Information Builders

FOCUS® for S/390®

Using Functions
Version 7.2

DN1001140.1101

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 for S/390 documentation set.

The documentation set consists of the following components:

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

The users' documentation for FOCUS Version 7.2 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 is organized as follows:

Chapter/Appendix		Contents
1	*Introducing Functions*	Offers an introduction to functions and explains the different types of functions available.
2	*Accessing and Invoking a Function*	Describes the considerations for supplying arguments in a function, explains how to use a function in a command, and how to access externally-stored functions.
3	*Character Functions*	Describes the available character functions, which enable you to manipulate alphanumeric fields and character strings
4	*Data Source and Decoding Functions*	Describes the available data source functions, which enable you to search for or retrieve data source records or values.
5	*Date and Time Functions*	Describes the available date and time functions, which enable you to manipulate date and time values.
6	*Format Conversion Functions*	Describes the available format conversion functions, which convert fields from one format to another.
7	*Numeric Functions*	Describes the available numeric functions, which enable you to perform calculations on numeric constants and fields.
8	*System Functions*	Describes the available system functions, which enable you to make calls to the operating system to obtain information about the operating environment or to use a system service
A	*Creating Your Own Subroutines*	Describes how to create and store site-specific functions.

Summary of New Features

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

New Feature	Manual	Chapter
Field-based Reformatting	*Creating Reports*	Chapter 1, *Creating Tabular Reports*
Increased Report Width	*Creating Reports*	Chapter 1, *Creating Tabular Reports*
ACROSS-TOTAL	*Creating Reports*	Chapter 4, *Sorting Tabular Reports*
Tiles	*Creating Reports*	Chapter 4, *Sorting Tabular Reports*
DEFINE FILE SAVE and DEFINE FILE RETURN	*Creating Reports*	Chapter 6, *Creating Temporary Fields*
Forecast	*Creating Reports*	Chapter 6, *Creating Temporary Fields*
Creating Comma-Delimited Files	*Creating Reports*	Chapter 11, *Saving and Reusing Report Output*
Creating Tab-Delimited Files	*Creating Reports*	Chapter 11, *Saving and Reusing Report Output*
Long Master File Names	*Creating Reports*	Chapter 11, *Saving and Reusing Report Output*
JOIN WHERE	*Creating Reports*	Chapter 13, *Joining Data Sources*
KEEPDEFINES	*Creating Reports*	Chapter 13, *Joining Data Sources*
Long Master File Names	*Describing Data*	Chapter 1, *Understanding a Data Source Description*
4K Alpha Fields	*Describing Data*	Chapter 4, *Describing an Individual Field*
Extended Currency Symbol Support	*Describing Data*	Chapter 4, *Describing an Individual Field*
SUFFIX = COMT/COMMA/TABT	*Describing Data*	Chapter 5, *Describing a Sequential, VSAM, or ISAM Data Source*

New Feature	Manual	Chapter
AUTODATE	*Describing Data*	Chapter 6, *Describing a FOCUS Data Source*
CDN parameter	*Developing Applications*	Chapter 1, *Customizing Your Environment*
CENT-ZERO parameter	*Developing Applications*	Chapter 1, *Customizing Your Environment*
ERROROUT parameter	*Developing Applications*	Chapter 1, *Customizing Your Environment*
KEEPDEFINES parameter	*Developing Applications*	Chapter 1, *Customizing Your Environment*
PCOMMA parameter	*Developing Applications*	Chapter 1, *Customizing Your Environment*
Unlimited nested -INCLUDE commands	*Developing Applications*	Chapter 2, *Managing an Application With Dialogue Manager*
SQUEEZ function	*Using Functions*	Chapter 3, *Character Functions*
STRIP function	*Using Functions*	Chapter 3, *Character Functions*
TRIM function	*Using Functions*	Chapter 3, *Character Functions*
DYNAM ALLOC LONGNAME	*Overview and Operating Environments*	Chapter 5, *OS/390 and MVS Guide to Operations*

Documentation Conventions

The following conventions apply throughout this manual:

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

Related Publications

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

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

Customer Support

Do you have questions about FOCUS?

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

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, http://www.informationbuilders.com. It connects you to the tracking system and known-problem 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 number (*xxxx.xx*).
- The FOCEXEC procedure (preferably with line numbers).
- Master File with picture (provided by CHECK FILE).
- Run sheet (beginning at login, including call to FOCUS), containing the following information:
  - ? RELEASE
  - ? FDT
  - ? LET
  - ? LOAD
  - ? COMBINE
  - ? JOIN
  - ? DEFINE
  - ? STAT
  - ? SET
  - ? SET GRAPH
  - ? USE
  - For MVS, ? TSO DDNAME
  - For VM, CMS QFI

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

User Feedback

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

Thank you, in advance, for your comments.

Information Builders Consulting and Training

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

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

Contents

CHAPTER 1

Introducing Functions

Topics:

- Using Functions
- Types of Functions

This topic offers an introduction to functions and explains the different types of functions available.

Using Functions

Functions operate on one or more arguments and return a single value or character string. The return value or string 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 two types of functions:

- **Internal** functions are built into FOCUS and require no extra work to access or use. The following are internal functions. All other functions are external.
  - ABS function
  - ASIS function
  - DMY, MDY, and YMD function
  - DECODE function
  - EDIT function
  - FIND function
  - LAST function
  - LOG function
  - LOOKUP function
  - MAX and MIN function
  - SQRT function
- **External** functions are stored in an external library that must be accessed. When you invoke these functions, an extra argument specifying the output field or format of the result is required.

For information on how to use an internal or external function, see Chapter 2, *Accessing and Invoking a Function*.

Types of Functions

You can access any of the following kinds of functions:

- **Character** functions manipulate alphanumeric fields or character strings. For details, see *Character Functions* on page 1-3.
- **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 1-6.
- **Date and Time** functions manipulate dates and times. For details see *Date and Time Functions* on page 1-7.
- **Format Conversion** functions convert fields from one format to another. For details, see *Format Conversion Functions* on page 1-12.
- **Numeric** functions perform calculations on numeric constants and fields. For details, see *Numeric Functions* on page 1-13.
- **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 1-15.

Character Functions

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

ARGLEN function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

ASIS function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, Windows NT/2000

Available Languages: reporting

BITSON function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

BITVAL function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

BYTVAL function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

CHKFMT function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

CTRAN function

Converts one character in a string to another character.

Available Operating Systems: All

Available Languages: reporting, Maintain

CTRFLD function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

EDIT function

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

Available Operating Systems: All

Available Languages: reporting

GETTOK function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

LCWORD function

Converts the letters in a given string to mixed case.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

LJUST function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

LOCASE function

Converts alphanumeric text to lowercase.

Available Operating Systems: All

Available Languages: reporting, Maintain

OVRLAY function

Overlays a base character string with a substring.

Available Operating Systems: All

Available Languages: reporting, Maintain

PARAG function

Divides lines of text into smaller lines with delimiters.

Available Operating Systems: All

Available Languages: reporting, Maintain

POSIT function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

Available Languages: reporting, Maintain

RJUST function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

SOUNDEX function

Searches for a character string phonetically rather than by its spelling.

Available Operating Systems: All

Available Languages: reporting, Maintain

SQUEEZ function

Reduces multiple contiguous blank characters within a string to a single blank character.

Available Operating Systems: All

Available Languages: reporting, Maintain

STRIP function

Removes all occurrences of a specific character from a string.

Available Operating Systems: All

Available Languages: reporting, Maintain

SUBSTR function

Extracts a substring based on where it starts and ends in the parent string.

Available Operating Systems: All

Available Languages: reporting, Maintain

TRIM function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

UPCASE function

Converts alphanumeric text to uppercase.

Available Operating Systems: All

Available Languages: reporting, Maintain

Data Source and Decoding Functions

The following functions search for data source records, retrieve data source records or values, and assign values. For details, see Chapter 4, *Data Source and Decoding Functions.*

DECODE function

Assigns values based on the value of an input field.

Available Operating Systems: All

Available Languages: reporting, Maintain

FIND function

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

Available Operating Systems: All

Available Languages: reporting

LAST function

Retrieves the preceding value selected for a field.

Available Operating Systems: All

Available Languages: reporting

LOOKUP function

Retrieves a value from a cross-referenced file.

Available Operating Systems: All

Available Languages: reporting

Date and Time Functions

The following functions manipulate dates and times. For details, see Chapter 5, *Date and Time Functions*.

Standard Date and Time Functions

DATEADD function

Adds or subtracts years, months, or days to or from a date.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

DATECVT function

Converts dates from one date format to another.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

DATEDIF function

Calculates the difference between two dates, expressed as years, months, or days.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

DATEMOV function

Moves a date to a significant point on the calendar.

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HADD function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HCNVRT function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HDATE function

Extracts the date portion of a date-time field and converts it to a date format.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HDIFF function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HDTTM function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HGETC function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000 NT/2000

Available Languages: reporting, Maintain

HHMMSS function

Retrieves the current time from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

HINPUT function

Converts an alphanumeric string to a date-time value.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HMIDNT function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HNAME function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HPART function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HSETPT function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

HTIME function

Converts the time portion of a date-time field to a numeric number of milliseconds (if the first argument is 8) or microseconds (if the first argument is 10).

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

TODAY function

Retrieves the current date from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

Legacy Date Functions

AYM function

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

Available Operating Systems: OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

AYMD function

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

Available Operating Systems:OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

CHGDAT function

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

Available Operating Systems: OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

DA functions

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

DADMY converts dates in day-month-year format.

DADYM converts dates in day-year-month format.

DAMDY converts dates in month-day-year format.

DAMYD converts dates in month-year-day format.

DAYDM converts dates in year-day-month format.

DAYMD converts dates in year-month-day format.

Available Operating Systems: All

Available Languages: reporting, Maintain

DMY, MDY, and YMD functions

Calculate the difference between two dates.

Available Operating Systems: All

Available Languages: reporting, Maintain

DOWK[L] functions

Determine the day of the week for dates.

Available Operating Systems: All

Available Languages: reporting, Maintain

DT functions

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

DTDMY converts numbers to day-month-year dates.

DTDYM converts numbers to day-year-month dates.

DTMDY converts numbers to month-day-year dates.

DTMYD converts numbers to month-year-day dates.

DTYDM converts numbers to year-day-month dates.

DTYMD converts numbers to year-month-day dates.

Available Operating Systems: All

Available Languages: reporting, Maintain

GREGDT function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

JULDAT function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

YM function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

Format Conversion Functions

The following functions convert fields from one format to another. For details, see Chapter 6, *Format Conversion Functions*.

ATODBL function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

EDIT function

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

Available Operating Systems: OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting

FTOA function

Converts a number in a numeric format to alphanumeric format.

Available Operating Systems: All

Available Languages: reporting, Maintain

HEXBYT function

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

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

ITONUM function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

ITOPACK function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

ITOZ function

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

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

PCKOUT function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

UFMT function

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

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

Available Languages: reporting, Maintain

Numeric Functions

The following functions perform calculations on numeric constants or fields. For details, see Chapter 7, *Numeric Functions.*

ABS function

Returns the absolute value of its argument.

Available Operating Systems: All

Available Languages: reporting, Maintain

ASIS function

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, Windows NT/2000

Available Languages: reporting

BAR function

Produces horizontal bar charts in reports.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

CHKPCK function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

DMOD, FMOD, and IMOD functions

Calculate the remainder from a division.

Available Operating Systems: All

Available Languages: reporting, Maintain

EXP function

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

Available Operating Systems: All

Available Languages: reporting, Maintain

EXPN function

Evaluates an argument expressed in scientific notation.

Available Operating Systems: AS/400, OpenVMS, Windows NT/2000

Available Languages: reporting

INT function

Returns the integer part of its argument.

Available Operating Systems: All

Available Languages: reporting, Maintain

LOG function

Returns the natural logarithm of its argument.

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

MAX and MIN functions

Return the maximum or minimum value from a list of arguments.

Available Operating Systems: All

Available Languages: reporting, Maintain

PRDNOR and PRDUNI functions

Generate reproducible random numbers.

Available Operating Systems: All

Available Languages: reporting, Maintain

RDNORM, and RDUNIF functions

Generate random numbers.

Available Operating Systems: All

Available Languages: reporting, Maintain

SQRT function

Returns the square root of its argument.

Available Operating Systems: All

Available Languages: reporting, Maintain

System Functions

The following functions call the operating system to obtain information about the operating environment or to use a system service. For details, see Chapter 8, *System Functions.*

FEXERR function

Retrieves error messages.

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

FINDMEM function

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

Available Operating Systems: OS/390, VM/CMS

Available Languages: reporting, Maintain

GETPDS function

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

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

GETUSER function

Retrieves the user ID from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

HHMMSS function

Retrieves the current time from the system.

Available Operating Systems:

Available Languages: reporting, Maintain

MVSDYNAM function

Passes a DYNAM command to the command processor.

Available Operating Systems: OS/390, VM/CMS

Available Languages: reporting, Maintain

TODAY function

Retrieves the current date from the system.

Available Operating Systems: All

Available Languages: reporting, Maintain

CHAPTER 2

Accessing and Invoking a Function

Topics:

- Invoking a Function
- Using an Argument in a Function
- Using a Function in a FOCUS Command
- Using a Function With a Dialogue Manager Command
- Using a Function in Another Function
- Using a Function in WHERE or IF Criteria
- Using a Function in WHEN Criteria
- Using a Function in a RECAP Command
- Accessing a Function
- Dynamic Language Environment Support

This topic describes the considerations for supplying arguments in a function, explains how to use a function in a command, and how to access externally-stored functions.

Invoking a Function

A function can be invoked in a command, or as part of an expression. It is invoked with the function name, arguments, and, for some functions, an output field.

You can invoke a function from a FOCUS command, Dialogue Manager command, or FML command. For details see the topic on the appropriate command.

Syntax How to Invoke a Function

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

where:

function

Is the name of the function.

arg1, arg2, ...

