(DATE)
----           ----------
01/01/2000     2000
02/01/2001     2001
03/01/2002     2002
```

8 Format Conversion Functions

Format conversion functions convert fields from one format to another. For information on field formats see the *Describing Data* manual.

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
- PTOA: Converting a Packed-Decimal Number to Alphanumeric Format
- UFMT: Converting an Alphanumeric String to Hexadecimal

ATODBL: Converting an Alphanumeric String to Double-Precision Format

How to:

Convert an Alphanumeric String to Double-Precision Format

Example:

Converting an Alphanumeric Field to Double-Precision Format

Converting an Alphanumeric Value to Double-Precision Format With MODIFY

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

Double precision floating-point

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 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 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

How to:

Convert the Format of a Field

Example:

Converting From Numeric to Alphanumeric Format

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:

```
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 3, *Character Functions*.

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

How to:

Convert a Number to Alphanumeric Format

Example:

Converting From Numeric to Alphanumeric Format

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 F or D (single and double precision floating-point)

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

How to:

Convert a Decimal Integer to a Character

Example:

Converting a Decimal Integer to a Character

Inserting Braces for Mainframe

The HEXBYT function obtains the ASCII, EBCDIC, or Unicode character equivalent of a decimal integer, depending on your configuration and operating environment. It returns a single alphanumeric character in the ASCII, EBCDIC, or Unicode character set. You can use this function to produce characters that are not on your keyboard, similar to the CTRAN function.

In Unicode configurations, this function uses values in the range:

- ❑ 0 to 255 for 1-byte characters.
- ❑ 256 to 65535 for 2-byte characters.
- ❑ 65536 to 16777215 for 3-byte characters.
- ❑ 16777216 to 4294967295 for 4-byte characters (primarily for EBCDIC).

The display of special characters depends on your software and hardware; not all special characters may appear. For printable ASCII and EBCDIC characters and their integer equivalents see the *Character Chart for ASCII and EBCDIC* in Chapter 1, *Introducing Functions*.

Syntax: How to Convert a Decimal Integer to a Character

```
HEXBYT(input, output)
```

where:

input

Integer

Is the decimal integer to be converted to a single character. In non-Unicode environments, 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
```

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

How to:

Convert a Large Binary Integer to Double-Precision Format

Example:

Converting a Large Binary Integer to Double-Precision Format

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

Double precision floating-point (D*n*)

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 D*n*.

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

How to:

Convert a Large Binary Integer to Packed-Decimal Format

Example:

Converting a Large Binary Integer to Packed-Decimal Format

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 P*n* or P*n*.*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
```

ITOZ: Converting a Number to Zoned Format

How to:

Convert to Zoned Format

Example:

Converting a Number to Zoned Format

The ITOZ 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 Zoned Format

```
ITOZ(outlength, number, outfield)
```

where:

outlength

Integer

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              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

How to:

Write a Packed Number of Variable Length

Example:

Writing a Packed Number of Variable Length

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              5
 HIRE_DATE                          HDT           I6YMD           6

 TOTAL                                                           26
 DCB USED WITH FILE SAVE     IS DCB=(RECFM=FB,LRECL=00026,BLKSIZE=00520)
```

PTOA: Converting a Packed-Decimal Number to Alphanumeric Format

How to:

Convert a Packed-Decimal Number to Alphanumeric Format

Example:

Converting From Packed to Alphanumeric Format

The PTOA function converts a packed-decimal number from numeric format to alphanumeric format. It retains the decimal positions of the number and right-justifies it with leading spaces. You can also add edit options to a number converted by PTOA.

When using PTOA 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 P12.2C format is converted to A14. If the output format is not large enough, the right-most characters are truncated.

Syntax: How to Convert a Packed-Decimal Number to Alphanumeric Format

```
PTOA(number, '(format)', outfield)
```

where:

number

Numeric P (packed-decimal)

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 packed-decimal format is supported. Include any edit options that you want to display in the output.

The format value does not require the same length or number of decimal places as the original field. If you change the number of decimal places, the result is rounded. If you make the length too short to hold the integer portion of the number, asterisks appear instead of the number.

If you use a field name for this argument, specify the name without quotation marks or parentheses. However, parentheses must be included around the format stored in this field. For example:

```
FMT/A10 = '(P12.2C)';
```

You can then use this field as the format argument when using the function in your request:

```
COMPUTE ALPHA_GROSS/A20 = PTOA(PGROSS, FMT, ALPHA_GROSS);
```

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 Packed to Alphanumeric Format

PTOA is called twice to convert the PGROSS field from packed-decimal to alphanumeric format. The format specified in the first call to the function is stored in a virtual field named FMT. The format specified in the second call to the function does not include decimal places, so the value is rounded when it appears:

```
DEFINE FILE EMPLOYEE
PGROSS/P18.2=GROSS;
FMT/A10='(P14.2C)';
END
TABLE FILE EMPLOYEE PRINT PGROSS NOPRINT
COMPUTE AGROSS/A17 = PTOA(PGROSS, FMT, AGROSS); AS ''
COMPUTE BGROSS/A37 = '<- THIS AMOUNT IS' |
                     PTOA(PGROSS, '(P5C)', 'A6') |
                     ' WHEN ROUNDED'; AS '' IN +1
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME NOPRINT
END
```

The output is:

```
2,475.00 <- THIS AMOUNT IS 2,475 WHEN ROUNDED
1,815.00 <- THIS AMOUNT IS 1,815 WHEN ROUNDED
2,255.00 <- THIS AMOUNT IS 2,255 WHEN ROUNDED
  750.00 <- THIS AMOUNT IS   750 WHEN ROUNDED
2,238.50 <- THIS AMOUNT IS 2,239 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,342.00 <- THIS AMOUNT IS 1,342 WHEN ROUNDED
1,760.00 <- THIS AMOUNT IS 1,760 WHEN ROUNDED
1,100.00 <- THIS AMOUNT IS 1,100 WHEN ROUNDED
  791.67 <- THIS AMOUNT IS   792 WHEN ROUNDED
  916.67 <- THIS AMOUNT IS   917 WHEN ROUNDED
```

UFMT: Converting an Alphanumeric String to Hexadecimal

How to:

Convert an Alphanumeric String to Hexadecimal

Example:

Converting an Alphanumeric String to Hexadecimal

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

Integer

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 JOBFILE
HEXCODE/A6 = UFMT(JOBCODE, 3, HEXCODE);
END
TABLE FILE JOBFILE
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
```

9 Numeric Functions

Numeric functions perform calculations on numeric constants and fields.

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
- FMLINFO: Returning FOR Values
- FMLLIST: Returning an FML Tag List
- FMLFOR: Retrieving FML Tag Values
- FMLCAP: Retrieving FML Hierarchy Captions
- INT: Finding the Greatest Integer
- LOG: Calculating the Natural Logarithm
- MAX and MIN: Finding the Maximum or Minimum Value
- MIRR: Calculating the Modified Internal Return Rate
- NORMSDST and NORMSINV: Calculating Cumulative Normal Distribution
- PRDNOR and PRDUNI: Generating Reproducible Random Numbers
- RDNORM and RDUNIF: Generating Random Numbers
- RDNORM and RDUNIF: Generating Random Numbers
- SQRT: Calculating the Square Root
- XIRR: Calculating the Modified Internal Return Rate (Periodic or Non-Periodic)

ABS: Calculating Absolute Value

How to:

Calculate Absolute Value

Example:

Calculating Absolute Value

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

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 3, *Character Functions*.

BAR: Producing a Bar Chart

How to:

Produce a Bar Chart

Example:

Producing a Bar Chart

Creating a Bar Chart With a Scale

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 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.

'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

How to:

Validate a Packed Field

Example:

Validating Packed Data

The CHKPCK 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 CHKPCK:

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 CHKPCK 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-decimal

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 example 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 output is:

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


 [EBCDIC|ALPHANUMERIC]  RECORD NAMED  TESTPACK
 FIELDNAME                         ALIAS         FORMAT        LENGTH

 EMP_ID                            EID           A9              9
 DEPARTMENT                        DPT           A10            10
 PACK_SAL                                        A8              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 appear 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

How to:

Calculate the Remainder From a Division

Example:

Calculating the Remainder From a Division

The MOD functions calculate the remainder from a division. Each function returns the remainder in a different format.

The functions use the following formula.

```
remainder = dividend - INT(dividend/divisor) * 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 344.

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

How to:

Raise "e" to the Nth Power

Example:

Raising "e" to the Nth Power

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 to which "e" is raised.

outfield

Double-precision floating-point

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
```

FMLINFO: Returning FOR Values

How to:

Retain FOR Values in an FML Request

Example:

Retrieving FOR Values for FML Hierarchy Rows

Using FMLINFO With an OR Phrase

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: You can use the SET parameter FORMULTIPLE=ON to enable 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: 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 in 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
```

FMLLIST: Returning an FML Tag List

How to:

Retrieve an FML Tag List

Example:

Retrieving an FML Tag List With FMLLIST

FMLLIST returns a string containing the complete tag list for each row in an FML request. If a row has a single tag value, that value is returned.

The FMLLIST function is supported for COMPUTE but not for DEFINE. Attempts to use it in a DEFINE result in blank values.

Syntax: How to Retrieve an FML Tag List

```
FMLLIST('A4096V')
```

where:

`'A4096V'`

Is the required argument.

Example: Retrieving an FML Tag List With FMLLIST

```
SET FORMULTIPLE=ON
TABLE FILE LEDGER
HEADING
"TEST OF FMLLIST"
" "
SUM AMOUNT
COMPUTE LIST1/A36 = FMLLIST('A4096V');
FOR ACCOUNT
'1010'                 OVER
'1020'                 OVER
'1030'                 OVER
BAR                    OVER
'1030' OR '1020' OR '1010'
END
```

The output is:

```
TEST OF FMLLIST

      AMOUNT  LIST1
      ------  -----
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  ------------------------------------
1010  21,239  1010 OR 1020 OR 1030
```

FMLFOR: Retrieving FML Tag Values

How to:

Retrieve FML Tag Values

Example:

Retrieving FML Tag Values With FMLFOR

FMLFOR retrieves the tag value associated with each row in an FML request. If the FML row was generated as a sum of data records using the OR phrase, FMLFOR returns the first value specified in the list. If the OR phrase was generated by an FML Hierarchy ADD command, FMLFOR returns the tag value associated with the parent specified in the ADD command.

The FMLFOR function is supported for COMPUTE but not for DEFINE. Attempts to use it in a DEFINE result in blank values.

Syntax: How to Retrieve FML Tag Values

```
FMLFOR(outfield)
```

where:

outfield

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

Example: Retrieving FML Tag Values With FMLFOR

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLFOR('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
```

FMLCAP: Retrieving FML Hierarchy Captions

How to:

Retrieve Captions in an FML Request Using the FMLCAP Function

Example:

Retrieving FML Hierarchy Captions Using FMLCAP

The FMLCAP function returns the caption value for each row in an FML hierarchy request. In order to retrieve caption values, the Master File must define an FML hierarchy and the request must use the GET CHILDREN, ADD, or WITH CHILDREN option to retrieve hierarchy data. If the FOR field in the request does not have a caption field defined, FMLCAP returns a blank string.