Are the arguments.

outputfield

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

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

Syntax How to Store Output in a Field

```
{DEFINE|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:

field

Is the field in which the output is to be stored.

file

Is the file in which the virtual field will be created.

var

Is the variable in which the output is to be stored.

fmt

Is the format of the output field.

function

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

input1, input2,...

Are the input function arguments, which are data values and fields that the function needs to do its processing. For more information about arguments see *Using an Argument in a Function* on page 2-3.

outfield

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

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

Using an Argument in a Function

When using an argument in a function, there are several considerations. You must understand what types of arguments are acceptable, the formats and lengths for these arguments, and the number and order of arguments.

Argument Types

The following are acceptable arguments for a function:

- Numeric constant, such as 6 or 15.
- Date constant, such as 022894.
- Alphanumeric literal, such as STEVENS or NEW YORK NY. A literal must be enclosed in single quotation marks.
- Number stored in alphanumeric format.
- Date in alphanumeric, numeric, or date 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.
- As an input argument for a RECAP command, row or column reference (R notation, E notation, or label) or names of other RECAP calculations.
- Another function.

Argument Formats

Depending on the function, an argument can be in either 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. These are the types of formats:

- 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 RUN commands (for example, -MVS RUN).
- A **numeric argument** is stored internally as a binary or packed number. A numeric argument includes integer (I), floating-point (F), double-precision (D), and packed (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 double-precision format when used with a function, but results are returned in the format specified for your output field.
- 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 specify an argument with a two digit year, the function will specify a century based on the DATEFNS, YRTHRESH, and DEFCENT settings.

Argument Length

An argument is passed to a function by reference, meaning that the memory location of the argument is passed. Therefore, no indication of the length of the argument is implied.

When needed (for alphanumeric strings), the argument length must be passed as a separate argument. Some functions require a length for the input arguments and output arguments (for example, SUBSTR), and others use one length for both input and output 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 a string is used. For example, passing the argument 'ABCDEF' and specifying a length of 3 is treated as a string of 'ABC'.
- If the specified length is too long, whatever is in memory beyond the length is included. For example, passing an argument of 'ABC' and specifying a length of 6 is treated as a string beginning with 'ABC' plus whatever three characters are in the next three positions of memory. Depending on memory utilization, the extra three characters can be anything.
- Some operating system routines are very sensitive to incorrectly specified lengths and read into incorrectly formatted memory areas.

Number and Order of Arguments

The number of arguments required varies according to each function. Built-in functions may require up to six arguments. Customized functions may require any number of arguments. The maximum number of arguments per function, including the output argument, is 28. If the function requires more than 28 arguments, you must use two or more call statements to pass the arguments to the function.

Arguments must be specified in the order specified in the syntax of each function in this manual. The required order varies between functions.

Using a Function in a FOCUS Command

A function can be called from the DEFINE command or Master File attribute, the COMPUTE command, or VALIDATE command.

Syntax **How to Use a Function in a COMPUTE or DEFINE 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 to be used.

tempfield

Is the temporary field to be created by the DEFINE or COMPUTE command. This is the same field specified by *outfield*. If the function returns output as the format of the output value, 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 defined; otherwise, it is optional.

function

Is the name of the function.

input1, input2, input3...

Are the arguments.

outfield

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. This is required only for some functions.

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

Using a Calculation or Compound IF Command With a COMPUTE Command

In a calculation or compound IF command, you must specify the format for the output value. There are two methods for this:

- Pre-define the format of the output field with a separate command. For example, in the following, 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 if AMOUNT is greater or less than 500 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';
```

- Specify the last argument in the argument list as the format. For example, in the following, the command tests the returned value directly. This is possible because the function defines the format of the return value (D8.2).

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

Using a Function With a Dialogue Manager Command

You can use a function with Dialogue Manager. You can do this in the following ways:

- Store the result of a function in a variable. For details see *Assigning the Result of a Function to a Variable* on page 2-7.
- Use a function in a -IF command. For details see *Using a Function in a -IF Command* on page *2-8.*
- Use a function in a -RUN command. For details see *Using a Function in an Operating System -RUN Command* on page 2-9.

Dialogue Manager converts a numeric argument to double precision format whether or not the argument type is supposed to be character. This means you must be careful when supplying arguments for a function in Dialogue Manager. If Dialogue Manager converts an argument to double-precision when you do not want it to, you will get incorrect results. If the function expects an alphanumeric string and the input is a numeric string, incorrect results will occur because of the conversion to double precision. To resolve this problem, append a non-numeric character to the end of the string, but do not count this extra character in the length of the argument. For example, to prevent the conversion of a delimiter blank character (' ') to a double precision zero in the GETTOK function, include a non-numeric character after the blank. GETTOK uses only the first character (the blank) as a delimiter and the extra character prevents conversion to double precision.

Assigning the Result of a Function to a Variable

You can store the result of a function in a variable. This is done with the -SET command, which Dialogue Manager uses to create variables.

Dialogue Manager variables contain only alphanumeric data. If a function returns a numeric value to a Dialogue Manager variable, the output is truncated to an integer and converted to a character string before being stored in the variable.

Note: You cannot specify a Dialogue Manager amper variable for the output argument unless you use the .EVAL suffix.

Syntax How to Store the Result of a Function in a Variable

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

where:

variable
Is the amper variable to which the returned value will be assigned.

function
Is the function.

input1
Is the first argument.

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

.LENGTH
Tests for the length. If a function requires the length of a character string as an input argument, you may prompt for the character string, and test the length.

Example Preventing Conversion to Double Precision Format

In the following Dialogue Manager command, GETTOK extracts the third word from a sentence stored in the &SN variable. The .LENGTH suffix passes the number of characters in the sentence to the function. The extra character (%) is included to prevent the conversion of a delimiter blank character to a double precision zero.

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

Example Using a Function in a -SET Command

In this example, the AYMD function, adds 14 days to the value of &INDATE. The &INDATE variable for the input date is previously set in the procedure and is 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. Although the format (I6) indicates that the output is an integer, it is stored in the &OUTDATE variable as a character string. For this reason, if you display the value of &OUTDATE, you will not see slashes separating the year, month, and day.

Using a Function in a -IF Command

You can use a function in the Dialogue Manager -IF command.

Note: If a branching command must span more than one line, you can continue on the next line by placing a dash in the first column.

Syntax How to Use a Function in a -IF Test

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

where:

function(args)
Is the function and its arguments.

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

expression
Is a valid relation or logical expression. Literals do not need to be enclosed in single quotation marks unless they contain commas or embedded blanks.

label1
Is a user-defined name of up to 12 characters. Do not use embedded blanks or the name of any other Dialogue Manager command except -QUIT or -EXIT. Do not use words that can be confused with functions, or arithmetic or logical operations.

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

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

Example Using a Function in a -IF Command

In the following example, the result of the AYMD function provides a condition for a -IF test. One of two requests is executed, depending on the result of the AYMD function.

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

This procedure processes as follows:

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

Using a Function in an Operating System -RUN Command

You can call a function with all alphanumeric arguments with the Dialogue Manager -CMS RUN, -TSO RUN, and -MVS RUN commands. These functions perform specific tasks but typically do not return any values.

All numeric arguments in Dialogue Manager are stored in alphanumeric format and require conversion before being passed to functions because unlike the -SET command, operating system -RUN commands do not automatically convert numeric arguments to double precision. For functions that require arguments in numeric format, you must first convert the arguments into double-precision numbers using the ATODBL function.

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

Syntax How to Use a Function in a RUN Command

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

where:

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 alphanumeric literals in single quotation marks.

,&output

Is a Dialogue Manager variable. Include this 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 requires an output argument that is eight bytes long, you need to define the variable with eight characters enclosed in single quotation marks before the function call:

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

Example Using a Function in a RUN Command

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

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

or

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

Using a Function in Another Function

A function can serve as an argument for another function.

Example **Using a Function in Another Function**

The command

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

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

Using a Function in WHERE or IF Criteria

A function may be used in WHERE or IF criteria. When this is done, the output value of the function is compared against the test value.

Example **Using a Function With a WHERE Test**

In this example, the SUBSTR function extracts the first two characters as a substring. The request prints an employee's name and salary if the result of the function 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 Function in WHEN Criteria

A function may be used as WHEN criteria as part of a Boolean expression.

Example

Using a Function in WHEN Criteria

The following example checks the values in LAST_NAME against the result of the CHKFMT function. When a match does not occur, a subfoot is printed.

```
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
```

Using a Function in a RECAP Command

You can use a function in a Financial Modeling Language (FML) RECAP command.

Syntax

How to Use a Function Call in a RECAP Command

```
RECAP name[(n)|(n,m)|(n,m,i)][/format] = 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 at 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 specified as 1, *m* is specified as 5, and *i* is specified as 2, the value will display in columns 1, 3, and 5.

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

function
: Is the function.

input1,...
: Are the arguments. The input arguments for a RECAP command 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 function output value. If the calculation consists of only the function, make sure this format agrees with the calculation's format. If the calculation format is larger than the column width, the value displays in that column as asterisks.

Example **Using a Function in a RECAP Command**

The following 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 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
```

Accessing a Function

Many of the functions are built in and do not require any additional work to access them.

Some functions are stored externally in load libraries. The way these functions are accessed is determined by your platform. The following topics describe how to access Information Builders-supplied functions on specific platforms.

You can also access private site-written functions. If you have a private collection of functions (that is, you created your own or use customized functions), do not store them in the function library. Store them separately to avoid overwriting them whenever your site installs a new release.

Storing and Accessing a Function on OS/390

In OS/390, load libraries are partitioned data sets containing link-edited modules. These load libraries are stored as part of EDALIB.LOAD or FUSELIB.LOAD. In addition to this load library, your site may have private function collections stored in separate load libraries.

OS/390 Batch Allocation

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

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

Example Allocating Load Libraries on OS/390

The following example allocates functions stored in BIGLIB.LOAD in JCL:

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

TSO Allocation

To use external functions in TSO, allocate the load libraries to ddname USERLIB using the ALLOCATE command. The ALLOCATE command can be issued:

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

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

Syntax How to Allocate a Load Library

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

where:

`MVS|TSO`

Is the prefix. Specify 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 function load libraries.

`lib1 lib lib3...`

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

Example Allocating the FUSELIB.LOAD Load Library

The following commands allocate the FUSELIB.LOAD load library.

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

or

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

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 that you specified them in the ALLOCATE command.

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

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

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

Storing and Accessing a Function on UNIX

No extra work is required.

Storing and Accessing a Function on VM/CMS

In VM/CMS, functions are stored as:

- The load library FUSELIB LOADLIB. In addition to the FUSELIB load library, your site may have private collections of functions stored in separate libraries or text files. If you create your own function in a text file or text library, the function must be 31-bit addressable and created as part of a LOADLIB.
- The text library FUSELIB TXTLIB. A text library is a file that 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. The file 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 (ID):

  ```
  EXCHANGE TEXT
  ```

Accessing a Function Automatically

For a function stored as a text file in VM/CMS, the access method is automatic. When your request calls the function, the attached disks are searched in alphabetical order, provided that you have proper authorization.

Reference

Search Sequence in VM/CMS

Functions are searched for in the standard VM/CMS search sequence:

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

Searching for a Function Library

For functions stored in a load or text library in VM/CMS, you need to issue the CMS GLOBAL command. The GLOBAL command enables your application to search specified libraries for the functions. 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 and PL/1, and for a function that calls system functions. 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 function collection, you need to specify the libraries in the GLOBAL command in addition to the FUSELIB load library.

Syntax **How to Enable Your Application to Search a Specified Library**

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

where:

`CMS`
Is required if you issue the GLOBAL command from a profile or procedure, or if you include it in a profile.

`LOADLIB`
Indicates the library is a load library.

`TXTLIB`
Indicates the library is a text library.

`library1 library2 library3...`
Are the file names of the load and text libraries containing the functions. The maximum number of libraries is 63.

Syntax **How to List 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 **Accessing a Library With the GLOBAL Command**

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

```
CMS GLOBAL LOADLIB FUSELIB
```

Example **Accessing Multiple Libraries With the GLOBAL Command**

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

```
CMS GLOBAL TXTLIB SUBLIB BIGLIB
```

Adding and Deleting a Subroutine Library

The GLOBAL library list automatically contains the FUSELIB subroutine library. If you need to add or delete private subroutine libraries you can use two CMS EXECs, FOCADLIB and FOCDELIB.

Before adding LOADLIBs to the GLOBAL 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 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. The EXECs FOCADLIB and FOCDELIB must be found in the CMS search sequence (A-Z).

Syntax How to Add and Delete a Subroutine 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.

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

Dynamic Language Environment Support

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

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

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

Syntax How to Control the LE Run-Time Environment

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

where:

`OFF`

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

`ON`

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

`ALL`

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

CHAPTER 3

Character Functions

Topics:

- Alphabetical List of Character Functions

Character functions manipulate alphanumeric fields and character strings.

ARGLEN: Measuring the Length of a String

Available Operating Systems: All

Available Languages: reporting, Maintain

The ARGLEN function measures the length of a character string within a field, excluding trailing spaces. The field format specifies the length of the 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 String

```
ARGLEN(inlength, infield, outfield)
```

where:

inlength

Integer

Is the length of the field containing the character string.

infield

Alphanumeric

Is the name of the field containing the character string.

outfield

Integer

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

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

Example Measuring the Length of a String

In the following example, 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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, Windows NT/2000

Available Languages: reporting

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

Syntax How to Distinguish Between a Space and a Zero

```
ASIS(argument)
```

where:

argument

Alphanumeric

Is the value to evaluate. You may 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 the ASIS function. No difference is detected between variables defined as space and 0.

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

The output is:

```
VAR1   EQ VAR2 0 TRUE
```

The next request uses ASIS to distinguish between the two variables.

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

The output is:

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

BITSON: Determining If a Bit is On or Off

Available Operating Systems: All

Available Languages: reporting, Maintain

The BITSON function evaluates an individual bit within a character string to determine whether it is on or off. If the bit is on, the function 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.

The result of the BITSON function varies between operating systems.

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

outfield

Integer or Alphanumeric

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

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

Example Evaluating a Bit in a Field

In this request, 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 a Binary Integer

Available Operating Systems: All

Available Languages: reporting, Maintain

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

Note: The result of the BITVAL function differs between operating systems.

Syntax How to Evaluate a Bit String

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

where:

string

Alphanumeric

Is the string. This can be the character string enclosed in single quotation marks, or the field that contains the 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 integer equivalent, or the format of the output value enclosed in single quotation marks.

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

Example Evaluating a Bit String

In this example, BITVAL evaluates the bits 12 through 20 of LAST_NAME and stores the result in a field with the format I5.

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

The output is:

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

BYTVAL: Translating a Character to a Decimal Value

Available Operating Systems: All

Available Languages: reporting, Maintain

The BYTVAL function translates a character to the ASCII or EBCDIC decimal value that represents it.

Syntax How to Translate a Character

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

where:

character

Alphanumeric

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

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.

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

Example Translating the First Character of a Field

In this example, BYTVAL translates the first character of LAST_NAME into its ASCII or EBCDIC decimal value, and stores the result in LAST_INIT_CODE. Since the input string has more than one character, BYTVAL evaluates the first one.

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

The output on an ASCII platform is:

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

The output on an EBCDIC platform is:

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

CHKFMT: Checking the Format of a String

Available Operating Systems: All

Available Languages: reporting, Maintain

The CHKFMT function checks a character string for incorrect characters or character types. It compares each string to a second string, called a mask, comparing each character in the first string to the corresponding character in the mask. If all characters in the 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 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 String

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

where:

numchar

Integer

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

string

Alphanumeric

Is the character string to be checked. This can be the character string enclosed in single quotation marks, or the field 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	Any of the letters A-Z (uppercase or lowercase).
9	Any of the digits 0-9.
X	Any of the letters A-Z or digits 0-9.
$	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 or Alphanumeric

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

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

Example Checking the Format of a Field

In this example, CHKFMT examines the EMP_ID field to see if it has nine numeric characters starting with 11, and stores the result in CHK_ID.

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

The output is:

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

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

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

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

A sample execution is:

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

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

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

The procedure processes as follows:

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

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

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

CTRAN: Translating One Character to Another

Available Operating Systems: All

Available Languages: reporting, Maintain

The CTRAN function translates a character within a 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 this function, you need to know the decimal equivalent of the characters in internal machine representation. Printable EBCDIC or ASCII characters and their decimal equivalents are listed in character charts.

Syntax How to Translate One Character to Another

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

where:

charlen

Integer

Is the length in characters of the input string.

string

Alphanumeric

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

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

Example Translating Spaces to Underscores on an ASCII Platform

In this example, CTRAN translates the spaces in ADDRESS_LN3 (ASCII decimal value 32) to underscores (ASCII decimal value 95), and stores the result in ALT_ADDR.

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

The output is:

```
EMP_ID     ADDRESS_LN3           ALT_ADDR
------     -----------           --------
117593129  RUTHERFORD NJ 07073   RUTHERFORD_NJ_07073_
119265415  NEW YORK NY 10039     NEW_YORK_NY_10039___
119329144  FREEPORT NY 11520     FREEPORT_NY_11520___
123764317  NEW YORK NY 10001     NEW_YORK_NY_10001___
126724188  FREEPORT NY 11520     FREEPORT_NY_11520___
451123478  ROSELAND NJ 07068     ROSELAND_NJ_07068___
543729165  JERSEY CITY NJ 07300  JERSEY_CITY_NJ_07300
818692173  FLUSHING NY 11354     FLUSHING_NY_11354___
```

Example Translating Spaces to Underscores on an EBCDIC Platform

In this example, 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, 95, ALT_ADDR);
BY EMP_ID
WHERE TYPE EQ 'HSM'
END
```

The output is:

```
EMP_ID     ADDRESS_LN3           ALT_ADDR
------     -----------           --------
117593129  RUTHERFORD NJ 07073   RUTHERFORD_NJ_07073_
119265415  NEW YORK NY 10039     NEW_YORK_NY_10039___
119329144  FREEPORT NY 11520     FREEPORT_NY_11520___
123764317  NEW YORK NY 10001     NEW_YORK_NY_10001___
126724188  FREEPORT NY 11520     FREEPORT_NY_11520___
451123478  ROSELAND NJ 07068     ROSELAND_NJ_07068___
543729165  JERSEY CITY NJ 07300  JERSEY_CITY_NJ_07300
818692173  FLUSHING NY 11354     FLUSHING_NY_11354___
```

Example 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 unusual 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 will appear. 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. The CTRAN function 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. The CTRAN function enables you to enter commas in names without having to enclose the names in single quotation marks. Instead of typing the comma, you type a semicolon, which is converted by the CTRAN function 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. The CTRAN function converts the semicolon in “BRADLEY; JR.” to a comma. The last name is now “BRADLEY, JR.”
4. The request adds the transaction to the data source.

This request displays the semicolon converted to a comma:

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

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

CTRFLD: Centering a Character String

Available Operating Systems: All

Available Languages: reporting, Maintain

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

The CTRFLD function 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 the CTRFLD function.

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. This can be the string enclosed in single quotation marks, or the name of the field that contains the string.

length

Integer

Is the length of *string* and *outfield*. 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.

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

Example

Centering a Field

In this example, CTRFLD centers the LAST_NAME field, and stores the results in CENTER_NAME.

```
SET STYLE=OFF

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

The output is:

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

EDIT: Extracting or Adding Characters

Available Operating Systems: All

Available Languages: reporting

The EDIT function extracts characters from or adds characters to an alphanumeric string. Another way to extract a substring is to use the SUBSTR function. The differences are:

- The EDIT function can extract a substring from different parts of the parent string. For example, it can extract the first two characters and the last two characters of a string to form a single substring. Also, it can insert characters into a substring.
- The SUBSTR function can vary the position of the substring depending on the values of other fields.

The EDIT function can also convert the format of a field. For information on converting a field with EDIT, see Chapter 6, *Format Conversion Functions*.

Syntax

How to Extract or Add Characters

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

where:

fieldname

Alphanumeric

Is the source field.

mask

Alphanumeric

Is a string, enclosed in single quotation marks. When EDIT encounters a 9 in the mask, it 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.

The length of the mask, excluding any characters other than 9 and $, must be the length of the source field.

Example Extracting and Adding a Character to a Field

In this example, EDIT extracts the first initial from the FIRST_NAME field, and stores the result in the FIRST_INIT field. EDIT also adds dashed to the EMP_ID field, and stores the result in the EMPIDEDIT field.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
FIRST_INIT/A1 = EDIT(FIRST_NAME, '9$$$$$$$$$');
EMPIDEDIT/A11 = EDIT(EMP_ID, '999-99-9999');
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

```
LAST_NAME        FIRST_INIT  EMPIDEDIT
---------        ----------  ---------
SMITH            M           112-84-7612
JONES            D           117-59-3129
MCCOY            J           219-98-4371
BLACKWOOD        R           326-17-9357
GREENSPAN        M           543-72-9165
CROSS            B           818-69-2173
```

GETTOK: Extracting a Substring (Token)

Available Operating Systems: All

Available Languages: reporting, Maintain

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

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

Syntax

How to Extract a Substring (Token)

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

where:

infield

Alphanumeric

Is the field containing the parent character string.

inlen

Integer

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

token

Integer

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

'delim'

Alphanumeric

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

Tip:

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.

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

Example Extracting a Token From a Field

In this example, 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
```

LCWORD: Converting a String to Mixed Case

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The LCWORD function converts the letters in a 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 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.

Syntax

How to Convert to Mixed Case

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

where:

length

Integer

Is the length of the field to be converted.

string

Alphanumeric

Is the string to be converted. This can be the name of the field containing the string, or the string enclosed in single quotation marks.

outfield

Alphanumeric

Is the name of the output field, or the format of the output value enclosed in single quotation marks. The length must be at least the length of *length*.

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

Example

Converting a String to Mixed Case

In the following, 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 String

Available Operating Systems: All

Available Languages: reporting, Maintain

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.

Syntax How to Left-Justify a String

```
LJUST(length, string, outfield)
```

where:

length

Integer

Is the length of *string* and *outfield*.

string

Alphanumeric

Is the string to be justified. This can be the field that contains the string, or the string enclosed in single quotation marks.

outfield

Alphanumeric

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

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

Example Left-Justifying a Formerly Numeric Field

In this example, FTOA converts CURR_SAL to an alphanumeric field called SAL_STRING. LJUST then left-justifies the SAL_STRING field and stores the result in LEFT_SAL.

```
SET STYLE=OFF

TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND COMPUTE
SAL_STRING/A12 = FTOA(CURR_SAL, '(D8.2M)', SAL_STRING);
LEFT_SAL/A12 = LJUST(12, SAL_STRING, LEFT_SAL);
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

```
LAST_NAME       FIRST_NAME  SAL_STRING    LEFT_SAL
---------       ----------  ----------    --------
BLACKWOOD       ROSEMARIE   $21,780.00    $21,780.00
CROSS           BARBARA     $27,062.00    $27,062.00
GREENSPAN       MARY         $9,000.00    $9,000.00
JONES           DIANE       $18,480.00    $18,480.00
MCCOY           JOHN        $18,480.00    $18,480.00
SMITH           MARY        $13,200.00    $13,200.00
```

LOCASE: Converting Text to Lowercase

Available Operating Systems: All

Available Languages: reporting, Maintain

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

Syntax How to Convert Text to Lowercase

```
LOCASE(length, string, outfield)
```

where:

length

Integer

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

string

Alphanumeric

Is the string to be converted. This can be the field that contains the string, or the string enclosed in single quotation marks.

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*.

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

Example Converting a Field to Lowercase

In this example, 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 Substring Within a String

Available Operating Systems: All

Available Languages: reporting, Maintain

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

Syntax

How to Overlay a Substring

OVRLAY(*string1*, *stringlen*, *string2*, *sublen*, *position*, *outfield*)

where:

string

Alphanumeric

Is the character string into which you want to overlay characters.

stringlen

Integer

Is the length of *string1* and *outfield*. If this argument is less than or equal to 0, unpredictable results occur.

string2

Alphanumeric

Is the string you want to overlay into *string1*.

sublen

Integer

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

position

Integer

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

outfield

Alphanumeric

Is the name of the field to which the overlaid string is returned, 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.

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

Example Replacing Characters in a String

In the following example, 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 Replacing Characters 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. The procedure is:

```
MODIFY FILE EMPLOYEE
CRTFORM LOWER
  "ENTER EMPLOYEE'S ID: <EMP_ID"
  "ENTER FIRST_NAME IN LOWER CASE: <FIRST_NAME"
MATCH EMP_ID
 ON NOMATCH REJECT
 ON MATCH COMPUTE
  F_UP/A1  = UPCASE (1, FIRST_NAME,'A1');
  FIRST_NAME/A10 = OVRLAY (FIRST_NAME, 10, F_UP, 1, 1, 'A10');
  ON MATCH TYPE "CHANGING FIRST NAME TO <FIRST_NAME "
  ON MATCH UPDATE FIRST_NAME
DATA
END
```

The COMPUTE command invokes two functions:

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

A sample execution is:

```
ENTER EMPLOYEE'S ID:  071382660
ENTER FIRST_NAME IN LOWER CASE:  alfred
```

The procedure processes as:

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

```
EMPLOYEE'S ID: 071382660
FIRST NAME:    alfred
```

2. The procedure searches the data source for the ID 071382660. If it finds the ID, it continues processing the transaction. In this case, the ID exists and belongs to Alfred Stevens.
3. The UPCASE function extracts the letter a from alfred and converts it to the letter A.
4. The OVRLAY function overlays the letter A on alfred. The first name is now Alfred.

```
ENTER EMPLOYEE'S ID:
ENTER FIRST_NAME IN LOWER CASE:

CHANGING FIRST NAME TO Alfred
```

5. The procedure updates the first name in the data source.
6. When you exit the procedure with PF3, the FOCUS transaction message indicates that one update occurred.

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

PARAG: Dividing Text Into Smaller Lines

Available Operating Systems: All

Available Languages: reporting, Maintain

The PARAG function divides lines of text into smaller lines by marking them off with a delimiter character. The PARAG function scans a specific number of characters from the beginning of the line and replaces the last space with a delimiter. It repeats this until reaching the end of the line. Each group of characters marked off by the delimiter becomes a subline. The GETTOK function can then place the sublines 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, be sure that no word of text is longer than the number of characters scanned by the function.

If the input lines of text are roughly equal in length, you can keep the sublines equal by specifying a subline length that evenly divides into the length of the text lines. For example, if you are dividing text lines 120 characters long, you can divide each of them into two sublines of 60 characters long, three sublines of 40 characters long, and so on. This enables you to print lines of text in paragraph form. However, if you divide the lines evenly, you may create more sublines than you intend. For example, suppose you divide 120-character text lines into two lines of 60 characters maximum length. One line is divided so that the first subline is 50 characters long and the second is 55. This leaves room for a third subline 15 characters long. To correct this, insert a space (using weak concatenation) at the beginning of the extra subline, then append this subline (using strong concatenation) to the end of the one before it.

Syntax How to Divide Text Into Smaller Lines

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

where:

length

Integer

Is the length of *string* and *outfield*.

string

Alphanumeric

Is the input string.

delim

Alphanumeric

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

subsize

Integer

Is the maximum length of each subline.

outfield

Alphanumeric

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

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

Example Dividing Text Into Smaller Lines

In the following example, PARAG divides ADDRESS_LN2 into smaller lines of not more than ten characters, using a comma as the delimiter. It then stores the result in PARA_ADDR.

```
TABLE FILE EMPLOYEE
PRINT ADDRESS_LN2 AND COMPUTE
PARA_ADDR/A20 = PARAG(20, ADDRESS_LN2, ',', 10, PARA_ADDR);
BY LAST_NAME
WHERE TYPE EQ 'HSM'
END
```

The output is:

```
LAST_NAME        ADDRESS_LN2           PARA_ADDR
---------        -----------           ---------
BANNING          APT 4C                APT 4C    ,
CROSS            147-15 NORTHERN BLD   147-15,NORTHERN,BLD
GREENSPAN        13 LINDEN AVE.        13 LINDEN,AVE.
IRVING           123 E 32 ST.          123 E 32,ST.       ,
JONES            235 MURRAY HIL PKWY   235 MURRAY,HIL PKWY
MCKNIGHT         117 HARRISON AVE.     117,HARRISON,AVE.
ROMANS           271 PRESIDENT ST.     271,PRESIDENT,ST.
SMITH            136 E 161 ST.         136 E 161,ST.
```

POSIT: Finding the Beginning of a Substring

Available Operating Systems: All

Available Languages: reporting, Maintain

The POSIT function finds the starting position of a substring within a larger string. For example, the beginning position of the substring DUCT in the string PRODUCTION is position 4. If the substring is not in the parent string, the function returns the value 0.

Syntax

How to Find the Beginning of a Substring

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

where:

`parent`

Alphanumeric

Is the field containing the parent character string.

`inlength`

Integer

Is the parent field length. If this argument is less than or equal to 0, the function returns 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 the *inlength* argument, the function returns a 0.

`outfield`

Integer

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

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

Example — Finding the Beginning of a String

In the following example, POSIT determines the position of the first capital letter I in LAST_NAME, and saves 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 String

Available Operating Systems: All

Available Languages: reporting, Maintain

The RJUST function right-justifies a character string within a field. All trailing spaces become leading spaces. This is helpful when you display alphanumeric fields containing numbers.

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

Syntax — How to Right-Justify a String

```
RJUST(length, string, outfield)
```

where:

length

Integer

Is the length of *string* and *outfield*. Their lengths must be the same to avoid justification problems.

string

Alphanumeric

Is the string to be justified. This can be the field that contains the string, or the string enclosed in single quotation marks.

outfield

Alphanumeric

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

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

Example Right-Justifying a Field

In the following example, RJUST right-justifies LAST_NAME 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 Strings Phonetically

Available Operating Systems: All

Available Languages: reporting, Maintain

The SOUNDEX function enables you to search for character strings phonetically without knowing how they are spelled. It converts character strings to 4-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 string.

To conduct a phonetic search, do the following:

1. Use the SOUNDEX function to translate data values from the field you are searching for to their phonetic codes.
2. Use the SOUNDEX function 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 Strings Phonetically

```
SOUNDEX(inlength, string, outfield)
```

where:

inlength

A2

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

string

Alphanumeric

Is the source of the input character string. It can be the character string itself enclosed in single quotation marks, or a field or amper variable that contains the string.

outfield

Alphanumeric

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

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

Example Comparing Strings Phonetically

The following request creates three fields:

- The field PHON_NAME contains the phonetic code of the employee's last name.
- The field PHON_COY contains the phonetic code of your guess, Micoy.
- The field PHON_MATCH contains YES if the phonetic code matches, NO if it does not.

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

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

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

The output is:

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

SQUEEZ: Reducing Multiple Blanks to a Single Blank

Available Operating Systems: All

Available Languages: reporting, Maintain

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

Syntax How to Reduce Multiple Blanks to a Single Blank

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

where:

length

Is a number or numeric field that specifies the length of the source and results fields.

string

Is an alphanumeric string or field from which the extra blank characters will be removed.

outfield

Alphanumeric

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

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

Example Reducing Multiple Blanks to a Single Blank

In the following example, SQUEEZ reduces multiple blanks in the NAME field to a single blank, and stores the result in a field with the format A30.

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

The output is:

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

STRIP: Removing a Character From a String

Available Operating Systems: All

Available Languages: reporting

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

Syntax How to Remove a Character From an Input String

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

where:

length

Integer

Is a number or numeric field that specifies the length of *string* and *outfield*.