FMLCAP is supported for COMPUTE but is not recommended for use with DEFINE.

Syntax: How to Retrieve Captions in an FML Request Using the FMLCAP Function

```
FMLCAP(fieldname|'format')
```

where:

fieldname

Is the name of the caption field.

'format'

Is the format of the caption field enclosed in single quotation marks.

Example: Retrieving FML Hierarchy Captions Using FMLCAP

The following request retrieves and aggregates the FML hierarchy that starts with the parent value 2000. FMLCAP retrieves the captions, while the actual account numbers appear as the FOR values.

```
SET FORMULTIPLE = ON
TABLE FILE CENTSTMT
SUM ACTUAL_AMT
COMPUTE CAP1/A30= FMLCAP(GL_ACCOUNT_CAPTION);
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD
END
```

The output is:

```
                Actual  CAP1
                ------  ----
2000      313,611,852.  Gross Margin
  2100    187,087,470.  Sales Revenue
    2200   98,710,368.  Retail Sales
    2300   13,798,832.  Mail Order Sales
    2400   12,215,780.  Internet Sales
  2500    100,885,159.  Cost Of Goods Sold
    2600   54,877,250.  Variable Material Costs
    2700    6,176,900.  Direct Labor
    2800    3,107,742.  Fixed Costs
```

INT: Finding the Greatest Integer

How to:

Find the Greatest Integer

Example:

Finding the Greatest Integer

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

How to:

Calculate the Natural Logarithm

Example:

Calculating the Natural Logarithm

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

How to:

Find the Maximum or Minimum Value

Example:

Determining the Minimum Value

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 for 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
```

MIRR: Calculating the Modified Internal Return Rate

How to:

Calculate the Modified Internal Rate of Return

Example:

Calculating the Modified Internal Rate of Return

Reference:

Usage Notes for the MIRR Function

The MIRR function calculates the modified internal rate of return for a series of periodic cash flows.

Syntax: How to Calculate the Modified Internal Rate of Return

```
TABLE FILE ...
{PRINT|SUM} field ...
COMPUTE rrate/fmt = MIRR(cashflow, finrate, reinvrate, output);
WITHIN {sort_field|TABLE}
```

where:

field ...

Are fields that appear in the report output.

rrate

Is the field that contains the calculated return rate.

fmt

Is the format of the return rate. The data type must be D.

`cashflow`

Is a numeric field. Each value represents either a payment (negative value) or income (positive value) for one period. The values must be in the correct sequence in order for for the sequence of cash flows to be calculated correctly. The dates corresponding to each cash flow should be equally spaced and sorted in chronological order. The calculation requires at least one negative value and one positive value in the *cashflow* field. If the values are all positive or all negative, a zero result is returned.

`finrate`

Is a finance rate for negative cash flows. This value must be expressed as a non-negative decimal fraction between 0 and 1. It must be constant within each sort group for which a return rate is calculated, but it can change between sort groups.

`reinvrate`

Is the reinvestment rate for positive cash flows. This value must be expressed as a non-negative decimal fraction between 0 and 1. It must be constant within each sort group but can change between sort groups. It must be constant within each sort group for which a return rate is calculated, but it can change between sort groups.

`output`

Is the name of the field that contains the return rate, or its format enclosed in single quotation marks.

`sort_field`

Is a field that sorts the report output and groups it into subsets of rows on which the function can be calculated separately. To calculate the function using every row of the report output, use the WITHIN TABLE phrase. A WITHIN phrase is required.

Reference: Usage Notes for the MIRR Function

- This function is only supported in a COMPUTE command with the WITHIN phrase.
- The cash flow field must contain at least one negative value and one positive value.
- Dates must be equally spaced.
- Missing cash flows or dates are not supported.

Example: Calculating the Modified Internal Rate of Return

The following request calculates modified internal return rates for categories of products. It assumes a finance charge of ten percent and a reinvestment rate of ten percent. The request is sorted by date so that the correct cash flows are calculated. The rate returned by the function is multiplied by 100 in order to express it as a percent rather than a decimal value. Note that the format includes the % character. This causes a percent symbol to display, but it does not calculate a percent.

In order to create one cash flow value per date, the values are summed. NEWDOLL is defined in order to create negative values in each category as required by the function:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END

TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN CATEGORY
  BY CATEGORY
  BY SDATE
  WHERE SYEAR EQ 97
END
```

A separate rate is calculated for each category because of the WITHIN CATEGORY phrase. A portion of the output is shown:

```
Category      SDATE             NEWDOLL       RRATE
--------      -----             -------       -----
Coffee        1997/01       -801,123.00      15.11%
              1997/02       -682,340.00      15.11%
              1997/03       -765,078.00      15.11%
              1997/04        691,274.00      15.11%
              1997/05        720,444.00      15.11%
              1997/06        742,457.00      15.11%
              1997/07        747,253.00      15.11%
              1997/08        655,896.00      15.11%
              1997/09        730,317.00      15.11%
              1997/10        724,412.00      15.11%
              1997/11        620,264.00      15.11%
              1997/12        762,328.00      15.11%
Food          1997/01       -672,727.00      16.24%
              1997/02       -699,073.00      16.24%
              1997/03       -642,802.00      16.24%
              1997/04        718,514.00      16.24%
              1997/05        660,740.00      16.24%
              1997/06        734,705.00      16.24%
              1997/07        760,586.00      16.24%
```

To calculate one modified internal return rate for all of the report data, use the WITHIN TABLE phrase. In this case, the data does not have to be sorted by CATEGORY:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END

TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN TABLE
  BY SDATE
  WHERE SYEAR EQ 97
END
```

The output is:

```
SDATE              NEWDOLL       RRATE
-----              -------       -----
1997/01      -1,864,129.00      15.92%
1997/02      -1,861,639.00      15.92%
1997/03      -1,874,439.00      15.92%
1997/04       1,829,838.00      15.92%
1997/05       1,899,494.00      15.92%
1997/06       1,932,630.00      15.92%
1997/07       2,005,402.00      15.92%
1997/08       1,838,863.00      15.92%
1997/09       1,893,944.00      15.92%
1997/10       1,933,705.00      15.92%
1997/11       1,865,982.00      15.92%
1997/12       2,053,923.00      15.92%
```

NORMSDST and NORMSINV: Calculating Cumulative Normal Distribution

How to:

Calculate the Cumulative Standard Normal Distribution Function

Calculate the Inverse Cumulative Standard Normal Distribution Function

Example:

Using the NORMSDST and NORMSINV Functions

Reference:

Characteristics of the Normal Distribution

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 a normalized value. Assuming that your data is normally distributed, you can convert a data point to a normalized value 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 = (raw_score - mean)/standard_deviation
```

To convert from a z-score back to a raw score, use the following formula:

```
raw_score = z * standard_deviation + mean
```

The mean of data points x_i, where i is from 1 to n is:

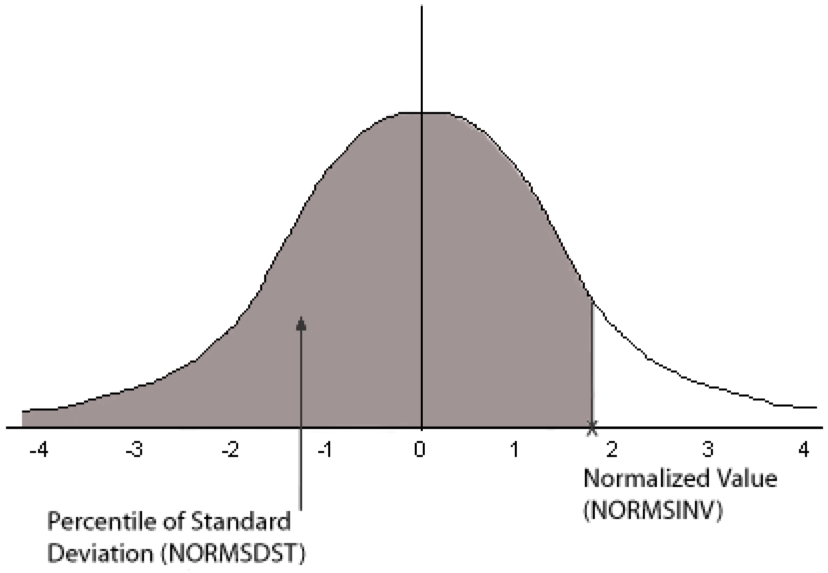
$$(\sum x_i)/n$$

The standard deviation of data points x_i, where i is from 1 to n is:

```
SQRT( ((∑xi2 - (∑xi)2/n)/(n - 1)))
```

$$\mathrm{SQRT}(\ ((\sum x_i^2 - (\sum x_i)^2/n)/(n - 1)))$$

The following diagram illustrates the results of the NORMSDST and NORMSINV functions:



Reference: Characteristics of the Normal Distribution

Many common measurements are normally distributed. A plot of normally distributed data values approximates a bell-shaped curve. The two measures required 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 (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
  16   -.07298     .47091       -.07298        16
  12   -.80273     .21106       -.80273        12
  12   -.80273     .21106       -.80273        12
  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

How to:

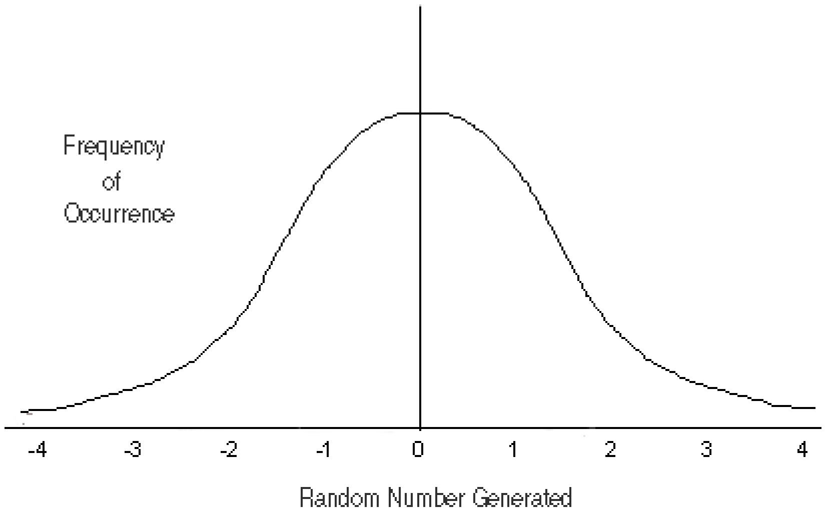
Generate Reproducible Random Numbers

Example:

Generating Reproducible Random Numbers

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).

CMS behavior differs from z/OS behavior. In CMS, the seed changes upon multiple executions as the function is reloaded unless you reissue the DEFINE for each execution. In z/OS, 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

How to:

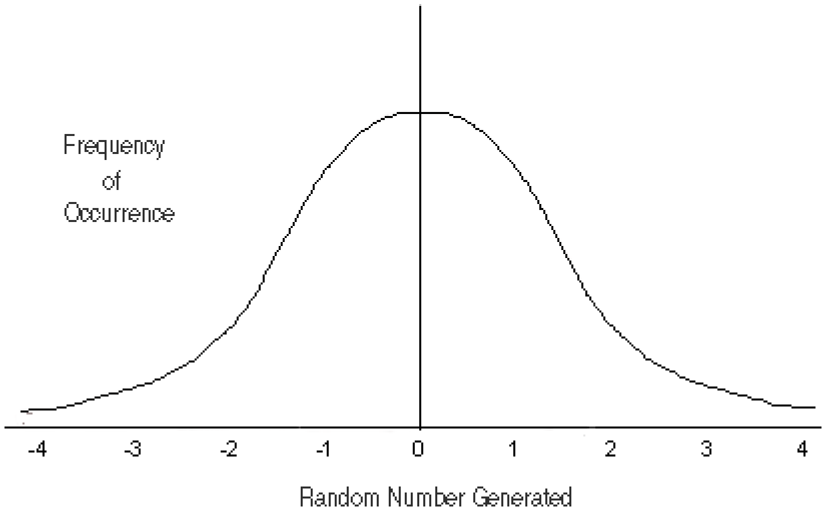
Generate Random Numbers

Example:

Generating Random Numbers

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

How to:

Calculate the Square Root

Example:

Calculating the Square Root

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
```

XIRR: Calculating the Modified Internal Return Rate (Periodic or Non-Periodic)

How to:

Calculate the Internal Rate of Return

Example:

Calculating the Internal Rate of Return

Reference:

Usage Notes for the XIRR Function

The XIRR function calculates the internal rate of return for a series of cash flows that can be periodic or non-periodic.

Syntax: How to Calculate the Internal Rate of Return

```
TABLE FILE ...
{PRINT|SUM} field ...
COMPUTE rrate/fmt = XIRR (cashflow, dates, guess, maxiterations, output);
WITHIN {sort_field|TABLE}
```

where:

field ...

Are fields that appear in the report output.

rrate

Is the field that contains the calculated return rate.

fmt

Is the format of the return rate. The data type must be D.

cashflow

Is a numeric field. Each value of this field represents either a payment (negative value) or income (positive value) for one period. The values must be in the correct sequence in order for the sequence of cash flows to be calculated correctly. The dates corresponding to each cash flow should be equally spaced and sorted in chronological order. The calculation requires at least one negative value and one positive value in the *cashflow* field. If the values are all positive or all negative, a zero result is returned.

dates

Is a date field containing the cash flow dates. The dates must be full component dates with year, month, and day components. Dates cannot be stored in fields with format A, I, or P. They must be stored in date fields (for example, format YMD, not AYMD). There must be the same number of dates as there are cash flow values. The number of dates must be the same as the number of cash flows.

guess

Is an (optional) initial estimate of the expected return rate expressed as a decimal. The default value is .1 (10%). To accept the default, supply the value 0 (zero) for this argument.

maxiterations

Is an (optional) number specifying the maximum number of iterations that can be used to resolve the rate using Newton's method. 50 is the default value. To accept the default, supply the value 0 (zero) for this argument. The rate is considered to be resolved when successive iterations do not differ by more than 0.0000003. If this level of accuracy is achieved within the maximum number of iterations, calculation stops at that point. If it is not achieved after reaching the maximum number of iterations, calculation stops and the value calculated by the last iteration is returned.

output

D

Is the name of the field that contains the return rate, or its format enclosed in single quotation marks.

sort_field

Is a field that sorts the report output and groups it into subsets of rows on which the function can be calculated separately. To calculate the function using every row of the report output, use the WITHIN TABLE phrase. A WITHIN phrase is required.

Reference: Usage Notes for the XIRR Function

- ❑ This function is only supported in a COMPUTE command with the WITHIN phrase.
- ❑ The cash flow field must contain at least one negative value and one positive value.
- ❑ Dates cannot be stored in fields with format A, I, or P. They must be stored in date fields (for example, format YMD, not AYMD).
- ❑ Cash flows or dates with missing values are not supported.

Example: Calculating the Internal Rate of Return

The following request creates a FOCUS data source with cash flows and dates and calculates the internal return rate.

The Master File for the data source is:

```
FILENAME=XIRR01,SUFFIX=FOC
SEGNAME=SEG1,SEGTYPE=S1
FIELDNAME=DUMMY,FORMAT=A2,$
FIELDNAME=DATES,FORMAT=YYMD,$
FIELDNAME=CASHFL,FORMAT=D12.4,$
END
```

The procedure to create the data source is:

```
CREATE FILE XIRR01
MODIFY FILE XIRR01
FREEFORM DUMMY DATES CASHFL
DATA
AA,19980101,-10000. ,$
BB,19980301,2750.   ,$
CC,19981030,4250.   ,$
DD,19990215,3250.   ,$
EE,19990401,2750.   ,$
END
```

The request is sorted by date so that the correct cash flows can be calculated. The rate returned by the function is multiplied by 100 in order to express it as a percent rather than a decimal value. Note that the format includes the % character. This causes a percent symbol to display, but it does not calculate a percent:

```
TABLE FILE XIRR01
PRINT CASHFL
COMPUTE RATEX/D12.2%=XIRR(CASHFL, DATES, 0., 0., RATEX) * 100;
WITHIN TABLE
BY DATES
END
```

One rate is calculated for the entire report because of the WITHIN TABLE phrase:

```
DATES                CASHFL        RATEX
-----                ------        -----
1998/01/01    -10,000.0000        37.49%
1998/03/01      2,750.0000        37.49%
1998/10/30      4,250.0000        37.49%
1999/02/15      3,250.0000        37.49%
1999/04/01      2,750.0000        37.49%
```

10 System Functions

System functions call the operating system to obtain information about the operating environment or to use a system service.

Topics:

- ❏ FEXERR: Retrieving an Error Message
- ❏ FINDMEM: Finding a Member of a Partitioned Data Set
- ❏ 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
- ❏ SLEEP: Suspending Execution for a Given Number of Seconds
- ❏ TODAY: Returning the Current Date

FEXERR: Retrieving an Error Message

How to:

Retrieve an Error Message

Example:

Retrieving an Error Message

The FEXERR function retrieves an Information Builders error message. It is especially useful in a prcedure 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
```

FINDMEM: Finding a Member of a Partitioned Data Set

How to:

Find a Member of a Partitioned Data Set

Example:

Finding a Member of a Partitioned Data Set

Available Operating Systems: z/OS

The FINDMEM function, used on z/OS, 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 eight-character literal enclosed in single quotation marks, or a variable that contains the ddname. If you supply a literal less than eight characters long, pad it with trailing spaces.

member

A8

Is the member for which you are searching. This value must be eight characters long. If you supply a literal that has less than eight 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 RECORDLIMIT EQ 4;
END
-EXIT
-NOMEM
-TYPE EMPLOYEE NOT FOUND IN MASTER FILE PDS
-EXIT
-NOPDS
-TYPE ERROR OCCURRED IN SEARCH
-TYPE CHECK IF FILE IS A PDS ALLOCATED TO DDNAME MASTER
-EXIT
```

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
```

GETPDS: Determining If a Member of a Partitioned Data Set Exists

How to:

Determine If a PDS Member Exists

Example:

Determining If a PDS Member Exists

Copying a Member for Editing in TED

Displaying the Attributes of a PDS

Available Operating Systems: z/OS

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 eight-character literal enclosed in single quotation marks, or a variable that contains the ddname. If you supply a literal less than eight characters long, pad it with trailing spaces.

member

A8

Is the member for which the function searches. This value must be eight characters long. If you supply a literal with less than eight 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
```

The output is similar to the following:

```
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.

```
-* 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
```

The sample output is:

```
THE DATA SET ATTRIBUTES INCLUDE:
DATA SET NAME IS: USER1.MASTER.DATA
VOLUME IS: USERM0
DISPOSITION IS: SHR
```

GETUSER: Retrieving a User ID

How to:

Retrieve a User ID

Example:

Retrieving a User ID

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

Alphanumeric, at least A8

Is the result field, whose length depends on the platform on which the function is issued. Provide a length as long as required for your platform; otherwise the output will be truncated.