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

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

Example Removing All Occurrences of a Character From a String

In the following example, 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:

```
DIRECTOR              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
```

SUBSTR: Extracting a Substring

Available Operating Systems: All

Available Languages: reporting, Maintain

The SUBSTR function extracts a substring based on where it begins and its length in the parent string. Another way to extract substrings is to use the EDIT function. The differences are:

- The EDIT function can extract a substring from different parts of the parent string. For example, it can extract the first two characters and the last two characters of a string to form a single substring. Also, it can insert characters into a substring.
- The SUBSTR subroutine can vary the position of the substring depending on the values of other fields.

Syntax How to Extract a Substring

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

where:

inlength

Integer

Is the length of the parent string.

parent

Alphanumeric

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

start
Integer

Is the starting position of the substring in the parent string. If this argument is less than 1, the 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 substring is returned, or the format of the output value enclosed in single quotation marks.

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

Example Extracting a String

In this example, POSIT determines the position of the first letter I in LAST_NAME and saves the result in I_IN_NAME. SUBSTR then extracts the three characters beginning with the letter I from I_IN_NAME, and saves the results in I_SUBSTR.

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE
    I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2');
    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
```

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

TRIM: Removing Leading and Trailing Occurrences

Available Operating Systems: All

Available Languages: reporting, Maintain

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

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 string.

string_length

Integer

Is the length of the source string.

pattern

Alphanumeric

Is the pattern to remove.

pattern_length

Integer

Is the length of the pattern.

outfield

Alphanumeric

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

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

Example Removing Leading Occurrences

The following request uses the TRIM function to remove leading occurrences of the characters BR from director names in the MOVIES data source.

```
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

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

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

The output is:

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

UPCASE: Converting Text to Uppercase

Available Operating Systems: All

Available Languages: reporting, Maintain

The UPCASE function converts a string of characters to uppercase. This 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 EBCIDIC always places lowercase letters before uppercase letters and the ASCII sorting sequence always places uppercase letters before lowercase letters. 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 as they were typed. You can use the UPCASE function to convert entries for particular fields to uppercase.

Syntax How to Convert Text to Uppercase

```
UPCASE(length, input, outfield)
```

where:

length

Integer

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

input

Alphanumeric

Is the mixed-case input string or field.

outfield

Alphanumeric

Is the uppercase output string or field, or the format of the output value enclosed in single quotation marks.

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

Example Converting a Mixed Case Field to Uppercase

Suppose you are sorting on a field that contains both uppercase and mixed case values. The following request defines a field called LAST_NAME_MIXED that contains both uppercase and mixed case values:

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
  LCWORD (15 , LAST_NAME, 'A15');
END
```

Suppose you execute a request that sorts by this field:

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

On an EBCDIC-based platform, the output is:

```
LAST_NAME_MIXED  FIRST_NAME
---------------  ----------
Banning          JOHN
BLACKWOOD        ROSEMARIE
CROSS            BARBARA
Mcknight         ROGER
MCCOY            JOHN
Romans           ANTHONY
```

On an ASCII-based platform, the output is:

```
LAST_NAME_MIXED  FIRST_NAME
---------------  ----------
BLACKWOOD        ROSEMARIE
Banning          JOHN
CROSS            BARBARA
MCCOY            JOHN
Mcknight         ROGER
Romans           ANTHONY
```

In the first example, Mcknight appears before MCCOY, since the EBCDIC sorting order places lowercase letters before uppercase letters. In the second example, Blackwood appears before Banning, since the ASCII sorting order places uppercase letters before lowercase letters. In either case, this is not how you would expect your report to be sorted.

The solution is to create a new field with all uppercase letters and sort using this field:

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
  LCWORD (15, LAST_NAME, 'A15');
LAST_NAME_UPPER/A15=UPCASE (15, LAST_NAME_MIXED, 'A15') ;
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME_MIXED AND FIRST_NAME BY LAST_NAME_UPPER
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
END
```

Now, when you execute the request, the names are sorted correctly:

```
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 do not want to see the field with all uppercase values, you can NOPRINT it.

Example Converting a Mixed Case Field to Uppercase With MODIFY

Suppose your company decided to store employee names in mixed case and the department assignments in uppercase in the EMPLOYEE data source.

To enter records of new employees, execute this MODIFY procedure:

```
MODIFY FILE EMPLOYEE
CRTFORM LOWER
 "ENTER EMPLOYEE'S ID : <EMP_ID"
 "ENTER LAST_NAME: <LAST_NAME FIRST_NAME: <FIRST_NAME"
 "TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET"
 " "
 "ENTER DEPARTMENT ASSIGNMENT: <DEPARTMENT"
MATCH EMP_ID
     ON MATCH REJECT
     ON NOMATCH COMPUTE
        DEPARTMENT = UPCASE (10, DEPARTMENT, 'A10');
     ON NOMATCH INCLUDE
     ON NOMATCH TYPE "DEPARTMENT VALUE CHANGED TO UPPERCASE: <DEPARTMENT"
DATA
END
```

A sample execution is as follows:

```
ENTER EMPLOYEE'S ID :  444555666
ENTER LAST_NAME:  Cutter           FIRST_NAME:  Alan
TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET

ENTER DEPARTMENT ASSIGNMENT:  sales
```

The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department on a CRTFORM screen. The CRTFORM LOWER option retains the case of entries as they were typed.
2. You type the following data and press the Enter key:

EMPLOYEE'S ID:	444555666
LAST_NAME:	Cutter
FIRST_NAME:	Alan
DEPARTMENT ASSIGNMENT:	sales

3. The procedure searches the data source for the ID 444555666. If it does not find the ID, it continues processing the transaction.
4. The UPCASE function converts the DEPARTMENT entry sales to SALES.

   ```
   ENTER EMPLOYEE'S ID :
   ENTER LAST_NAME:                  FIRST_NAME:
   TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET

   ENTER DEPARTMENT ASSIGNMENT:

   DEPARTMENT VALUE CHANGED TO UPPERCASE: SALES
   ```

5. The procedure adds the transaction to the data source.
6. When you exit the procedure with PF3, the FOCUS transaction message indicates the number of transactions accepted or rejected.

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

CHAPTER 4

Data Source and Decoding Functions

Topics:

- Alphabetical List of Data Source and Decoding Functions

Data source and decoding functions search for data source records, retrieve data source records or values, and assign values.

DECODE: Decoding Values

Available Operating Systems: All

Available Languages: reporting, Maintain

The DECODE function assigns values based on the value of an input field.

This is helpful for giving a coded value in a field a more useful value. For example, the field SEX may have the code F for female employees and M for male employees. This allows the value for the field to be stored more efficiently (for example, one character instead of six for female), and reduces the storage requirement for the file. The DECODE function expands (decodes) these values.

You can use DECODE by typing values directly into the DECODE function or reading values from a separate file.

Syntax How to Decode Values Supplied in the DECODE Function

```
DECODE fieldname(code1 result1 code2 result2...[ELSE default ]);
```

where:

fieldname

Alphanumeric or Numeric

Is the name of the input field.

code

Any supported format

Is the code value DECODE is searching for; once it has found the specified value, it will assign the corresponding result. If the value has embedded blanks, commas, or other special characters, enclose the value in single quotation marks.

result

Any supported format

Is the value to be assigned when the field has the corresponding code. If the value has embedded blanks or commas or contains a negative number, enclose the value in single quotation marks.

default

Any supported format

Is the value to be assigned if the code is not found among the list of codes. If this value is omitted, DECODE will assign a blank or zero for non-matching codes.

Note: You can use up to 40 lines to define the code and result pairs for any given DECODE expression, or 39 if you also use an ELSE phrase. You can use either commas or blanks to separate the code from the result, or one pair from another.

Syntax

How to Decode Values in a Separate File

```
DECODE fieldname(ddname [ELSE default ]);
```

where:

fieldname

Alphanumeric or Numeric

Is the name of the input field.

ddname

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

default

Any supported format

Is the value to be assigned if the code is not found among the list of codes. If this default is omitted, DECODE will assign a blank or zero for non-matching codes.

Note:

- Each record in the separate file is expected to contain one pair of elements separated by a comma or blanks.
- All data is interpreted in ASCII format on UNIX and Windows, or in EBCDIC format on OS/390 or VM/CMS, and converted to the USAGE formats of the DECODE pairs.
- Leading and trailing blanks are ignored.
- The remainder of each record is ignored and can be used for comments or other data. This convention is followed in all cases, except when the file name is HOLD. In that case the file is presumed to have been created by the FOCUS HOLD command, which writes fields in their internal format, and the DECODE pairs are interpreted accordingly. In this case, extraneous data in the record is ignored.
- If each record in the file consists of only one element, this element is interpreted as the code, and the result becomes either 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 will be 0 for SELECT values found in the literal file and 1 in all other cases. The VALUE computation is carried out as if the expression had been:

  ```
  IF SELECT (filename) THEN... ELSE...;
  ```
- When using DECODE with a file, you can have up to 32,767 characters in the file.

Example Decoding Values Supplied in the DECODE Function

In the following example, EDIT extracts the first character of the CURR_JOBCODE field, then DECODE replaces these values with either ADMINISTRATIVE or DATA PROCESSING.

```
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
```

Example Reading DECODE Values From a File

The following example has two parts. The first part creates a file with a list of the employee IDs for the employees who have taken classes. The second part reads this file and assigns 0 to those employees who have taken classes and 1 to those employees who have not. (Notice that the HOLD file contains only one column of values; therefore DECODE assigns the value 0 to an employee 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 an Indexed Field

Available Operating Systems: All

Available Languages: MODIFY, Maintain

The FIND function verifies if an incoming data value is in an indexed data source field, whether the field is in the data source you are modifying, or if it is in another data source. The function sets a temporary field to a non-zero value if the incoming value is in the data source field, and to 0 if it is not. A value greater than zero confirms the presence of the data value, not the number of instances in the data source field.

The FIND function can also be used in a VALIDATE command to test 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 to fail and the request to reject the transaction.

You can use any number of FIND functions in a COMPUTE or VALIDATE command. However, more FIND functions increase processing time and require more buffer space in memory.

Limit:

The FIND function does not work on files with different DBA passwords.

Syntax How to Verify the Existence of an Indexed Field

```
field = FIND(fieldname [AS dbfield] IN file;
```

where:

field

Is the name of the temporary field to which the result is returned.

fieldname

Is the full field name of the incoming field being tested.

AS *dbfield*

Is the full field name of the data source field containing values to be compared with the incoming data field. This field must be indexed. If the incoming field and the data source field have the same name, you can omit this phrase.

file

Is the name of the data source.

Note: There is no space between FIND and the left parenthesis.

Example Verifying the Existence of an Indexed Field in Another File

The following tests if each employee ID entered is also in the EDUCFILE data source. It then displays a message informing you whether it found the ID in the data source.

```
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
```

Example Using the FIND Function in a VALIDATE Command

The following updates the number of hours employees spent in class. It rejects employees not listed in the EDUCFILE data source, which records class attendance.

This VALIDATE command will discard any incoming EMP_ID value not found in the EDUCFILE data source.

```
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
```

LAST: Retrieving the Preceding Value

Available Operating Systems: All

Available Languages: reporting

The LAST function retrieves the preceding value selected for a field.

The effect of the keyword LAST depends on whether it appears in a DEFINE or COMPUTE.

- In a DEFINE command, the LAST value is the previous record retrieved from the file before sorting takes place.
- In a COMPUTE command, the LAST value is the record in the previous line in the report.

Limit:

LAST cannot be used 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

In the following example, LAST retrieves the previous value of the DEPARTMENT field to determine whether to restart the running total of salaries by 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 File

Available Operating Systems: All

Available Languages: MODIFY

The LOOKUP function retrieves data values from cross-referenced files in a MODIFY request. You can retrieve data from a file cross-referenced statically in the Master File or a file joined dynamically by the JOIN command. The LOOKUP function is necessary because unlike TABLE requests, MODIFY requests cannot read cross-referenced files freely. The LOOKUP function allows a request to use the data in computations and in messages, but not modify a cross-referenced file; to modify more than one file in one request, use the COMBINE command or the Maintain facility.

The LOOKUP function can read cross-referenced segments that are linked directly to a segment in the host data source (the host segment). This means that the cross-referenced segments must have segment types of KU, KM, DKU, or DKM (but not KL or KLU) or 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 which the LOOKUP function uses:

- The field containing the values you want. Alternatively, you can retrieve all of the fields in the segment at one time. The field, or your decision to retrieve all the segment's fields, is specified in the LOOKUP function.

  For example, this LOOKUP function retrieves a single value from the DATE_ATTEND field:

  ```
  RTN = LOOKUP(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. The LOOKUP function uses the cross-referenced field, which is indexed, to locate a specific segment instance.

When using the LOOKUP function, the MODIFY request reads a transaction value for the host field. The LOOKUP function 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 the LOOKUP function 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 one 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, DKM, or if you specified the ALL keyword in the JOIN command

Syntax How to Read Cross-Referenced FOCUS Files

```
rcode = LOOKUP(field)
```

where:

rcode

Is a variable you specify to receive a return code value. The value returned is 1 if the LOOKUP function can locate a cross-referenced segment instance, 0 if the function cannot.

field

Is the name of the field that you want to retrieve in the cross-referenced file. If the field name also exists in the host file, you must qualify it here.

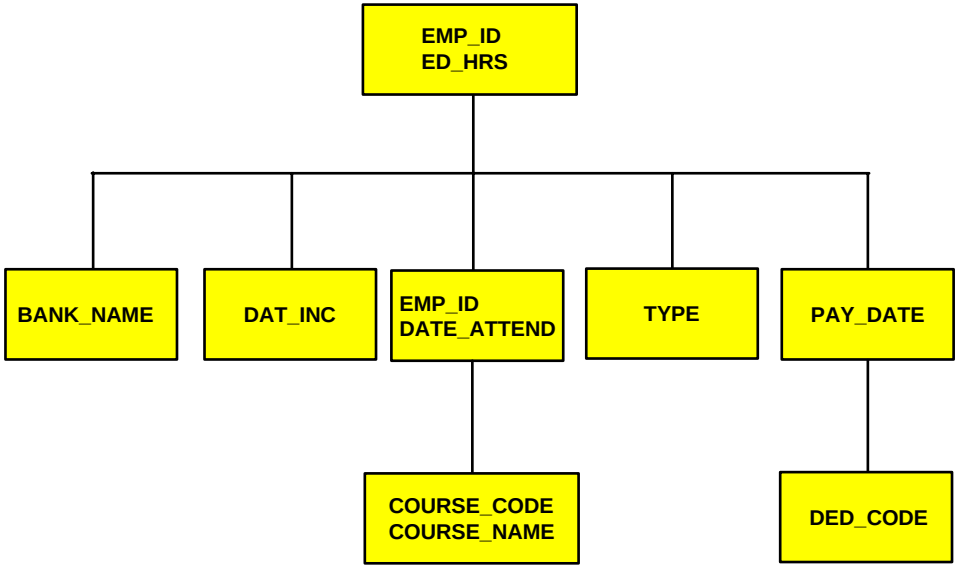
Note: No spaces are permitted between LOOKUP and the left parenthesis.

Example Reading Cross-Referenced FOCUS Files

Suppose you wish to update the amount of classroom hours employees have spent. Because of a new system of accounting, employees taking classes after January 1, 1985 are to be credited with 10% more classroom hours than their records indicate.

The employee IDs (EMP_ID) and classroom hours (ED_HRS) are located in the host segment. The class dates (DATE_ATTEND) are located in the cross-referenced segment. The shared field is the employee ID field.

The file structure is shown in this diagram:



The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID ED_HRS
COMPUTE
    EDTEST  = LOOKUP(DATE_ATTEND);
    COMPUTE
    ED_HRS =    IF DATE_ATTEND 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 of this request might go as follows:

1. The request prompts you for an employee ID and number of class hours. You enter the ID 117593129 and 10 class hours.
2. The LOOKUP function 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 class date as 821028 (October 28, 1982).
3. The LOOKUP function retrieves the value 821028 for the DATE_ATTEND field.
4. The COMPUTE statement tests the value of the DATE_ATTEND field. Since October 28, 1982 is after January 1, 1982, the statement increases the incoming ED_HRS value from 10 to 11 hours.
5. The request updates the classroom hours for employee 117593129 using the new ED_HRS value.

Example Using a Data Source Value in a Segment to Search a File

You may use a data source value in a specific host segment instance to search the cross-referenced segment. To do this, prepare the request this way:

- In the MATCH statement that selects the host segment instance, activate the host field. This can be done with the ACTIVATE phrase.
- In the same MATCH statement, place the LOOKUP function after the ACTIVATE phrase.

This request displays the employee IDs, dates of salary raises, employee names, and the position each employee held after the raise was granted:

- The employee IDs and names (EMP_ID) are in the root segment.
- The date of raise (DAT_INC) is in the descendant host segment.
- The job titles are in the cross-referenced segment.
- The shared field is JOBCODE. You never enter any job codes; the values are all 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 might execute as follows:

1. The request prompts you for an employee ID and date of pay raise. You enter employee ID 071382660 and date of raise 820101 (January 1, 1982).
2. The request locates the instance containing the ID 071382660, then locates the child instance containing the date of raise 820101.
3. This child instance contains the job code A07. The ACTIVATE statement activates this value, making it available to the LOOKUP function.

4. The LOOKUP function locates the job code A07 in the cross-referenced segment. It returns a 1 into the RTN variable and retrieves the corresponding job description of SECRETARY.
5. The request displays the values using a TYPE statement:

```
EMPLOYEE ID:       071382660
DATE INCREASE:     82/01/01
NAME:              ALFRED STEVENS
POSITION:          SECRETARY
```

Note: You may also need to activate the host field if you are using the LOOKUP function within a NEXT statement. This request, similar to the previous one except for the NEXT statement, displays the latest position held by a particular 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 in a VALIDATE Command

When you use the LOOKUP function, you may want to 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 statement. If the function cannot locate the instance in the cross-referenced segment, it sets the value of the return variable to 0. This causes the request to reject the transaction.

The following request updates an employee's classroom hours (ED_HRS). If the employee attended classes on or after January 1, 1982, the request increases the number of classroom hours by 10%. The classroom attendance dates are stored in a cross-referenced segment (field DATE_ATTEND). The shared field is the employee ID.

The request is as follows:

```
MODIFY FIELD EMPLOYEE
PROMPT EMP_ID ED_HRS
VALIDATE
    TEST_DATE = LOOKUP(DATE_ATTEND):
COMPUTE
    ED_HRS =   IF DATE_ATTEND GE 820101 THEN ED_HRS * 1.1
           ELSE ED_HRS;
MATCH EMP_ID
    ON MATCH UPDATE ED_HRS
    ON NOMATCH REJECT
DATA
```

If the employee is not recorded in the cross-referenced segment, then the employee has never attended a class. This means that a transaction recording the employee's classroom hours is an error and should be rejected.

This is the purpose of the LOOKUP function in the VALIDATE statement. If the function cannot locate an employee's record in the cross-referenced segment, it returns a 0 to the TEST_DATE field. This causes the request to reject the transaction.

Using the Extended LOOKUP Function

If the function cannot locate a value of the host field in the cross-referenced segment, you may specify that the LOOKUP function locates the next highest or lowest cross-referenced field value in the cross-referenced segment by using an extended syntax.

To use this LOOKUP feature, the index must have been created on FOCUS Release 4.5 or later with the INDEX parameter set to NEW (the binary tree scheme). To determine what type of index your file uses, enter the ? FDT command.

Note that fields retrieved by the LOOKUP function do not require the D. prefix to be displayed in TYPE statements. FOCUS treats the field values as transaction values that are not active.

Syntax How to Use the Extended LOOKUP Function

```
COMPUTE
rcode = LOOKUP(field action)
```

where:

rcode

Is a variable you specify to receive a return code value. (The value the variable receives depends on the outcome of the function below.)

field

Is the name of the field you want to use in MODIFY computations. If the field name also exists in the host file, you must qualify it here.

action

Specifies the action the request takes if there is no cross-referenced segment instance corresponding to the host field value. Valid actions are the following:

`EQ` causes the LOOKUP function 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 the LOOKUP function to locate the instance with the next highest value of the cross- referenced field. The value of *rcode* is set to 2.

`LE` causes the LOOKUP function to locate the instance with the next lowest value of the indexed field. The value of *rcode* is set to -2.

Note that there can be no space between LOOKUP and the left parenthesis.

The following table summarizes the value of *rcode*, depending on which instance the LOOKUP function 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 field value not located

CHAPTER 5

Date and Time Functions

Topics:

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 5-2.
- Legacy date functions for use with legacy dates. For more information see *Using Legacy Date Functions* on page 5-32.

Note: If you are have dates in alphanumeric or numeric fields that contain date display options, you must use legacy date functions.

Using Standard Date and Time Functions

When using standard date and time functions, you need to understand the settings that alter the behavior of these functions, as well as what formats are acceptable and how to supply values in these formats.

You can affect the behavior of date and time functions in the following ways:

- Define which days of the week are work days and which are not. Then, when you use a date function, dates that are not work days are ignored. For details see *Setting Business Days* on page 5-3.
- Determine whether to display leading zeros when a date function in Dialogue Manager returns a date with leading zeros. For details see *Enabling Leading Zeros For Date and Time Functions in Dialogue Manager* on page 5-5.

Reference

Component Names and Values for Use With Date-Time Functions

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

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

Notes:

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

Specifying Work Days

You can determine which days are work days and which are not. Work days affect the DATEADD, DATEDIF, and DATEMOV functions. You can specify work days in the following ways:

- Specifying business days. For details see *Setting Business Days* on page 5-3.
- Specifying holidays. For details see *Setting Holidays* on page 5-4.

Example Setting Business Days

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

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

Syntax How to Set Business Days

```
SET BUSDAYS = smtwtfs
```

where:

smtwtfs

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

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

If 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
```

Example Setting 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.

In order to define a list of holidays, you must:

1. Create a holiday file. You create a holiday file in a text editor.
2. Select the holiday file by issuing the SET command with the HDAY parameter in a report request.

Reference Rules for Creating a Holiday File

The following guidelines must be followed in order for the holiday file to work:

- Dates must be in YYMD format.
- Dates must be listed in chronological order.
- Each date must be on its own line.
- An optional description of the holiday may be included, separated from the date by a space.
- Each year for which data exists must be represented in the holiday file. Calling a date function with a date value outside the range of the holidays file returns a zero on business day requests.

Procedure How to Create a Holiday File

1. In a text editor, create a list of dates designated as holidays. For details on this file, see *Rules for Creating a Holiday File* on page 5-3.
2. Save the file:
   - In OS/390 this file should be a member in ERRORs called HDAY*xxxx*.
   - In CMS the list should be HDAY*xxxx* ERRORS.

   where:

   xxxx

   Is a string of text four characters long.

Example Creating a Holiday File

The following file establishes holidays:

```
19910325 TEST HOLIDAY
19911225 CHRISTMAS
```

Syntax How to Select the 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 Using a Holiday File

The following is the HDAYTEST file and establishes holidays:

```
19910325 TEST HOLIDAY
19911225 CHRISTMAS
```

The following request uses the HDAYTEST file 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
```

Syntax How to View the Current Setting of HDAY

```
? SET HDAY
```

Enabling Leading Zeros For Date and Time Functions in Dialogue Manager

If you use a date and time function in Dialogue Manager that returns a numeric integer format, Dialogue Manager truncates any leading zeros. For example, if your function returns the value 000101 (indicating January 1, 2000), Dialogue Manager will truncate the leading zeros and use 101, producing an incorrect date. To avoid this problem, you can use the LEADZERO parameter.

Syntax How to Set the Display of Leading Zeros

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

where:

ON
Allows the display of leading zeros.

OFF
Truncates leading zeros. This is the default.

Example Displaying Leading Zeros

The following request uses the AYM function to add one month to the input date of December 1999.

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

Using the default setting, this yields:

```
1
```

This represents the date of January 2000 incorrectly. Modifying the request by adding the LEADZERO parameter

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

results in the following:

```
0001
```

This correctly indicates January 2000.

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

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

will always truncate leading zeros.

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

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

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

- **Year.**
- **Month.** If your calculation using the month unit creates an invalid date, DATEADD corrects it by using 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 and Sunday. For example, if you add one day to a Friday, the result is Monday.
- **Business day.** When using the business day unit, DATEADD uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. This means that if Monday is not a working day, then one business day past a Sunday is Tuesday. See *Setting Business Days* on page 5-3 for more information.

The DATEADD function cannot be used with Dialogue Manager. DATEADD requires dates to be in date format; Dialogue Manager interprets a date as alphanumeric or numeric.

Note: You add or subtract non day-based dates (for example YM, YQ) directly without using DATEADD.

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

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

where:

date

Date

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

unit

Alphanumeric

Can be one of the following:

`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 you wish to add to or subtract from *date*. If this number is not a whole unit, it is rounded down to the next largest integer.

Example Rounding With DATEADD

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

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

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

Example Using Weekday Units

If you use weekday units and use a Saturday or Sunday as your date, DATEADD changes the day to Monday. The functions

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

and

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

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

Example Adding Days to a Date

In this example, three weekdays are added to HIRE_DATE. DATECVT converts HIRE_DATE to YYMD format and stores the result in NEW_DATE. DATEADD then adds three weekdays to NEW_DATE.

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

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
```

Note: In some cases, DATEADD added more than three days, because otherwise HIRE_DATE_PLUS_THREE would have been on a weekend.

Example Determining if a Date is a Business Day

In the following example, DATEADD determines which values in the TRANSDATE field of the VIDEOTRK data source do not represent business days.

DATEADD adds zero days to TRANSDATE using the business day unit. If TRANSDATE does not represent a business day, DATEADD returns the next business day, which is not the same as TRANSDATE.

```
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
```

DATECVT: Converting a Date Format

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The DATECVT function converts date formats within applications without requiring intermediate calculations. If an invalid format is supplied, DATECVT returns a zero or a blank.

Syntax How to Convert a Date Format

```
DATECVT(date, 'infmt', 'outfmt')
```

where:

date

Date

Is the date whose format you wish to change. If you supply an invalid date, DATECVT returns a zero value. When performing the conversion, an i*ndate* with an old format obeys any DEFCENT and YRTHRESH values supplied for that field.

infmt

Alphanumeric

Is one of the following:

- A new date format (for example, YYMD, YQ, M, DMY, JUL) that matches the format of *indate*. It can also be the format of the output value enclosed within single quotes.
- An old date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in the *infmt* parameter functions as an offset from the base date of a YYMD field (12/31/1900).

The format of the field for which the value is being calculated must have the same format as *outfmt*.

outfmt

Alphanumeric

Is one of the following:

- A new date format (for example, YYMD, YQ, M, DMY, JUL) that matches the format of *indate*. It can also be the format of the output value enclosed within single quotes.
- An old date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in the *outfmt* parameter receives an offset from the base date of a YYMD field (12/31/1900).

The format of the field for which the value is being calculated must have the same format as *outfmt*.

Example Converting a DMY Date to YYMD

For example,

```
field/DMY = DATECVT(indate, 'YYMD', 'DMY');
```

If the value of *indate* is 19991231 then *field* is set to the offset, which is 311299. *Indates* with old formats obey any DEFCENT and YRTHRESH values implied for that field when performing the conversion.

Example Converting a Field to Date Format

In this example, DATECVT converts HIRE_DATE from I6YMD format to dates formatted as YYMD.

```
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
```

DATECVT also supplies a century for HIRE_DATE according to the DEFCENT and YRTHRESH parameter settings.

DATEDIF: Finding the Difference Between Two Dates

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The DATEDIF function returns the difference between two dates in units. A unit can be any of the following:

- **Year.** Using the year unit with DATEDIF yields the inverse of DATEADD. If adding one year to date X creates date Y, then the count of years between date X and date Y must be one year. Note that adding one year to February 29 produces the date February 28.
- **Month.** Using the month unit with DATEDIF yields the inverse of DATEADD. If adding one month to date X creates date Y, then the count of months between date X and date Y must be one month. The rule is 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.
- **Day.**
- **Weekday.** If you use the weekday unit, DATEDIF does not count Saturday and Sunday when adding days. This means that the difference between a Friday and Monday is one day.
- **Business day.** When using 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 *Setting Business Days* on page 5-3 for more information.

DATEDIF returns a whole number. If the difference between two dates is not a whole number, DATEDIF rounds down to the next largest integer. For example, the number of years between March 2, 2001 and March 1, 2002 would be zero. If the ending date is before the starting date, DATEDIF returns a negative number.

If you use month units, and one or both of your input dates is the end of the month, DATEDIF takes this into account. This means that the difference between January 31 and April 30 is three months, not two months.

Note: You add or subtract non day-based dates (for example YM, YQ) directly without using DATEDIF.

Syntax How to Return the Difference Between Two Dates

```
DATEDIF(from_date, to_date, 'unit')
```

where:

from_date

Date

Is the starting date from which to calculate the difference.

to_date

Date

Is the ending 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.

Example Rounding With DATEDIF

The following expression

```
DATEDIF(19960302, 19970301, 'Y')
```

calculates the difference between March 2, 1996 and March 1, 1997. It returns a zero because the difference is less than a year.

Example Using Month Calculations

The following expressions

```
DATEDIF(19990228, 19990128, 'M')
DATEDIF(19990228, 19990129, 'M')
DATEDIF(19990228, 19990130, 'M')
DATEDIF(19990228, 19990131, 'M')
```

all return a result of minus one month.

Additional examples:

`DATEDIF(March31, May31, 'M')` yields 2.

`DATEDIF(March31, May30, 'M')` yields 1 (because May 30 is not the end of the month).

`DATEDIF(March31, April30, 'M')` yields 1.

Example Determining the Number of Weekdays Between Two Dates

In this example, DATEDIF determines the number of weekdays between the dates in 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
```

DATEMOV: Moving a Date to a Significant Point

Available Operating Systems: AS/400, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The DATEMOV function moves a date to a significant point on the calendar. DATEMOV works with date format only.