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

The HHMMSS function retrieves the current time from the operating system as an eight-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 6, *Date and Time Functions*. For details on how to use HHMMSS in Maintain, see *HHMMSS: Retrieving the Current Time (Maintain)* in Chapter 7, *Maintain-specific Date and Time Functions*.

MVSDYNAM: Passing a DYNAM Command to the Command Processor

How to:

Pass a DYNAM Command to the Command Processor

Example:

Passing a DYNAM Command to the Command Processor

Available Operating Systems: z/OS

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
>
```

SLEEP: Suspending Execution for a Given Number of Seconds

How to:

Suspend Execution for a Specified Number of Seconds

Example:

Suspending Execution for Four Seconds

The SLEEP function suspends execution for the number of seconds you specify as its input argument.

This function is most useful in Dialogue Manager when you need to wait to start a specific procedure. For example, you can start a FOCUS Database Server and wait until the server is started before initiating a client application.

Syntax: How to Suspend Execution for a Specified Number of Seconds

```
SLEEP(delay, outfld);
```

where:

delay

Numeric

Is the number of seconds to delay execution. The number can be specified down to the millisecond.

outfld

Numeric

Is the name of a field or a format enclosed in single quotation marks. The value returned is the same value you specify for *delay*.

Example: Suspending Execution for Four Seconds

The following example computes the current date and time, suspends execution for 4 seconds, and computes the current date and time after the delay:

```
TABLE  FILE VIDEOTRK
PRINT TRANSDATE NOPRINT
COMPUTE
START_TIME/HYYMDSa = HGETC(8, START_TIME);
DELAY/I2 = SLEEP(4.0, 'I2');
END_TIME/HYYMDSa = HGETC(8, END_TIME);
IF RECORDLIMIT EQ 1
END
```

The output is:

```
START_TIME              DELAY  END_TIME
----------              -----  --------
2007/10/26  5:04:36pm       4  2007/10/26  5:04:40pm
```

TODAY: Returning the Current Date

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 6, *Date and Time Functions*. For details on using TODAY in Maintain, see *TODAY: Retrieving the Current Date (Maintain)* in Chapter 7, *Maintain-specific Date and Time Functions*.

11 Input/Output Functions

Input/Output functions use your operating environment's input/output routines to open, close, and write records to sequential files. These functions are not available for FOCUS Maintain.

Topics:

❏ PUTDDREC and CLSDDREC: Opening, Closing, and Writing Files

PUTDDREC and CLSDDREC: Opening, Closing, and Writing Files

How to:

Write a Character String as a Record in a Sequential File

Close All Files Opened by the PUTDDREC Function

Example:

Calling PUTDDREC in a TABLE Request

Calling PUTDDREC and CLSDDREC in Dialogue Manager -SET Commands

Reference:

Usage Notes for the PUTDDREC and CLSDDREC Functions

The PUTDDREC function writes a character string as a record in a flat file. The file must be identified with a CMS FILEDEF command on z/VM (DYNAM on z/OS. TSO ALLOCATE does not work). If the file is defined with the APPEND option, the new record is appended. Without the APPEND option, the new record overwrites any existing file. For information about the FILEDEF command, see the *Overview and Operating Environments* manual.

If the file is not already open, PUTDDREC opens it the first time it is called. Each call to PUTDDREC can use the same file or open a new one. All of the files opened by PUTDDREC remain open until the end of a request or session. At the end of the request or session, all files opened by PUTDDREC are closed automatically.

If PUTDDREC is called in a Dialogue Manager -SET command, the files opened by PUTDDREC are not closed automatically until the end of a request or session. In this case, you can manually close the files and free the memory used to store information about open files by calling the CLSDDREC function.

Syntax: How to Write a Character String as a Record in a Sequential File

`PUTDDREC(ddname, dd_len, record_string, record_len, outfield)`

where:

ddname

Alphanumeric

Is the logical name assigned to the sequential file in a CMS FILEDEF or DYNAM (on z/OS) command. For information about the FILEDEF command, see the *Overview and Operating Environments* manual.

dd_len

Numeric

Is the number of characters in the logical name.

record_string

Alphanumeric

Is the character string to be added as the new record in the sequential file.

record_len

Numeric

Is the number of characters to add as the new record. It cannot be larger than the number of characters in *record_string*. To write all of *record_string* to the file, *record_len* should equal the number of characters in *record_string* and should not exceed the record length declared in the FILEDEF command. If *record_len* is shorter than the length declared in the FILEDEF command, the resulting file may contain extraneous characters at the end of each record. If *record_string* is longer than the length declared in the FILEDEF command, *record_string* may be truncated in the resulting file.

outfield

Integer

Is the return code, which can have one of the following values:

Return Code	Description
0	Record is added.
-1	FILEDEF statement is not found.
-2	Error while opening the file.
-3	Error while adding the record to the file.

Syntax: How to Close All Files Opened by the PUTDDREC Function

```
CLSDDREC(outfield)
```

where:

outfield

Integer

Is the return code, which can be one of the following values:

Return Code	Description
0	Files are closed.
-1	Error while closing the files.

Reference: Usage Notes for the PUTDDREC and CLSDDREC Functions

- The open, close, and write operations are handled by the operating system. Therefore, the requirements for writing to the file and the results of deviating from the instructions when calling PUTDDREC are specific to your operating environment. Make sure you are familiar with and follow the guidelines for your operating system when performing input/output operations.
- You can call PUTDDREC in a DEFINE FILE command or in a DEFINE in the Master File. However, PUTDDREC does not open the file until its field name is referenced in a request.

Example: Calling PUTDDREC in a TABLE Request

The following example defines a new file whose logical name is PUTDD1. The TABLE request then calls PUTDDREC for each employee in the EMPLOYEE data source and writes a record to the file composed of the employee's last name, first name, employee ID, current job code, and current salary (converted to alphanumeric using the EDIT function). The return code of zero (in OUT1) indicates that the calls to PUTDDREC were successful:

```
CMS FILEDEF PUTDD1 DISK PUTDD1 DATA A
TABLE FILE EMPLOYEE
PRINT EMP_ID CURR_JOBCODE AS 'JOB' CURR_SAL
COMPUTE SALA/A12 = EDIT(CURR_SAL); NOPRINT
COMPUTE EMP1/A50= LAST_NAME|FIRST_NAME|EMP_ID|CURR_JOBCODE|SALA;
NOPRINT
COMPUTE OUT1/I1 = PUTDDREC('PUTDD1',6, EMP1, 50, OUT1);
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME        FIRST_NAME  EMP_ID     JOB          CURR_SAL  OUT1
---------        ----------  ------     ---          --------  ----
BANNING          JOHN        119329144  A17        $29,700.00     0
BLACKWOOD        ROSEMARIE   326179357  B04        $21,780.00     0
CROSS            BARBARA     818692173  A17        $27,062.00     0
GREENSPAN        MARY        543729165  A07         $9,000.00     0
IRVING           JOAN        123764317  A15        $26,862.00     0
JONES            DIANE       117593129  B03        $18,480.00     0
MCCOY            JOHN        219984371  B02        $18,480.00     0
MCKNIGHT         ROGER       451123478  B02        $16,100.00     0
ROMANS           ANTHONY     126724188  B04        $21,120.00     0
SMITH            MARY        112847612  B14        $13,200.00     0
                 RICHARD     119265415  A01         $9,500.00     0
STEVENS          ALFRED      071382660  A07        $11,000.00     0
```

After running this request, the sequential file contains the following records:

```
BANNING        JOHN      119329144A17000000029700
BLACKWOOD      ROSEMARIE 326179357B04000000021780
CROSS          BARBARA   818692173A17000000027062
GREENSPAN      MARY      543729165A07000000009000
IRVING         JOAN      123764317A15000000026862
JONES          DIANE     117593129B03000000018480
MCCOY          JOHN      219984371B02000000018480
MCKNIGHT       ROGER     451123478B02000000016100
ROMANS         ANTHONY   126724188B04000000021120
SMITH          MARY      112847612B14000000013200
SMITH          RICHARD   119265415A01000000009500
STEVENS        ALFRED    071382660A07000000011000
```

Example: Calling PUTDDREC and CLSDDREC in Dialogue Manager -SET Commands

The following example defines a new file whose logical name is PUTDD1. The first -SET command creates a record to add to this file. The second -SET command calls PUTDDREC to add the record. The last -SET command calls CLSDDREC to close the file. The return codes are displayed to make sure operations were successful:

```
CMS FILEDEF PUTDD1 DISK PUTDD1 DATA A
-SET &EMP1 = 'SMITH'|'MARY'|'A07'|'27000';
-TYPE DATA = &EMP1
-SET &OUT1 = PUTDDREC('PUTDD1',6, &EMP1, 17, 'I1');
-TYPE PUT RESULT = &OUT1
-SET &OUT1 = CLSDDREC('I1');
-TYPE CLOSE RESULT = &OUT1
```

The output is:

```
DATA = SMITHMARYA0727000
PUT RESULT = 0
CLOSE RESULT = 0
```

After running this procedure, the sequential file contains the following record:

```
SMITHMARYA0727000
```

A Creating a Subroutine

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 388.
- ❏ Compiling the subroutine. For details, see *Compiling and Storing a Subroutine* on page 401.
- ❏ Storing the subroutine in a separate file; do not include it in the main program. For details, *Compiling and Storing a Subroutine* on page 401.
- ❏ Testing the subroutine. For details, see *Testing the Subroutine* on page 403.

Note: All subroutines called by FOCUS must be fully LE compliant.

Topics:

- ❏ Writing a Subroutine
- ❏ Compiling and Storing a Subroutine
- ❏ Testing the Subroutine
- ❏ Using a Custom Subroutine: The MTHNAM Subroutine
- ❏ Subroutines Written in REXX

Writing a Subroutine

In this section:

Naming a Subroutine

Creating Arguments

Language Considerations

Programming 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 389.
- ❏ Argument considerations. For details, see *Creating Arguments* on page 389.
- ❏ Language considerations. For details, see *Language Considerations* on page 392.
- ❏ Programming considerations. For details, see *Programming a Subroutine* on page 395.

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 398.
- **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 2, *Accessing and Calling a Function*.
- **Input arguments.** Input arguments are passed to a subroutine using standard conventions. Register one points to the list of argument addresses. Each address is a full word.

  The input parameter values should not be changed in the subroutine code. You should not assume that the input parameters are stored in contiguous memory. The same parameter cannot be used for both input and output.

- **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 2, *Accessing and Calling a Function*.

The lengths of the calling arguments as defined in FOCUS must match the lengths of the corresponding arguments defined in the subroutine.

Any deviation from these rules may result in problems in using the subroutine. In some cases, linking the subroutine below the line may resolve these issues, but at the cost of a reduction in efficiency. Information Builders recommends that you modify the subroutine to conform to the stated rules and then link it above the line. In order to load subroutines above the line, the following are the recommended link-edit options for compiling and storing the subroutine:

- AMODE 31 (Addressing Mode - 31-bit addressing)
- RMODE ANY (System can load this routine anywhere)

MSO Considerations

If a subroutine must run below the line in the MSO environment, the FOCMSO configuration file must contain the following parameter in the SERVICE block:

```
LE_STACK=BELOW
```

z/VM Considerations With LE

In this section:

Creating a LOADLIB

Generating a Module

Linking a Subroutine

Calling a Subroutine

To create a subroutine, you can create a LOADLIB containing the subroutine or generate a module containing the subroutine.

Creating a LOADLIB

To create a subroutine with LE, run the GENSUBLL EXEC to create the subroutine module as follows:

```
GENSUBLL libname textfiles ([lked_parms] {PLI|COBOL}
```

where:

libname

Is the name of the load library that will contain the subroutine.

textfiles

Are the names of the object modules to include in the library separated by spaces.

lked_parms

Are any link edit parameters required.

PLI|COBOL

Is the type of subroutine being created: PLI or COBOL. Note that if you specify COBOL, you must have an LE-compliant version of COBOL and you need the file LECOBOPT TEXT to be accessible. This file is unloaded from the FOCUS distribution cartridge as part of the FOCUS installation procedure.

Generating a Module

If you prefer not to create a LOADLIB, issue the following commands:

```
LOAD textfiles
GENMOD programname
```

where:

`textfiles`

Are the names of the object modules separated by spaces.

`programname`

Is the name of the module that will contain the subroutine.

However, keep in mind that this method does not support multiple entry points.

Linking a Subroutine

To link a subroutine with LE, you must issue the following command before linking:

```
GLOBAL TXTLIB SCEELKED
```

Calling a Subroutine

To call a subroutine with LE, you must issue the following command at runtime:

```
GLOBAL LOADLIB SCEERUN
```

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 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, z/OS supports FORTRAN input/output operations.

COBOL. When writing a subroutine in COBOL:

- ❏ The subroutine must use the GOBACK command to return to the calling program. STOPRUN is not supported.
- ❏ Numeric arguments received from a request must be declared as COMP-2 (double precision floating point).

- The format described in the DEFINE or COMPUTE command determines the format of the output argument:

FOCUS Format	Picture
`A`*n*	`X(`*n*`)`
`I`	`S9(9) COMP`
`P`	`S9(`*m*`)[V9(`*m*`)]` where: `(1+`*n*`+`*m*`)/2 = 8` for small packed numbers. `(1+`*n*`+`*m*`)/2 = 16` for large packed numbers.
`D`	`COMP-2`
`F`	`COMP-1`

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**
`A`*`n`*	`CHARACTER (`*`n`*`)`
`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**
`A`*`n`*	`char *`*`xxx n`* 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

In this section:

Executing a Subroutine at an Entry Point

Including More Than 28 Arguments in a Subroutine Call

How to:

Execute a Subroutine at an Entry Point

Create a Subroutine With Multiple Call Statements

Example:

Executing a Subroutine at an Entry Point

Creating a Subroutine With 32 Arguments

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. However, it cannot be used for both input and output.

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 396.

❏ 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 398.

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 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 396.

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

In this section:

Compiling and Storing a Subroutine on CMS

Compiling and Storing a Subroutine on z/OS

How to:

Query the Location of a Compiled Object

Reference:

Storing a Subroutine in a Text File or Library

After you write a subroutine, you need to compile and store it. This topic discusses compiling and storing your subroutine for CMS, and z/OS.

Compiling and Storing a Subroutine on 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 FOCUS. 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 396.

There are two 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 z/OS

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.

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 CMS or the ALLOCATE command for z/OS.

If an error occurs during testing, check to see if the error is in the request or in the subroutine.

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

In this section:

Writing the MTHNAM Subroutine

Calling the MTHNAM Subroutine From a Request

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

Reference:

MTHNAM Subroutine Written in FORTRAN

MTHNAM Subroutine Written in COBOL

MTHNAM Subroutine Written in PL/I

MTHNAM Subroutine Written in BAL Assembler

MTHNAM Subroutine Written in C

Example:

Compiling the FORTRAN Version of MTHNAM Under LE

Compiling the COBOL Version of MTHNAM Under LE

Compiling the PL/I Version of MTHNAM Under LE

Compiling the Assembler Version of MTHNAM Under LE

Compiling the C Version of MTHNAM Under LE

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
```

Example: Compiling the FORTRAN Version of MTHNAM Under LE

The following example compiles and linkedits the FORTRAN version of MTHNAM:

```
//COMPILE   EXEC  PGM=FORTVS2,
//          PARM='LANGLVL(66),NODECK,NOLIST,OPT(0)'
//*         PARM='NODECK,NOLIST,OPT(0)'
//STEPLIB   DD  DSN=VSF2.VSF2COMP,DISP=SHR
//SYSLIB    DD  DSN=CEE.SCEESAMP,DISP=SHR
//SYSPRINT  DD  SYSOUT=*,DCB=BLKSIZE=3429
//SYSTERM   DD  SYSOUT=*
//SYSPUNCH  DD  SYSOUT=B,DCB=BLKSIZE=3440
//SYSLIN    DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSD
//          SPACE=(3200,(25,6)),DCB=BLKSIZE=3200
//SYSIN     DD  *
      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
//IEBGENER  EXEC  PGM=IEBGENER,
//          COND=(0,NE)
//SYSUT1    DD  DSN=&&LOADSET,DISP=(OLD,PASS)
//SYSUT2    DD  SYSOUT=*
//SYSPRINT  DD  DUMMY
//SYSIN     DD  DUMMY
//*
//LINKEDIT  EXEC  PGM=HEWL,
//          PARM='MAP,LIST,XREF,SIZE=(500K,65K),RMODE(24),AMODE(24)',
//          COND=(0,NE)
//SYSPRINT  DD  SYSOUT=*
```

```
//SYSLIB     DD  DSN=CEE.SAFHFORT,DISP=SHR
//           DD  DSN=CEE.SCEELKED,DISP=SHR
//SCEESAMP   DD  DSN=CEE.SCEESAMP,DISP=SHR
//SYSUT1     DD  UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSLMOD    DD  DSN=prefix.TSO.LOAD,DISP=SHR
//OBJECT     DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//*          DD  DDNAME=SYSIN
//SYSLIN     DD  *
  INCLUDE OBJECT
  INCLUDE SYSLIB(CEESG007)
  NAME    MTHNAM(R)
/*
//*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

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.
```

Example: Compiling the COBOL Version of MTHNAM Under LE

The following example compiles and linkedits the COBOL version of MTHNAM:

```
//COMPILE   EXEC  PGM=IGYCRCTL,
//          PARM='APOST,RES,RENT'
//STEPLIB   DD  DSN=SYS1.COB2COMP,DISP=SHR
//          DD  DSN=SYS1.COB2LIB,DISP=SHR
//SYSPRINT  DD  SYSOUT=*
//SYSLIN    DD  DSNAME=&&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),
//          SPACE=(TRK,(3,3))
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN     DD  *
      *
      * THE ORIGINAL COBOL PROGRAM HAS BEEN MODIFIED. 'DISPLAYS' HAVE
      * BEEN ADDED TO ILLUSTRATE THE DIFFERENCE BETWEEN EXECUTION IN A
      * 'REUSABLE RUNTIME ENVIRONMENT' AND A 'NON-REUSABLE' ENVIRONMENT.
      *
       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.
          01  WHEN-COMPILED-LINE.
              05  FILLER                 PIC X(32)   VALUE
                  'MTHNAM   - LAST COMPILATION WAS '.
              05  WHEN-COMPILED-FLD      PIC X(20)   VALUE SPACE.
                  88 INITIAL-ENTRY                   VALUE SPACE.
        LINKAGE SECTION.
          01  MTH     COMP-2.
          01  MONTH   PIC X(9).
        PROCEDURE DIVISION USING MTH, MONTH.
        BEG-1.
            IF INITIAL-ENTRY
                MOVE WHEN-COMPILED TO WHEN-COMPILED-FLD
                DISPLAY WHEN-COMPILED-LINE
                DISPLAY 'INITIAL MTHNAM PROGRAM ENTRY'
            ELSE
                DISPLAY 'MTHNAM PROGRAM RE-ENTRY     '.
             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.
/*
//*
//LINKEDIT  EXEC  PGM=IEWL,
//           PARM='REUS,MAP,LIST'
//STEPLIB   DD  DSN=CEE.SCEELKED,DISP=SHR
//OBJECT    DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//          DD  DDNAME=SYSIN
//SYSLIB    DD  DSN=CEE.SCEELKED,DISP=SHR
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT  DD  SYSOUT=*
//SYSLMOD   DD  DSN=prefix.TSO.LOAD,DISP=SHR
//SYSLIN    DD  *
 MODE AMODE(24),RMODE(24)
 INCLUDE OBJECT
 ENTRY MTHNAM
 NAME MTHNAM(R)
/*
//*
```

where:

`prefix`

Is the high-level qualifier for your production FOCUS data sets.