Syntax How to Move a Date to a Significant Point

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

where:

date

Date

Is the date you wish to move. This date can be any new date format as long as it includes a day component. For example, MDYY can be used but MYY cannot be.

move-point

Alphanumeric

Is the significant point to which you wish to move. An invalid point results in a zero being returned. Valid points to which to move the date 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.

An invalid point results in a zero being returned.

Note: When using a business day calculation, the result is affected by the days specified as working days.

Example Determining Significant Move Points for a Field

The following sets the business days to Monday, Tuesday, Wednesday, and Thursday. DATEMOV determines significant move 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 'WB+' AND
COMPUTE WBM/WT = DATEMOV(NEW_DATE, 'BD-'); AS 'WB-' BY LAST_NAME NOPRINT
HEADING
"Examples of DATEMOV"
"Business days are Monday, Tuesday, Wednesday, + Thursday "
" "
"START DATE..... | MOVE POINTS........................."
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
Examples of DATEMOV
Business days are Monday, Tuesday, Wednesday, + Thursday

START DATE.....| MOVE POINTS.........................
NEW_DATE    DOW  NWD  PWD  WD+  WD-  NBD  PBD  BD+  BD-
--------    ---  ---  ---  ---  ---  ---  ---  ---  ---
1982/04/01  WED  THU  TUE  WED  WED  SUN  TUE  WED  WED
1981/11/02  SUN  MON  THU  SUN  SUN  MON  WED  SUN  SUN
1982/04/01  WED  THU  TUE  WED  WED  SUN  TUE  WED  WED
1982/05/01  FRI  MON  WED  SUN  THU  MON  TUE  SUN  WED
1981/07/01  TUE  WED  MON  TUE  TUE  WED  MON  TUE  TUE
1981/07/01  TUE  WED  MON  TUE  TUE  WED  MON  TUE  TUE
```

Example Determining the End of the Week

In this example, DATEMOV determines the date for the end of the week for the dates in NEW_DATE, and stores the results in the EOW field.

```
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
```

HADD: Incrementing a Date-Time Field

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HADD function increments a date-time field by a given number of units.

Syntax How to Increment a Date-Time Field

```
HADD (dtfield, 'component', increment, length, 'format')
```

where:

dtfield

Is the date-time value to increment. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be incremented, enclosed in single quotation marks. For a list of these components see *Component Names and Values for Use With Date-Time Functions* on page 5-2.

increment

Is the number of units by which to increment the specified component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

length

Is the length of the returned date-time value. Valid values are:

8 for time values to include milliseconds.

10 for time values to include microseconds.

format

Is the format of the returned date-time value, enclosed in single quotation marks. The format must be a date-time format (data type H).

Example Incrementing the Month Component of a Date-Time Field

In the following, HADD adds two months to the values in the TRANSDATE field.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

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

If necessary, the day is adjusted to be valid for the resulting month.

HCNVRT: Converting a Date-Time Field to Alphanumeric Format

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HCNVRT function converts a date-time field to alphanumeric format for use with operators such as EDIT, CONTAINS, and LIKE.

Syntax How to Convert a Date-Time Field to Alphanumeric Format

```
HCNVRT (value, '(fmt)', length, 'outputfmt')
```

where:

value

Is the date-time value to convert. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

fmt

Is the USAGE format of the date-time field being converted, enclosed in parentheses and single quotation marks. The format must be a date-time format (data type H).

length

Is the length of the alphanumeric field being 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, a blank field is returned.

outputfmt

Alphanumeric

Is the format of the returned alphanumeric value, enclosed in single quotation marks.

Example Converting a Date-Time Field to Alphanumeric Format

In the following, HCNVRT converts the TRANSDATE field to alphanumeric format.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME1/A20 = HCNVRT (TRANSDATE,'(H17)', 17, 'A20');
ALPHA_DATE_TIME2/A20 = HCNVRT (TRANSDATE,'(HYYMDS)', 20, 'A20');
WHERE DATE EQ 2000
END
```

The output is:

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

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HDATE function extracts the date portion of a date-time field, converts it to a date format, and returns the result in the format YYMD. The result can then be converted to other date formats.

Syntax How to Convert the Date Portion of a Date-Time Field to a Date Format

```
HDATE (dtfield, 'YYMD')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

Example Converting the Date Portion of a Field to a Date Format

In the following, HDATE converts the date portion of the TRANSDATE field to the date format YYMD.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
WHERE DATE EQ 2000
END
```

The output is:

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

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HDIFF function calculates the number of boundaries of a given type crossed between two dates.

Syntax How to Find the Number of Units Between Two Date-Time Values

```
HDIFF (dtvalue1, dtvalue2, 'component', 'format')
```

where:

dtvalue1

Is the ending date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

dtvalue2

Is the starting date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be used in the calculation, enclosed in single quotation marks. If the unit is weeks, the WEEKFIRST setting is used in the calculation.

format

Is the format of the result, enclosed in single quotation marks. The format must be double-precision format.

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

In the following, HDIFF finds the number of days between the ADD_MONTH and TRANSDATE fields.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
DIFF_DAYS/D12.2 =  HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ADD_MONTH                DIFF_DAYS
------  ---------         ---------                ---------
1118    2000/06/26 05:45  2000/08/26 05:45:00          61.00
1237    2000/02/05 03:30  2000/04/05 03:30:00          60.00
```

HDTTM: Converting a Date field to a Date-Time Field

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HDTTM function converts a date field to a date-time field. The time portion is set to midnight.

Syntax How to Convert a Date Field to a Date-Time Field

```
HDTTM (date, length, format)
```

where:

date

Is the date value to be converted. You can supply the name of a date field, a date constant, or an expression that returns a date value.

length

Is the length of the returned date-time value. Valid values are:

`8` for time values including milliseconds.

`10` for time values including microseconds.

format

Is the format of the returned date-time value. The format must be a date-time format (data type H).

Example Converting a Date Field to a Date-Time Field

In the following, HDTTM converts the date field TRANSDATE_DATE to a date-time field.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         TRANSDATE_DATE  DT2
------  ---------         --------------  ---
1118    2000/06/26 05:45  2000/06/26      2000/06/26 12:00AM
1237    2000/02/05 03:30  2000/02/05      2000/02/05 12:00AM
```

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000 NT/2000

Available Languages: reporting, Maintain

The HGETC function stores the current date and time in a date-time field. If millisecond or microsecond values are not available in your operating environment, the value returned for these components is zero.

Syntax How to Store the Current Date and Time in a Date-Time Field

```
HGETC (length, 'format')
```

where:

length

Is the length of the returned date-time value. Valid values are:

8 for time values including milliseconds.

10 for input time values including microseconds.

format

Is the format of the returned date-time value, enclosed in single quotation marks. The format must be a date-time format (data type H).

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

In the following, HGETC stores the current date and time in field DT2:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DT2/HYYMDm = HGETC(10, 'HYYMDm');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         DT2
------  ---------         ---
1118    2000/06/26 05:45  2000/10/03 15:34:24.000000
1237    2000/02/05 03:30  2000/10/03 15:34:24.000000
```

HHMMSS: Returning the Current Time

Available Operating Systems: All

Available Languages: reporting, Maintain

The HHMMSS function retrieves the current time from the operating system and returns the time as an eight-character string, separating the hours minutes and seconds with periods for reporting and colons for Maintain.

Note:

- &TOD returns the current time of day.
- Compiled MODIFY procedures must use the HHMMSS function to obtain the time; they cannnot use the &TOD variable. The &TOD variable is made current only when you execute a MODIFY, SCAN, or FSCAN procedure.

Syntax How to Retrieve the Current Time

```
HHMMSS(outfield)
```

where:

outfield

Alphanumeric

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

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

Example Displaying the Current Time

The following retrieves the current time and displays it in a report footing:

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES' AND COMPUTE
NOWTIME/A8 = HHMMSS(NOWTIME); NOPRINT
BY DEPARTMENT
FOOTING
"SALARY REPORT RUN AT TIME <NOWTIME"
END
```

The output is:

```
DEPARTMENT   TOTAL SALARIES
----------   --------------
MIS             $108,002.00
PRODUCTION      $114,282.00

SALARY REPORT RUN AT TIME 15.21.14
```

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HINPUT function converts an alphanumeric string to a date-time value.

Syntax How to Convert an Alphanumeric String to a Date-Time Value

```
HINPUT (inputlength, 'inputstring', length, 'Hfmt')
```

where:

inputlength

Is the length of the alphanumeric string to convert. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

inputstring

Is the alphanumeric string to convert. You can supply the actual string enclosed in single quotation marks, the name of an alphanumeric field, or an expression that returns an alphanumeric value. The alphanumeric string can consist of any valid date-time input value as described in *Describing Data*.

length

Is the length of the returned date-time value. Valid values are:

`8` for time values down to milliseconds.

`10` for time values down to microseconds.

H*fmt*

Is the format of the returned date-time value, enclosed in single quotation marks.

Example Converting an Alphanumeric String to a Date-Time Value

In the following request, HCNVRT converts the TRANSDATE field to alphanumeric format, and 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
------  ---------         ---------------       -------------
1118    2000/06/26 05:45  20000626054500000     2000/06/26 05:45:00
1237    2000/02/05 03:30  20000205033000000     2000/02/05 03:30:00
```

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HMIDNT function changes the time portion of a date-time field to midnight (all zeroes). This function can be used for testing date-time fields for a given date.

Syntax How to Set the Time Portion of a Date-Time Field to Midnight

```
HMIDNT (value, length, 'format')
```

where:

value

Is a date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

length

Is the length of the returned date-time value. Valid values are:

`8` for time values down to milliseconds.

`10` for time values down to microseconds.

format

Is the format of the returned date-time value, enclosed in single quotation marks. The format must be a date-time format (data type H).

Example Setting the Time to Midnight

In the following request, HMIDNT sets the time portion of the TRANSDATE field to midnight.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANSDATE_MID_24/HYYMDS  = HMIDNT(TRANSDATE, 8, 'HYYMDS');
TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         TRANSDATE_MID_24     TRANSDATE_MID_12
------  ---------         ----------------     ----------------
1118    2000/06/26 05:45  2000/06/26 00:00:00  2000/06/26 12:00:00AM
1237    2000/02/05 03:30  2000/02/05 00:00:00  2000/02/05 12:00:00AM
```

HNAME: Extracting a Date-Time Component in Alphanumeric Format

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HNAME function extracts a specified component from a date-time field and returns it in alphanumeric format.

Syntax How to Extract a Date-Time Component in Alphanumeric Format

```
HNAME (value, 'component', format)
```

where:

value

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be extracted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 5-2 for a list of supported components.

format

Is the alphanumeric format of the returned component, enclosed in single quotation marks. All other components are converted to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

Example Extracting the Week Component From a Field

In the following request, HNAME extracts the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST setting changes the value of the extracted component.

```
SET WEEKFIRST = 7
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
WHERE DATE EQ 2000
END
```

When WEEKFIRST is set to 7, the output is:

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

When WEEKFIRST is set to 3, the output is:

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

Example Extracting the Day Component From a Date-Time Field

In the following request, HNAME extracts the day in alphanumeric format from the TRANSDATE field.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/A2  = HNAME(TRANSDATE, 'DAY', 'A2');
WHERE DATE EQ 2000
END
```

The output is:

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

HPART: Returning a Date-Time Component in Numeric Format

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HPART function extracts a specified component from a date-time field and returns it in numeric format.

Syntax How to Return a Date-Time Component in Numeric Format

```
HPART (value, 'component', 'format')
```

where:

value

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be extracted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 5-2 for a list of supported components.

format

Is the integer format of the returned component, enclosed in single quotation marks. The year is always four digits, and the hour assumes the 24-hour system.

Example Extracting the Day Component in Numeric Format From a Date-Time Field

In the following request, HPART extracts the day in integer format from the TRANSDATE field.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
DAY_COMPONENT/I2  = HPART(TRANSDATE, 'DAY', 'I2');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         DAY_COMPONENT
------  ---------         -------------
1118    2000/06/26 05:45             26
1237    2000/02/05 03:30              5
```

HSETPT: Inserting a Component Into a Date-Time Field

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HSETPT function inserts the numeric value of a specified component into a date-time field.

Syntax How to Insert a Component Into a Date-Time Field

```
HSETPT (dtfield, 'component', value, length, 'format')
```

where:

dtfield

Is the date-time value. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

component

Is the name of the component to be inserted, enclosed in single quotation marks. See *Component Names and Values for Use With Date-Time Functions* on page 5-2 for a list of supported components.

value

Is the numeric value to use for the requested component. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

length

Is the length of the returned date-time value. Valid values are:

`8` for time values down to milliseconds.

`10` for time values down to microseconds.

format

Is the format of the returned date-time value, enclosed in single quotation marks. The format must be a date-time format (data type H).

Example Inserting the Day Component Into a Date-Time Field

In the following request, HSETPT inserts the day into the ADD_MONTH field.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ADD_MONTH/HYYMDS = HADD (TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
WHERE DATE EQ 2000
END
```

The output is:

```
CUSTID  DATE-TIME         ADD_MONTH            INSERT_DAY
------  ---------         ---------            ----------
1118    2000/06/26 05:45  2000/08/26 05:45:00  2000/08/28 05:45:00
1237    2000/02/05 03:30  2000/04/05 03:30:00  2000/04/28 03:30:00
```

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

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The HTIME function converts the time portion of a date-time field to a numeric number of milliseconds if the first argument is 8, or microseconds if the first argument is 10. To include microseconds, the input date-time field must be a 10-byte field.

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

```
HTIME (length, value, 'format')
```

where:

length

Is the length of the input date-time value. Valid values are:

8 for time values down to milliseconds.

10 for input time values down to microseconds.

value

Is the date-time value from which to extract the time. You can supply the name of a date-time field, a date-time constant, or an expression that returns a date-time value.

format

Is the format of the returned number of milliseconds or microseconds, enclosed in single quotation marks. The format must be a double-precision format.

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

In the following request, HTIME converts the time portion of the TRANSDATE field to the number of milliseconds.

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
WHERE DATE EQ 2000
END
```

The output is:

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

TODAY: Returning the Current Date

Available Operating Systems: All

Available Languages: reporting, Maintain

The TODAY function retrieves the current date from the system in the format MM/DD/YY or MM/DD/YYYY. The TODAY function always returns a date that is current. Therefore, if you are running an application late at night, you may want to use the TODAY function. You can remove the embedded slashes using the EDIT function.

You can also retrieve the date in the same format (separated by slashes) by using the system variable &DATE. You can retrieve the date without the slashes using the system variables &YMD, &MDY, and &DMY. The system variable &DATE*fmt* retrieves the date in a specified format.

Syntax How to Retrieve the Current Date

```
TODAY(outfield)
```

where:

outfield

Alphanumeric, at least A8

Is the name of the field to which the current date in MM/DD/YY[YY] format is returned, or the format of the output value enclosed in single quotation marks. The following determines the format:

- 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*.

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

Example Displaying the Current Date

In the following request, TODAY retrieves the current date and stores it in the DATE field. The DATE field is then used to display the date in a report heading.

```
DEFINE FILE EMPLOYEE
DATE/A10 WITH EMP_ID=TODAY(DATE);
END
TABLE FILE EMPLOYEE
SUM CURR_SAL BY DEPARTMENT
HEADING
"PAGE <TABPAGENO  "
"SALARY REPORT RUN ON <DATE  "
END
```

The output is:

```
PAGE  1
SALARY REPORT RUN ON 12/13/1999
DEPARTMENT     CURR_SAL
----------     --------
MIS         $108,002.00
PRODUCTION  $114,282.00
```

Using Legacy Date Functions

The functions listed in this topic are legacy functions. These functions were created for use with dates in integer, packed decimal, or alphanumeric format.

The following is the difference between a date format (formerly called a smart date) and a legacy date:

- A date format refers to an internally stored integer that represents the number of days between a real date value and a base date (either December 31, 1900, for dates with YMD or YYMD format; or January 1901, for dates with YM, YYM, YQ, or YYQ format). A Master File does not specify a data type or length for a date format; instead, it specifies display options such as D (day), M (month), Y (2-digit year), or YY (4-digit year). For example, MDYY in the USAGE (also known as FORMAT) attribute of a Master File is a date format. A real date value such as March 5, 1999, displays as 03/05/1999, and is internally stored as the offset from December 31, 1900.
- A legacy date refers to an integer, packed decimal, or alphanumeric format with date edit options, such as I6YMD, A6MDY, I8YYMD, or A8MDYY. For example, A6MDY is a 6-byte alphanumeric string; the suffix MDY indicates how Information Builders will return the data in the field. The sample value 030599 displays as 03/05/99.

Using Legacy Versions of Date Functions

All date functions have been rewritten to 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 the function, for example, if you do not use year 2000 dates. You can "turn off" the new versions with the DATEFNS parameter.

For details of how the DATEFNS parameter affects a specific function, see the description of the function.

Syntax How to Activate Legacy Date Functions

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

where:

ON

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

OFF

Deactivates the functions that support dates for the year 2000 and beyond.

Using Dates With Two and Four-Digit Years

Legacy date functions accept dates with two or four digit years. Four digit years that display the century, such as 2000 and 1900, can be used if their formats are specified as I8YYMD, P8YYMD, D8YYMD, F8YYMD, or A8YYMD. Two-digit years that do not specify the century can utilize 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 following example illustrates how to use the EDIT function to assign dates with four-digit years. It then converts these dates to Julian and Gregorian formats.

```
DEFINE FILE EMPLOYEE
DATE/I8YYMD = EDIT('19'|EDIT(HIRE_DATE));
JDATE/I7 = JULDAT(DATE, 'I7');
GDATE/I8 = GREGDT(JDATE, 'I8');
END
TABLE FILE EMPLOYEE
PRINT DATE JDATE GDATE
END
```

The output is:

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

Example Using Two-Digit Years

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

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

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

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

The output is:

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

AYM: Adding or Subtracting Months to or From Dates

Available Operating Systems: OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

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

Syntax How to Add or Subtract Months to or From Dates

AYM(*indate, months, outfield*)

where:

indate

Numeric

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

months

Integer

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

outfield

Integer

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

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

Tip:

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

Example Adding Months to a Date

The following request adds six months to the hire date of employees. AYM takes a starting date that you supply (in this case, HIRE_MONTH, in YM format), and uses a monthly interval that you supply (in this case, 6), to determine a resulting date (AFTER6MONTHS, also in YM format).

Note that the COMPUTE command converts the dates from year-month-day to year-month formats by dividing the dates by 100.

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

The 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 Dates

Available Operating Systems:OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The AYMD function takes a valid date in year-month-day format and adds or subtracts a given number of days from the date. You can convert a date to this format using the CHGDAT function or EDIT function.

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

Syntax

How to Add or Subtract Days to or From Dates

```
AYMD(indate, days, outfield)
```

where:

indate

Integer

Is the input date in year-month-day format or the field that contains the input date. If *indate* is a field name, it must refer to a field with I6, I6YMD, I8, I8YYMD, P6, P6YMD, F6, F6YMD, D6, or D6YMD format. If the date is not valid, the function returns a 0.

days

Integer

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

outfield

I6, I6YMD, I8, or I8YYMD

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

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

Example Adding Days to a Date

The following request adds 35 days to the hire date of employees. AYMD takes the date in HIRE_DATE and uses the interval 30 to calculate a new date, and saves 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 Date Formats

Available Operating Systems: OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

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

The format of the date to be converted and the resulting date contain the following characters in any combination:

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

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

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

Any other character in the format is ignored.

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

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

Syntax How to Change Date Formats

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

where:

'oldformat'

A5

Is the format of the input date.

'newformat'

A5

Is the format of the converted date.

indate

Alphanumeric

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

outfield

Alphanumeric or A17

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

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

Tip:

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

Example Converting a Numeric Date to Its Full Name

In this example, CHGDAT takes a date that you supply (in this case, DATE) in YMD format and converts it to a resulting date (NEWDATE) in MDYYX format.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A6 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

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

DA Functions: Converting a Date to an Integer

Available Operating Systems: All

Available Languages: reporting, Maintain

The DA functions convert dates to the number of days between the date and December 31, 1899. By converting dates to the number of days, you can add and subtract dates and calculate the intervals between them. You can convert the results back to date format by using the DT functions discussed in *DT Functions: Converting an Integer to a Date* on page 5-67.

There are six DA functions; each one accepts dates 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 dates in day-month-year format.

DADYM converts dates in day-year-month format.

DAMDY converts dates in month-day-year format.

DAMYD converts dates in month-year-day format.

DAYDM converts dates in year-day-month format.

DAYMD converts dates in year-month-day format.

indate

Numeric

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

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

Alphanumeric

Is the name of the field to which the number of days this century is returned, or the format of the output value enclosed in single quotation marks.

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

Example

Calculating the Difference Between Two Dates

The following example shows the number of days elapsed between the time employees get raises and the time they were hired. DAYMD takes two dates that you supply (in this case, DAT_INC and HIRE_DATE) in YMD format, converts both to the number of days since December 31, 1899, and subtracts the smaller from the larger:

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

The output is:

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

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

Available Operating Systems: All

Available Languages: reporting, Maintain

The DMY, MDY, and YMD functions calculate the difference between two dates in integer, alphanumeric, or packed format.

Syntax How to Calculate the Difference Between Two Dates

function(*begin*, *end*)

where:

function

Is one of the following:

DMY calculates the difference between two dates in day-month-year format.

MDY calculates the difference between two dates in month-day-year format.

YMD calculates the difference between two dates in year-month-day format.

begin

Numeric

Is the beginning date. You may supply the actual date or the name of a field that contains the date.

end

Numeric

Is the end date. You may supply the actual date or the name of a field that contains the date.

Example Calculating the Number of Days Between Two Dates

The following request calculates the number of days between an employee's start date and first pay raise. YMD takes the dates in HIRE_DATE and DAT_INC, and calculates the number of days between them.

```
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

Available Operating Systems: All

Available Languages: reporting, Maintain

The DOWK and DOWKL functions find the day of the week that corresponds to a date. The DOWK function returns the day as a 3-letter abbreviation; the DOWKL function displays the full name of the day.

Syntax How to Find the Day of the Week

```
DOWK(indate, outfield)
```

or

```
DOWKL(indate, outfield)
```

where:

indate

Numeric

Is the input date in year-month-day format. If the date is not valid, the function returns spaces. If the date specifies a 2-digit year and DEFCENT and YRTHRESH values have not been set, the function assumes the 20th century.

outfield

DOWK: A4

DOWKL: A12

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

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

Example Finding the Day of the Week

In this example, DOWK uses the argument in HIRE_DATE to determine the day of the week employees were hired, and stores the result in DATED.

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND HIRE_DATE AND COMPUTE
DATED/A4 = DOWK(HIRE_DATE, DATED);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
EMP_ID     HIRE_DATE  DATED
------     ---------  -----
071382660   80/06/02  MON
119265415   82/01/04  MON
119329144   82/08/01  SUN
123764317   82/01/04  MON
126724188   82/07/01  THU
451123478   82/02/02  TUE
```

DT Functions: Converting an Integer to a Date

Available Operating Systems: All

Available Languages: reporting, Maintain

The DT functions convert an integer representing the days elapsed since December 31, 1899 to the corresponding date. The DT functions are useful when you are performing arithmetic on a date converted to the number of days (see *DA Functions: Converting a Date to an Integer* on page 5-40). The DT functions convert the result back to date format.

There are six DT functions. Each one converts a number into a date of a different format.

Syntax How to Convert Integers to Dates

function(number, outfield)

where:

function

Is one of the following:

DTDMY converts numbers to day-month-year dates.

DTDYM converts numbers to day-year-month dates.

DTMDY converts numbers to month-day-year dates.

DTMYD converts numbers to month-year-day dates.

DTYDM converts numbers to year-day-month dates.

DTYMD converts numbers to year-month-day dates.

number

Numeric

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

outfield

Integer

Is the name of the field to which the corresponding date is returned, or the format of the output value enclosed in single quotation marks. The date format is determined by the function being used.

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

Example Converting an Integer to a Date

The following takes a date that has been converted to the number of days (34650) and converts it back to the corresponding date, in month-day-year format. DAYMD takes the argument HIRE_DATE, determines how many days have passed since December 31, 1899, and stores the result in NEWF. Then DTMDY takes NEWF, converts the result back to a date, this time with a four-digit year, and stores the result in NEW_HIRE_DATE.

```
-* THIS PROCEDURE CONVERTS HIRE_DATE, WHICH IS IN I6YMD FORMAT,
-* TO A DATE IN I8MDYY FORMAT.
-* FIRST IT USES THE DAYMD FUNCTION TO CONVERT HIRE_DATE
-* TO A NUMBER OF DAYS.
-* THEN IT USES THE DTMDY FUNCTION TO CONVERT THIS NUMBER OF
-* DAYS TO I8MDYY FORMAT
-*
DEFINE FILE EMPLOYEE
NEWF/I8 WITH EMP_ID=DAYMD(HIRE_DATE,NEWF);
NEW_HIRE_DATE/I8MDYY WITH EMP_ID=DTMDY(NEWF,NEW_HIRE_DATE);
END
TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE
BY FN BY LN
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

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

GREGDT: Converting From Julian to Gregorian Format

Available Operating Systems: All

Available Languages: reporting, Maintain

The GREGDT function converts dates in Julian format to year-month-day format. Dates in Julian format are five- or seven-digit numbers. The first two or four digits are the year; the last three digits are the number of the day counting from January 1. For example, January 1, 1999 in Julian format is either 99001 or 1999001.

Depending on the format of the output, GREGDT converts a Julian date to either YMD or YYMD format, using the DEFCENT and YRTHRESH settings.

GREGDT returns dates in the following format:

DATEFNS setting	I6 or I7 format	I8 format or greater
ON	YMD	YYMD
OFF	YMD	YMD

Syntax How to Convert Julian Format Dates to Gregorian Format

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

where:

indate

Numeric

Is the Julian date, which is truncated to an integer before conversion. Each value must be a 5- or 7-digit number after truncation. The first two or four digits represent the year, the last three digits must be between 001 and 365 (366 for a leap year). If the date is invalid, the function returns a 0.

outfield

I6 or larger

Is the name of the field to which the date in year-month-day format is returned, or the format of the output value enclosed in single quotation marks.

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

Example Converting a Date to a Julian and a Gregorian Date

In this example, GREGDT takes the argument JULIAN and converts it to YYMD (Gregorian) format.

```
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
```

Notice that GREGDT determines the century (using the DEFCENT and YRTHRESH settings).

JULDAT: Converting a Date From Gregorian to Julian Format

Available Operating Systems: All

Available Languages: reporting, Maintain

The JULDAT function converts a date from year-month-day format to Julian (year-day) format. 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.

Depending on the format of the output, JULDAT uses the DEFCENT and YRTHRESH parameter settings to convert a date to either YYNNN or YYYYNNN format.

JULDAT returns dates in the following format:

DATEFNS setting	I5 or I6 format	I7 format or greater
ON	YYNNN	YYYYNNN (JULDAT uses the DEFCENT and YRTHRESH settings to determine the century, if necessary).
OFF	YYNNN	YYNNN

Syntax How to Convert a Gregorian Date to a Julian Date

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

where:

indate

Numeric

Is the date or field containing the date in year-month-day format (YMD or YYMD).

outfield

Integer at least I5

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

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

Example Converting a Gregorian Date to a Julian Date

In this example, JULDAT takes the argument HIRE_DATE and converts it to Julian format.

```
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
```

Notice that JULDAT determines the century (using the DEFCENT and YRTHRESH settings).

YM: Calculating Elapsed Months

Available Operating Systems: All

Available Languages: reporting, Maintain

The YM function calculates the number of months that elapse between two dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT function or the EDIT function.

Syntax

How to Calculate Elapsed Months

YM(*fromdate, todate, outfield*)

where:

fromdate

Numeric

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

todate

Numeric

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

outfield

Integer

Is the name of the field to which the number of months between the two dates is returned, or the format of the output value enclosed in single quotation marks.

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

Tip:

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

Example Calculating the Difference in Months Between Two Dates

In the following example, YM takes the arguments HIRE_DATE/100 and DAT_INC/100, calculates the difference in months between the two dates, and stores the results in MONTHS_HIRED.

Note that the COMPUTE expression converts the dates from year-month-day to year-month format by dividing the dates by 100.

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100; NOPRINT AND COMPUTE
MONTH_INC/I4YM = DAT_INC/100; NOPRINT AND COMPUTE
MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
BY LAST_NAME BY FIRST_NAME BY HIRE_DATE
IF MONTHS_HIRED NE 0
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME       FIRST_NAME  HIRE_DATE  RAISE DATE  MONTHS_HIRED
---------       ----------  ---------  ----------  ------------
CROSS           BARBARA      81/11/02    82/04/09             5
GREENSPAN       MARY         82/04/01    82/06/11             2
JONES           DIANE        82/05/01    82/06/01             1
MCCOY           JOHN         81/07/01    82/01/01             6
SMITH           MARY         81/07/01    82/01/01             6
```

CHAPTER 6

Format Conversion Functions

Topics:

- Alphabetical List of Format Conversion Functions

Format conversion functions convert fields from one format to another.

ATODBL: Converting an Alphanumeric String to Double-Precision Format

Available Operating Systems: OS/390, VM/CMS

Available Languages: reporting, Maintain

The ATODBL function converts a number from alphanumeric to double-precision format.

The ATODBL function is useful when executing a -RUN command in Dialogue Manager. All numeric arguments in Dialogue Manager are stored in alphanumeric format. However, these arguments must be converted to double-precision format for use with a function. The -SET command automatically converts these arguments, but the -RUN command does not. Therefore, you must convert each numeric argument into double-precision format and store it in a Dialogue Manager variable, which is used as a function argument.

For other applications, the EDIT function performs this conversion. Since the EDIT function cannot store double-precision numbers in Dialogue Manager variables, you must call the ATODBL function to convert the arguments.