Note:

- ❏ RENT is required for reentrancy.
- ❏ The linkedit parameter REUSE must be used to make the load module reusable.
- ❏ CEEUOPT is automatically included for compiles and linkedits.

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;
```

Example: Compiling the PL/I Version of MTHNAM Under LE

This example includes the following steps for compiling, linkediting, and calling the PL/I version of MTHNAM:

1. Compiling the COBOL stub:

```
//* Step 1 - compile the COBOL stub
//*
//COBSTUB   EXEC  IGYWCL,
//    PARM.COBOL='APOST,DYNAM,RENT',
//    PARM.LKED='LIST,MAP,SIZE=2046K'
//COBOL.SYSIN     DD  *
       IDENTIFICATION DIVISION.
       PROGRAM-ID. COBSTUB.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-370.
       OBJECT-COMPUTER. IBM-370.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       LINKAGE SECTION.
       01  DUMMY-IO  PIC X(99).
       PROCEDURE DIVISION USING DUMMY-IO.
       MAIN SECTION.
           CALL 'MTHNAM' USING DUMMY-IO.
       MAIN-EXIT.   EXIT.
           GOBACK.
/*
//*
//LKED.SYSIN    DD  *
 MODE AMODE(31),RMODE(24)
 ENTRY COBSTUB
 NAME COBSTUB(R)
/*
//LKED.SYSLMOD   DD  DSN=prefix.TSO.LOAD,DISP=SHR
//*
```

2. Compiling the PL/I subroutine:

```
//* Step 2 - compile the PLI program
//*
//COMPILE   EXEC  IEL1CL,
//      PARM.PLI='OBJECT,NODECK',
//      PARM.LKED='XREF,LIST'
//*
//PLI.SYSIN DD  *
 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',
                                 'PL/I SUB',
                                 'DECEMBER',
                                 '**ERROR**');
       MONTHNUM = MTHNUM + 0.00001 ;
       IF MONTHNUM < 1 | MONTHNUM > 12 THEN
          MONTHNUM = 13 ;
       FULLMTH = MONTH_TABLE(MONTHNUM);
        DISPLAY ('MTHNAM  ENTERED - ' || MONTHNUM ||' '|| FULLMTH);
        RETURN;
    END MTHNAM;
/*
//*
//LKED.SYSLMOD DD  DSN=prefix.TSO.LOAD,DISP=SHR
//LKED.SYSIN   DD  *
   ENTRY  MTHNAM
   NAME   MTHNAM(R)
/*
//*
```

3. Executing FOCUS:

```
//FOCUS      EXEC  PGM=FOCUS,COND=(0,NE)
//STEPLIB    DD  DSN=CEE.SCEERUN,DISP=SHR
//           DD  DSN=prefix.TSO.LOAD,DISP=SHR
//           DD  DSN=prefix.FOCLIB.LOAD,DISP=SHR
//           DD  DSN=prefix.FUSELIB.LOAD,DISP=SHR
//USERLIB    DD  DSN=prefix.FOCLIB.LOAD,DISP=SHR
//ERRORS     DD  DSN=prefix.ERRORS.DATA,DISP=SHR
//MASTER     DD  DSN=prefix.MASTER.DATA,DISP=SHR
//FOCEXEC    DD  DSN=prefix.FOCEXEC.DATA,DISP=SHR
//EMPLOYEE   DD  DSN=prefix.EMPLOYEE.FOCUS,DISP=SHR
 //SYSOUT     DD  SYSOUT=*
 //SYSPRINT   DD  SYSOUT=*
 //FSTRACE    DD  SYSOUT=*
 //OUT        DD  SYSOUT=*,DCB=BLKSIZE=121
 //OFFLINE    DD  SYSOUT=*
 //*SYSUDUMP   DD  DUMMY
 //SYSIN      DD  *
 SET PRINT = OFFLINE
 DEFINE FILE EMPLOYEE
 MONTH_NUM/M = PAY_DATE;
 PAY_MONTH/A12 = MTHNAM (MONTH_NUM, PAY_MONTH);
 END
 TABLE FILE EMPLOYEE
 HEADING
 "          "
 "FOCUS RELEASE - &FOCREL  PUT LEVEL - " &PUTLEVEL
 "SUBROUTINE - MTHNAM "
 "          "
 PRINT PAY_MONTH GROSS
BY EMP_ID BY FIRST_NAME BY LAST_NAME
BY PAY_DATE
IF LAST_NAME IS STEVENS
END
FIN
/*
//
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

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
*
         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
```

Example: Compiling the Assembler Version of MTHNAM Under LE

The following example compiles and linkedits the Assembler version of MTHNAM:

```
//ASSEMBLE  EXEC  PGM=ASMA90,
//          PARM='OBJECT,LIST,ESD,NODECK'
//SYSLIB    DD  DSN=CEE.SCEEMAC,DISP=SHR
//SYSUT1    DD  UNIT=SYSDA,SPACE=(TRK,(45,15))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(TRK,(45,15))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(TRK,(45,15))
//SYSPUNCH  DD  DUMMY
//SYSPRINT  DD  SYSOUT=*
//SYSLIN    DD  DSNAME=&&LINKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//          SPACE=(TRK,(3,3))
//SYSIN     DD  *
* ====================================================================
*
*   A SIMPLE MAIN ASSEMBLE ROUTINE THAT CALLS THE LE CALLABLE SERVICES
*
* ====================================================================
MTHNAM   CEEENTRY PPA=MAINPPA,AUTO=WORKSIZE,MAIN=NO
         USING WORKAREA,13
*
         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
*
*   TERMINATE THE CEE ENVIRONMENT AND RETURN TO THE CALLER
*
         CEETERM  RC=0
* ======================================================================
*              CONSTANTS
* ======================================================================
         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**'
*
MAINPPA  CEEPPA                      CONSTANTS DESCRIBING THE CODE BLOCK
* ======================================================================
*        THE WORKAREA AND DSA
* ======================================================================
WORKAREA DSECT
         ORG     *+CEEDSASZ          LEAVE SPACE FOR THE DSA FIXED PART
PLIST    DS      0D
PARM1    DS      A
PARM2    DS      A
```

```
PARM3    DS     A
PARM4    DS     A
PARM5    DS     A
*
FOCPARM1 DS     F                  SAVE FIRST PARAMETER PASSED
FOCPARM2 DS     F                  SAVE SECOND PARAMETER PASSED
*
         DS     0D
WORKSIZE EQU    *-WORKAREA
         CEEDSA                    MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA                    MAPPING OF THE COMMON ANCHOR AREA
*
          END   MTHNAM              NOMINATE MTHNAM AS THE ENTRY POINT
/*
//*
//IEBGENER  EXEC  PGM=IEBGENER,
//          COND=(0,NE)
//SYSUT1    DD   DSN=&&LINKSET,DISP=(OLD,PASS)
//SYSUT2    DD   SYSOUT=*
//SYSPRINT  DD   DUMMY
//SYSIN     DD   DUMMY
//*
//LINKEDIT  EXEC  PGM=IEWL,
//          PARM='LIST,XREF,LET,REUS',
//          COND=(0,NE)
//SYSLIB    DD   DSN=CEE.SCEELKED,DISP=SHR
//SYSUT1    DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLMOD   DD   DSN=prefix.TSO.LOAD(MTHNAM),DISP=SHR
//SYSPRINT  DD   SYSOUT=*
//SYSLIN    DD   DSNAME=&&LINKSET,DISP=(OLD,PASS)
//          DD   DDNAME=SYSIN
/*
//SYSIN     DD   *
 ENTRY MTHNAM
 NAME MTHNAM(R)
/*
//*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

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];
}
```

Example: Compiling the C Version of MTHNAM Under LE

The following example compiles and linkedits the C version of MTHNAM:

```
//COMPILE EXEC PGM=CBCDRVR,
//              PARM='SOURCE'
//STEPLIB     DD  DSNAME=CEE.SCEERUN,DISP=SHR
//            DD  DSNAME=CEE.SCEEH.H,DISP=SHR
//SYSLIN      DD  DSNAME=&&LOADSET,UNIT=VIO,
//            DD     DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT1      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=12800)
//SYSUT9      DD     UNIT=VIO,SPACE(32000,(30,30)),
//            DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10    DD  SYSOUT=*
//SYSUT14    DD     UNIT=VIO,SPACE(32000,(30,30)),
//           DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSPRINT   DD  SYSOUT=*
//SYSCPRT    DD  SYSOUT=*
//SYSIN      DD  *.DLM=XX
/*  #include <stdio.h> */
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];
 }
XX
//LKED      EXEC PGM=HEWL,
//          PARM='MAP'
//SYSLIB    DD  DSNAME=CEE.SCEELKED,DISP=SHR
//OBJECT    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD  DSNAME=SYSIN
//SYSPRINT  DD  SYSOUT=*
//SYSLMOD   DD  DSN= prefix.LOADLIB,DISP=SHR
//SYSUT1    DD  UNIT=VIO,SPACE=(32000,(30,30))
//SYSLIN    DD  *
  INCLUDE OBJECT
  NAME MTHNAM(R)
//*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

Note:

- A STEPLIB and/or SYSLIB may be required to access C functions.
- The Language Environment has been enhanced to load the required libraries for LE-C upon entering FOCUS.

Calling the MTHNAM Subroutine From a Request

Example:

Calling the MTHNAM Subroutine

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

In this section:

Formats and REXX Subroutines

Compiling FUSREXX Macros in CMS

How to:

Call a REXX Subroutine

Example:

Returning the Day of the Week

Passing Multiple Arguments to a REXX Subroutine

Accepting Multiple Tokens in a Parameter

Reference:

Storing and Searching for a REXX Subroutine

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 CMS and z/OS environments.

❑ In 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 z/OS, 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 CMS, the FILETYPE of a REXX subroutine is FUSREXX; it can be stored on any accessed disk.
- ❏ On z/OS, DDNAME FUSREXX must be allocated to a PDS. This library is searched before other z/OS libraries.

The search order for a REXX subroutine is:

1. FUSREXX
2. Standard CMS or z/OS 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 appears 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 YYMMDD 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 has 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 YYMMDD 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 appears 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 appears 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.

`Correct`	`Not as Clear`
`/* Some FUSREXX function */`	`/* Another FUSREXX function */`
`Arg input`	`Arg input`
`some rexx process ...`	`some rexx process ...`
`Return data_to_FOCUS`	`Exit 0`

Formats and REXX Subroutines

Example:

Returning a Result in Alphanumeric Format

Returning a Result in Integer Format

Passing a Date Value as an Alphanumeric Field With Date Options

Passing a Date as a Date Converted to Alphanumeric Format

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 a 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 appears 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 CMS

The SUM2 FUSREXX macro takes two amounts as input and returns the sum in integer format:

```
/* SUM2 routine. Add amount1 to amount2 and return as integer */
Arg amt1, amt2 .
Return D2C(amt1 + amt2,4)
```

To compile and compress this FUSREXX macro in CMS, issue the following command. Note that the file identifier must be in 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
```

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 328

accessing external functions 62
- CMS 64
- z/OS 62

accessing functions 39, 41 to 42
- z/OS 62

ADD function 296 to 297

alphanumeric argument 43

alphanumeric strings 306

ARGLEN function 68 to 69

argument formats 43

argument length 44

argument types 42

ASCII character chart 32

ASIS function 69 to 70, 329

assigning date-time values 203
- COMPUTE command 203 to 204
- DEFINE command 203
- IF criteria 203, 206
- WHERE criteria 203, 205

ATODBL function 306 to 307

AYM function 271 to 272

AYMD function 273 to 274

B

bar charts 329

BAR function 329 to 331

batch allocation 62

bit strings 72 to 73

BITSON function 70 to 71

BITVAL function 72 to 73

branching in procedures 52
- functions and 52, 55

BUSDAYS parameter 195

business days 194 to 195
- BUSDAYS parameter 195

BYTVAL function 74 to 75

C

calling functions 39 to 42
- Dialogue Manager and 50
- FOCUS commands and 48
- from another function 55
- IF criteria 56 to 57
- in Maintain 42
- RECAP command and 59 to 60
- WHEN criteria 58
- WHERE criteria 56 to 57

CHAR2INT function 142

character chart 32

character functions 20, 67
ARGLEN 68 to 69
ASIS 69 to 70
BITSON 70 to 71
BITVAL 72 to 73
BYTVAL 74 to 75
CHKFMT 76 to 77
CTRAN 79 to 81
CTRFLD 86 to 87
DCTRAN 88
DSTRIP 89
EDIT 90 to 91
GETTOK 92 to 93
JPTRANS 94
LCWORD 99 to 100, 144 to 145
LJUST 100 to 101
LOCASE 102 to 103
Maintain-specific 22, 141
OVRLAY 103 to 105
PARAG 106 to 108
POSIT 108 to 110
RJUST 110 to 111
SOUNDEX 112 to 113
SPELLNM 114 to 115
SQUEEZ 115 to 116
STRIP 117 to 119
SUBSTR 122 to 124, 135
TRIM 124 to 126
TRIMV 137
UPCASE 126 to 127

character strings 67 to 68, 100
adding 147
bits 70, 72
centering 86 to 87
comparing 112, 158, 160
converting case 102, 126, 144, 147, 165
determining length 164
dividing 106
extracting 147, 161
extracting characters 90

character strings *(continued)*
extracting substrings 92, 122, 124, 135, 162
finding substrings 108, 152
format 76
justifying 100, 110, 153
measuring length 68, 145
overlaying 103, 150
reducing spaces 115
removing occurrences 124, 164
right-justifying 110, 153
spelling out numbers 114
substrings 156, 160
translating characters 74, 79 to 80, 142 to 143

CHGDAT function 274, 276 to 277

CHKFMT function 76 to 77

CHKPCK function 332 to 333

CMS 401
compiling 401
storing external functions 64
storing subroutines 401

commands 376
passing 376

compiling subroutines 401
CMS 401
z/OS 402

components 208

COMPUTE command
assigning date-time values 204
COMPUTE command 48

controlling function parameter verification 46 converting formats 305 to 306

creating subroutines 387 to 388

cross-referenced data sources 184

CTRAN function 79 to 81

CTRFLD function 86 to 87

custom subroutines 403 to 404, 407, 411, 416, 420, 423

D

DA functions 278

DADMY function 278

DADYM function 278

DAMDY function 278

DAMYD function 278

data sets 367, 369

data source functions 24, 167
- FIND 177 to 181
- LAST 182 to 183
- LOOKUP 184 to 187, 189 to 190

data source values 167
- decoding 172
- retrieving 182 to 183
- verifying 177 to 181

data sources 167
- cross-referenced 184, 190
- retrieving values 182 to 184, 190
- values 167
- verifying values 177 to 181

date and time functions 24, 191
- arguments and 208
- AYM 271 to 272
- AYMD 273 to 274
- CHGDAT 274, 276 to 277

date and time functions *(continued)*
- DA 278
- DADMY 278
- DADYM 278
- DAMDY 278
- DAMYD 278
- DATEADD 211 to 213
- DATECVT 214 to 215
- DATEDIF 216 to 218
- DATEMOV 219 to 222
- DATETRAN 223
- DAYDM 278
- DAYMD 278 to 279
- DOWK 281 to 282
- DOWKL 281
- DTDMY 282 to 283
- DTDYM 282 to 283
- DTMDY 282 to 284
- DTMYD 282 to 283
- DTYDM 282 to 283
- DTYMD 282 to 283
- GREGDT 285 to 286
- HADD 237 to 239
- HCNVRT 239 to 240
- HDATE 241 to 242
- HDIFF 242 to 244
- HDTTM 244 to 246
- HGETC 248 to 249
- HHMMSS 249 to 250
- HINPUT 251 to 252
- HMIDNT 255 to 256
- HNAME 257 to 258
- HPART 259 to 260
- HSETPT 261 to 262
- HTIME 263 to 264
- JULDAT 287 to 288
- legacy 24, 26, 191 to 192
- Maintain-specific 291
- standard 24, 191 to 193
- terminology 192
- TIMETOTS 265 to 266

date and time functions *(continued)*
  TODAY 266 to 267
  work days 194
  YM 289 to 290
  YMD 280

date argument 43

date formats 199
  formatted-string format 201
  international 223
  numeric string format 200
  translated-string format 201

DATEADD function 211 to 213

DATECVT function 214 to 215

DATEDIF function 216 to 219

DATEFNS parameter 268
  GREGDT function 285
  JULDAT function 287

DATEFORMAT parameter 199

DATEMOV function 219 to 222

date-time format
  ISO standard input values 202

date-time functions 246
  HEXTR 246
  HMASK 252 to 253

date-time values 291
  adding 271, 273, 296
  assigning 203
  converting 263, 265
  converting formats 239, 241, 244, 274, 278, 282, 287
  determining day of week 303
  determining quarter 300
  elapsed time 289, 298
  extracting 297, 299, 304
  finding day of week 281

date-time values *(continued)*
  finding difference 216, 242, 280
  incrementing 237
  moving dates 219
  retrieving 292 to 295
  retrieving components 259
  retrieving time 249
  returning dates 266
  setting 301
  setting time 255
  storing 248
  subtracting 271, 273, 302

DATETRAN function 223

DAY function 297

DAYDM function 278

DAYMD function 278 to 279

DB_LOOKUP function 168
  COMPUTE command 168
  DEFINE 168
  MODIFY 168
  TABLE COMPUTE 168

DCTRAN function 88

DECODE function 172 to 176

decoding functions 24, 167, 172 to 176

decoding values 172
  from files 172, 174 to 176
  in a function 172 to 173

DEFINE command 48
  functions and 48

Dialogue Manager 50
  functions and 50

Dialogue Manager commands 50
    -IF 52
    -RUN 54 to 55
    -SET 50 to 51

DMOD function 335 to 336

DMY function 280

DOWK function 281 to 282

DOWKL function 281

DSTRIP function 89

DTDMY function 282 to 283

DTDYM function 282 to 283

DTMDY function 282 to 284

DTMYD function 282 to 283

DTSTRICT parameter 209

DTYDM function 282 to 283

DTYMD function 282 to 283

E

EBCDIC character chart 32

EDALIB.LOAD library 61

EDIT function 90 to 91, 310

enabling parameter verification 45

entry points 396 to 397

error messages 366

EXP function 336 to 337

external functions 18, 61

F

FEXERR function 366

FIND function 177 to 181

FINDMEM function 367 to 368

first day of the week 207

FML (Financial Modeling Language) 340
    retrieving tag lists 340
    retrieving tag values 341

FML hierarchies 342

FMLCAP function 342 to 343

FMLFOR function 341

FMLINFO function 338 to 339

FMLLIST function 340

FMOD function 335 to 336

FOCUS commands 48

FOR lists 340
    retrieving 340

format conversion functions 28, 305
    ATODBL 306 to 307
    EDIT 310
    FTOA 311 to 312
    HEXBYT 312 to 314
    ITONUM 316 to 317
    ITOPACK 317 to 318
    ITOZ 319 to 320
    PCKOUT 321 to 322
    PTOA 322 to 324
    UFMT 325 to 326

format conversions 306
- packed numbers 321
- to alphanumeric 311, 322
- to characters 312
- to double-precision 316
- to hexadecimal 325
- to packed decimal 317
- to zoned format 319

formats 306
- alphanumeric 310
- converting 306

formatted-string format 201

FTOA function 311 to 312

function arguments 42
- formats 43
- functions as 55
- in subroutines 389
- length 44
- number 44
- types 42

function libraries 42

function types 19
- character 20, 67, 141
- data source 24, 167
- date and time 24, 191
- decoding 24, 167
- format conversion 28, 305
- light update support 381
- Maintain-specific character 22
- numeric 29, 327
- system 30 to 31, 365

functions 18, 39 to 40, 168, 379
- accessing 39
- branching in procedures 52, 57
- calling 39 to 42, 48, 55
- COMPUTE command and 48
- DCTRAN 88

functions *(continued)*
- DEFINE command and 48
- Dialogue Manager and 50
- DSTRIP 89
- external 18
- FIND 177 to 181
- FMLCAP 342 to 343
- FMLFOR 341
- FMLINFO 337 to 339
- FMLLIST 340
- FOCUS commands and 49
- HEXTR 246
- HMASK 252 to 253
- -IF command and 52
- internal 18
- invoking 41
- JPTRANS 94
- MIRR 347
- operating system commands and 54 to 55
- -RUN command and 54 to 55
- SLEEP 379
- STRREP 120
- subroutines 18, 387
- types 19
- VALIDATE command and 48
- variables and 50 to 51
- XIRR 361

FUSELIB.LOAD library 61

G

GETPDS function 369, 371, 373

GETTOK function 92 to 93

GETUSER function 374

GREGDT function 285 to 286
- DATEFNS parameter 285

H

HADD function 237 to 239

HCNVRT function 239 to 240

HDATE function 241 to 242

HDAY parameter 196 to 197

HDIFF function 242 to 244

HDTTM function 244 to 246

HEXBYT function 312 to 314

HEXTR function 246

HGETC function 248 to 249

HHMMSS function 249 to 250, 292 to 293, 375

HINPUT function 251 to 252

HMASK function 252 to 253

HMIDNT function 255 to 256

HNAME function 257 to 258

holidays 194 to 197
    HDAY parameter 196 to 197
    holiday files 196

HPART function 259 to 260

HSETPT function 261 to 262

HTIME function 263 to 264

I

-IF command 52

IF criteria 56
    assigning date-time values 206
    functions and 56 to 57

IMOD function 335 to 336

Initial_HHMMSS function 293

Initial_TODAY function 294

INT function 344

INT2CHAR function 143

internal functions 18

internal modified rate of return 347

internal rate of return 361

international date formats 223

invoking functions 40 to 41

ISO standard date-time formats 202

ITONUM function 316 to 317

ITOPACK function 317 to 318

ITOZ function 319 to 320

J

JPTRANS function 94

JULDAT function 287 to 288
    DATEFNS parameter 287

JULIAN function 298

L

LAST function 182 to 183

LCWORD function 99 to 100, 144 to 145

LCWORD2 function 144 to 145

LEADZERO parameter 198

legacy date functions 24, 192
- DATEFNS parameter 268
- DMY 280
- legacy dates 269 to 270
- legacy versions 268
- MDY 280
- YMD 280

legacy dates 269 to 270

LENGTH function 145 to 146

light update support functions 381

LJUST function 100 to 101, 146

load libraries 61 to 62

LOCASE function 102 to 103

LOG function 345

LOOKUP function 184 to 187, 189 to 190
- extended function 190

LOWER function 147

M

Maintain data source functions 178 to 179

Maintain-specific character functions 22, 141
- CHAR2INT 142
- INT2CHAR 143
- LCWORD 144
- LCWORD2 144 to 145
- LENGTH 145 to 146
- LJUST 146
- LOWER 147
- MASK 147 to 148
- NLSCHR 149
- OVRLAY 150 to 151
- POSIT 152 to 153
- RJUST 153 to 154
- SELECTS 154 to 155

Maintain-specific character functions *(continued)*
- STRAN 156 to 157
- STRCMP 158 to 159
- STRICMP 160
- STRTOKEN 161 to 162
- SUBSTR 162 to 163
- TRIM 164
- TRIMLEN 164 to 165
- UPCASE 165

Maintain-specific date and time functions 291
- ADD 296 to 297
- DATEDIF 219
- DATEMOV 222
- DAY 297
- HHMMSS 292 to 293
- Initial_HHMMSS 293
- Initial_TODAY 294
- JULIAN 298
- MONTH 299
- QUARTER 300
- SETMDY 301
- standard 291
- SUB 302
- TODAY 294 to 295
- TODAY2 295
- WEEKDAY 303
- YEAR 304

Maintain-specific functions 42
- light update support 381
- MNTUWS function library 42
- SELECTS 154
- STRNCMP 160 to 161
- STRTOKEN 161

MASK function 147 to 148

MAX function 346

MDY function 280

MIN function 346

MIRR function 347

MNTUWS function library 42

modified rate of return 347

MODIFY data source functions 180 to 181

MONTH built-in function 299

MONTH function 299

MTHNAM subroutine 403 to 404, 407, 411, 416, 420, 423

MVSDYNAM function 376

N

naming subroutines 389

National Language Support (NLS) 149

NLS (National Language Support) 149

NLSCHR function 149

NORMSDST function 351, 353 to 354

NORMSINV function 351, 353 to 354

number of arguments 44

numbers 328
- absolute value 328
- bar charts 329
- calculating remainders 335
- generating random 355, 358
- greatest integer 344
- logarithms 345
- maximum 346
- minimum 346
- raising to a power 336
- square root 360
- standard normal deviation 351 to 352
- validating packed fields 332

numeric argument 43

numeric functions 29, 327
- ABS 328
- ASIS 329
- BAR 329 to 331
- CHKPCK 332 to 333
- DMOD 335 to 336
- EXP 336 to 337
- FMLCAP 342 to 343
- FMLFOR 341
- FMLINFO 337 to 340
- FMOD 335 to 336
- IMOD 335 to 336
- INT 344
- LOG 345
- MAX 346
- MIN 346
- NORMSDST 351, 353 to 354
- NORMSINV 351, 353 to 354
- PRDNOR 355 to 357
- PRDUNI 355 to 356
- RDNORM 358 to 359
- RDUNIF 358 to 359
- SQRT 360

numeric string format 200

numeric values 327

O

operating system commands 54 to 55

order of arguments 44

OVRLAY function 103 to 105, 150 to 151

P

PARAG function 106 to 108

PCKOUT function 321 to 322

POSIT function 108 to 110, 152 to 153

PRDNOR function 355 to 357

PRDUNI function 355 to 356

programming subroutines 395
- arguments 398 to 400

PTOA function 322 to 324

Q

QUARTER function 300

R

rate of return 347, 361

RDNORM function 358 to 359

RDUNIF function 358 to 359

RECAP command 59 to 60

retrieving FML hierarchy captions 342

return rate functions 347
- MIRR 347
- XIRR 361

REXX subroutines 424 to 427, 429 to 434
- formats 430

RJUST function 110 to 111, 153 to 154

-RUN command 54 to 55

S

SELECTS function 154 to 155

-SET command 50 to 51

SET parameters 194
- BUSDAYS 195
- DATEFNS 268
- DATEFORMAT 199
- DTSTRICT 209
- HDAY 195 to 197
- LEADZERO 198
- WEEKFIRST 207

SETMDY function 301

SLEEP function 379

SOUNDEX function 112 to 113

SPELLNM function 114 to 115

SQRT function 360

SQUEEZ function 115 to 116

standard date and time functions 24, 192 to 193

standard normal deviation 351 to 352

storing external functions 62
- CMS 64
- z/OS 62 to 63

storing subroutines 401 to 402
- CMS 401
- libraries 402
- text files 402
- z/OS 402

STRAN function 156 to 157

STRCMP function 158 to 159

STRICMP function 160

string replacement 120

STRIP function 117 to 119

STRNCMP function 160 to 161

STRREP function 120

STRTOKEN function 161 to 162

SUB function 302

subroutines 18, 387
- compiling 401
- creating 387
- custom 403 to 404, 407, 411, 416, 420, 423
- entry points 396 to 397
- MTHNAM 403 to 404, 407, 411, 416, 420, 423
- naming 389
- programming 395
- REXX 424 to 427, 429 to 434
- storing 401
- testing 403
- writing 388

SUBSTR function 122 to 124, 135, 162 to 163

substrings 90
- comparing 160
- extracting 90, 92, 122, 124, 135, 162
- finding 108, 152
- overlaying character strings 103, 150
- substituting 156

system functions 30 to 31, 365
- FEXERR 366
- FINDMEM 367 to 368
- GETPDS 369, 371, 373
- GETUSER 374
- HHMMSS 375
- MVSDYNAM 376
- TODAY 380

T

tag lists 340
- retrieving 340

tag values 341

testing subroutines 403

time formats 201 to 202

TIMETOTS function 265 to 266

TODAY function 266 to 267, 294 to 295, 380

TODAY2 function 295

translated-string format 201

TRIM function 124 to 126, 164

TRIMLEN function 164 to 165

TRIMV function 137

U

UFMT function 325 to 326

UPCASE function 126 to 127, 165

user IDs 374

USERFCHK setting 45 to 46

USERFNS setting 45

V

VALIDATE command 48

values 172
- decoding 172
- verifying 177 to 181

verifying function parameters 45
- controlling 46
- enabling 45

W

WEEKDAY function 303

WEEKFIRST 194

WEEKFIRST parameter 207

WHEN criteria 58

WHERE criteria 205
- assigning date-time values 205
- functions and 56 to 57

work days 194
- business days 194 to 195
- holidays 194 to 196

writing subroutines 388
- creating arguments 389
- entry points 396 to 397
- languages 392
- naming subroutines 389
- programming 395, 398 to 400

X

XIRR function 361

Y

YEAR function 304

YM function 289 to 290

YMD function 280

Z

z/OS 402
- compiling subroutines 402
- storing functions 62 to 63
- storing subroutines 402

z/VM 64
- accessing external functions 64
- compiling subroutines 401

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

Information Builders, Inc. Two Penn Plaza, New York, NY 10121-2898 (212) 736-4433

Using Functions
Version 7.6

DN1001140.0308