Procedure How to Convert an Alphanumeric String to Double-Precision Format With the -RUN Command

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

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

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

   where:

   &outfield

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

2. Call the ATODBL function from an operating system -RUN command. The syntax is

   ```
   -{CMS|TSO|MVS} RUN ATODBL, number, inlength, &outfield
   ```

   where:

   CMS|TSO|MVS

   Is the operating system.

   number

   Alphanumeric

   Is the number you want to convert. This can be a numeric constant, or a variable that contains the number. The number can be up to 15 bytes long and can contain a sign and a decimal point but no other character; if it does, the function returns a 0.

inlength
Alphanumeric
Is the number of bytes in *number*. The maximum value is 15.
Note: This must be a character string.

outfield
A8
Is the variable defined in step 1.

Syntax How to Convert an Alphanumeric String to Double-Precision Format From a FOCUS Command

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

where:

number
Alphanumeric
Is the number you want to convert. This can be a numeric constant, or a field or variable that contains the number. The number can be up to 15 bytes long and can contain a sign and a decimal point but no other characters; if it does, the function returns a 0.

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

outfield
Double-Precision
Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks.
Note: In Dialogue Manager, the format must be specified. In Maintain, the name of the field must be specified.

Example Converting an Alphanumeric Value to a Double-Precision Number 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,$
    .
    .
    .
```

In the following procedure, ATODBL converts the value entered 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 follows:

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

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

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

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

The procedure processes as follows:

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

Example Converting an Alphanumeric Field to Double-Precision Format

In this example, 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
```

EDIT: Converting the Format of a Field

Available Operating Systems: OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting

The EDIT function converts an alphanumeric field that contains numeric characters to numeric format, or converts a numeric field to alphanumeric format. This is useful when you need to manipulate a field using a command that requires a particular format.

When you use EDIT to assign the converted value to a field, the format of the new field must correspond to the format of the returned value. For example, if you use EDIT to convert a numeric field to alphanumeric format, and then assign the resulting value to an alphanumeric field, you must give the new field an alphanumeric format as follows:

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE);
```

When the EDIT function encounters a symbol, it deals with it in the following way:

- When converting an alphanumeric field to numeric format, a sign or decimal point in the field is acceptable and remains in the value stored in the numeric field.
- When converting a floating-point or packed-decimal field to alphanumeric format, EDIT removes the sign, the decimal point, and any number to the right of the decimal point. It then right-justifies the remaining digits and adds leading zeros to the specified field length. Also, converting a number with more than nine significant digits in floating-point or packed-decimal format may produce an incorrect result.

The EDIT function can also extract characters from or add characters to an alphanumeric string. For more information, see Chapter 3, *Character Functions*.

Syntax How to Convert Field Formats

```
EDIT(fieldname);
```

where:

fieldname

Alphanumeric or Numeric

Is the field name enclosed in parentheses.

Example Converting From Numeric Format to Alphanumeric Format

In the following example, EDIT converts HIRE_DATE to alphanumeric format. CHGDAT is then able to use the field, which it expects to be in alphanumeric format.

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

The output is:

```
LAST_NAME        FIRST_NAME  HIRE_DATE  HIRE_MDY
---------        ----------  ---------  --------
BLACKWOOD        ROSEMARIE    82/04/01  APRIL 01 1982
CROSS            BARBARA      81/11/02  NOVEMBER 02 1981
GREENSPAN        MARY         82/04/01  APRIL 01 1982
JONES            DIANE        82/05/01  MAY 01 1982
MCCOY            JOHN         81/07/01  JULY 01 1981
SMITH            MARY         81/07/01  JULY 01 1981
```

FTOA: Converting a Number to Alphanumeric Format

Available Operating Systems: All

Available Languages: reporting, Maintain

The FTOA function converts a number up to 16 digits long from numeric format to alphanumeric format. It retains the decimal positions of a number and right-justifies it with leading spaces. You can also add edit options to a number converted by FTOA.

Syntax How to Convert a Number to Characters

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

where:

number

Numeric

Is the number to be converted. This can be the number, or the field containing the number.

'(format) '

Alphanumeric

Is the format of the number as it is stored in numeric format, enclosed in both single quotation marks and parentheses. Only single-precision floating point and double-precision formats are supported. Include any edit options that you want to appear in the output.

If you are using a field for this argument, specify the field name without quotation marks or parentheses. The values in the field must be enclosed in parentheses.

outfield

Alphanumeric

Is the field to which the result is returned, 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. The D format automatically supplies commas.

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

Example Converting From Numeric to Alphanumeric Format

In this example, FTOA converts the GROSS field from double-precision to alphanumeric format, and stores the result in ALPHA_GROSS.

```
TABLE FILE EMPLOYEE
PRINT GROSS AND COMPUTE
ALPHA_GROSS/A14 = 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 Number to a Character

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

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

The display of special characters depends upon your software and hardware; not all special characters may display. Printable EBCDIC and ASCII characters and their integer equivalents are listed in character charts. If you are using the Hot Screen facility, some unusual characters cannot be displayed. If Hot Screen does not support the character you chose, enter the commands

```
SET SCREEN = OFF
RETYPE
```

and redisplay the output which will appear as regular terminal output.

Syntax How to Convert a Number to a Character

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

where:

input

Numeric

Is the decimal value to be translated to a single character. A value greater than 255 is treated as the remainder of *input* divided by 256.

output

Alphanumeric

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

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

Example Determining the Decimal Value of a Character

In this example, HEXBYT converts LAST_INIT_CODE into a letter 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 in S/390

In the following example, HEXBYT converts the value 192 into its character equivalent which is a left brace, and the value 208 to its character equivalent which is a right brace. If the value of CURR_SAL is less than 12000, the value in LAST_NAME is enclosed in braces.

```
DEFINE FILE EMPLOYEE
BRACE/A17 = HEXBYT(192, 'A1') | LAST_NAME | HEXBYT(208, 'A1');
BNAME/A17 = IF CURR_SAL LT 12000 THEN BRACE
    ELSE LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT BNAME CURR_SAL BY EMP_ID
END
```

The output is:

```
EMP_ID     BNAME                    CURR_SAL
------     -----                    --------
071382660  {STEVENS        }       $11,000.00
112847612  SMITH                   $13,200.00
117593129  JONES                   $18,480.00
119265415  {SMITH          }        $9,500.00
119329144  BANNING                 $29,700.00
123764317  IRVING                  $26,862.00
126724188  ROMANS                  $21,120.00
219984371  MCCOY                   $18,480.00
326179357  BLACKWOOD               $21,780.00
451123478  MCKNIGHT                $16,100.00
543729165  {GREENSPAN      }        $9,000.00
818692173  CROSS                   $27,062.00
```

ITONUM: Converting a Large Binary Integer to Double-Precision Format

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The ITONUM function converts large binary integers 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 syntax so require conversion to double-precision format. The user specifies how many of the rightmost bytes in the input string are significant, and the result is an 8-byte double-precision field.

Syntax How to Convert Large Binary Integers to Double-Precision Format

```
ITONUM(maxbytes, infield, outfield)
```

where:

maxbytes

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign. Valid values are:

`5` ignores the left-most 3 bytes.

`6` ignores the left-most 2 bytes.

`7` ignores the left-most byte.

infield

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats of the field must be A8.

outfield

Numeric

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

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

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 called 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/D12.2 = ITONUM(6, BINARYFLD, MYFLD);
END
TABLE FILE EUROCAR
PRINT MYFLD BY CAR
END
```

ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The ITOPACK function converts large binary integers 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 syntax so require conversion to packed format. The user specifies how many of the rightmost bytes in the input string are significant, and the output is an 8-byte packed field of up to 15 significant numeric positions (for example, P15 or P16.d).

Limit:

For a field defined as ‘PIC 9(15) COMP’ or the equivalent (15 significant digits), the maximum number that can be translated is 167,744,242,712,576.

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 points).

`6` ignores the left-most 2 bytes (up to 14 significant points).

`7` ignores the left-most byte (up to 15 significant points).

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 field to which the result is returned, or the format of the output value enclosed in single quotation marks. The format must be specified as Pn or Pn.d.

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

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 to FOCUS in the EUROCAR Master File as a field called BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The following request converts the field to packed decimal :

```
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

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The ITOZ function converts numbers in numeric format to zoned format. Although a request cannot process zoned numbers, it can write zoned fields to extract files for use by external programs.

Syntax

How to Convert to Zoned Format

`ITOZ(outlength, number, outfield)`

where:

outlength

Numeric

Is the length of *number* in bytes. The maximum number of bytes is 15. The last byte includes the sign.

number

Numeric

Is the number to be converted, or the field that contains the number. The number is truncated to an integer before it is converted.

outfield

Alphanumeric

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

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

Example Converting a Number to Zoned Format

The following request prepares an extract file containing employee IDs and salaries in zoned format for a COBOL program. The request is:

```
DEFINE FILE EMPLOYEE
ZONE_SAL/A8 = ITOZ(8, CURR_SAL, ZONE_SAL);
END

TABLE FILE EMPLOYEE
PRINT 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  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, the output is:

```
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
```

Note: The left brace in EBCIDIC is C0; this indicates a positive sign and a final digit of 0. The capital B in EBCIDIC is C2; this indicates a positive sign and a final digit of 2.

PCKOUT: Writing Packed Numbers of Different Lengths

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The PCKOUT function enables a request to write packed numbers of different lengths to an extract file. When a request saves a packed field in an extract file, it writes it as an 8- or 16-byte field regardless of its format specifications. With the PCKOUT function, you can vary the field's length between 1 to 16 bytes.

Syntax

How to Write Packed Numbers of Different Lengths

PCKOUT(*infield, outlength, outfield*)

where:

infield

Numeric

Is the input field that contains the values. The field can be 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 field to which the result is returned, or the format of the output value enclosed in single quotation marks. The function returns the field as alphanumeric although it contains packed data.

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

Example Writing Packed Numbers of Different Lengths

In the following example, PCKOUT converts the CURR_SAL field to a 5 byte packed field and stores the results 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 output is:

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


 EBCDIC  RECORD NAMED  SAVE
 FIELDNAME                         ALIAS         FORMAT        LENGTH

 LAST_NAME                         LN            A15            15
 SHORT_SAL                                       A5              5
 HIRE_DATE                         HDT           I6YMD           6

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

UFMT: Converting Alphanumeric to Hexadecimal

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

Available Languages: reporting, Maintain

The UFMT function converts characters in an alphanumeric field to their hexadecimal (HEX) representation. This function is useful for examining data of unknown format. As long as the length of the data is known, its content can be examined.

Syntax How to Convert Alphanumeric to Hexadecimal

UFMT(*string, inlength, outfield*)

where:

string

Alphanumeric

Is the value to be converted. This can be an alphanumeric string enclosed in single quotation marks, or the field that contains the string.

inlength

Numeric

Is the length of *string*.

outfield

Alphanumeric

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. The format of *outfield* must be alphanumeric and have a length that is twice as long as *inlength*.

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

Example Converting an Alphanumeric Field to Hexadecimal

In the following example, UFMT converts each value in JOBCODE to its hexadecimal representation, and stores the result in the HEXCODE field.

```
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
```

CHAPTER 7

Numeric Functions

Topics:

- Alphabetical List of Numeric Functions

Numeric functions perform calculations on numeric constants and fields.

ABS: Calculating Absolute Value

Available Operating Systems: All

Available Languages: reporting, Maintain

The ABS function returns the absolute value of its argument.

Syntax How to Calculate Absolute Value

```
ABS(argument)
```

where:

argument

Numeric

Is the value for which the absolute value is returned. This can be the value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation.

Example Calculating Absolute Value

In the following example, the COMPUTE command creates the DIFF field. The ABS function then calculates the absolute value of DIFF.

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

The output is:

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

ASIS: Distinguishing Between a Blank and a Zero

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, Windows NT/2000

Available Languages: reporting

The ASIS function distinguishes between a blank and a zero in Dialogue Manager. It differentiates between numeric string constants or variables defined as numeric strings, and fields defined simply as numeric.

For details on the ASIS function, see Chapter 3, *Character Functions*.

BAR: Producing Bar Charts

Available Operating Systems: AS/400, OpenVMS, OS/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The BAR function enables you to produce horizontal bar charts.

A bar chart plots bars consisting of repeating characters for a printed field. Optionally, you can create a scale to clarify the meaning of a bar chart. This is done by replacing the title of the column where the bar is stored with a scale.

Syntax

How to Produce Bar Charts

```
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 field you wish to illustrate as a bar chart.

maxvalue

Numeric

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

'char'

Alphanumeric

Is the repeating character that creates the bars enclosed in single quotation marks. If more than one character is specified, only the first character is used to create the bars.

outfield

Alphanumeric

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. The output field must be large enough to contain a bar at maximum length as defined by *barlenth*.

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

Example Creating a Bar Chart

In the following example, 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

In the following example, BAR creates a bar chart for the CURR_SAL field. It then replaces the field name SAL_BAR with a scale using the AS phrase.

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

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

The 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 Packed Fields

Available Operating Systems: All

Available Languages: reporting, Maintain

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

To use the CHKPCK function, use these steps:

1. Ensure that the Master File (FORMAT, USAGE, and ACTUAL attributes), or the MODIFY FIXFORM command describing the file 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 causing data exceptions.
2. Call the CHKPCK function to examine the field. The function returns its output to a field defined as packed. If the value it examines is a valid packed number, the function returns the value; if it is not packed, it returns an error code.

Syntax How to Validate Packed Fields

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

where:

inlength

Numeric

Is the length of the field to be validated. This can be between 1 and 16 bytes.

infield

Alphanumeric

Is the field to be validated. 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, the error code may appear on a report with a decimal point because of the format of the output field.

outfield

Packed

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

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

Example Validating Packed Data

First

Prepare a data source that includes invalid packed data. The following creates the TESTPACK file, which contains the PACK_SAL field that is defined as an alphanumeric field but contains packed data. The invalid data contained in TESTPACK is returned 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 result 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)]
>
```

Second

Create a Master File for the TESTPACK data source. Define the PACK_SAL field as alphanumeric in the USAGE and ACTUAL attributes. The following is the Master File:

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

Last

Create a report request that uses the CHKPCK function to validate the values in the PACK_SAL field. The following validates the values in the PACK_SAL field, and stores the output in the GOOD_PACK field. Values that are not in packed format will return the error code -999.

```
DEFINE FILE TESTPACK
GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
END
TABLE FILE TESTPACK
PRINT DEPARTMENT GOOD_PACK BY EMP_ID
END
```

The output is:

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

DMOD, FMOD, and IMOD: Calculating the Remainder From a Division

Available Operating Systems: All

Available Languages: reporting, Maintain

The MOD functions calculate the remainder from a division. There are three MOD functions which differ in the format in which they return the remainder:

- DMOD returns the remainder as a decimal number.
- FMOD returns the remainder as a floating-point number.
- IMOD returns the remainder as an integer.

The three functions use the following formula:

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

Syntax

How to Calculate the Remainder From a Division

function(*dividend*, *divisor*, *outfield*)

where:

function

Is one of the following:

IMOD Returns the remainder as an integer.

FMOD Returns the remainder as a floating-point number.

DMOD Returns the remainder as a decimal number.

dividend

Numeric

Is the number being divided.

divisor

Numeric

Is the number dividing the dividend.

outfield

Numeric

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

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

Example

Calculating the Remainder of a Division

In the following example, IMOD divides ACCTNUMBER by 1000, and returns the remainder to LAST3_ACCT.

```
TABLE FILE EMPLOYEE
PRINT ACCTNUMBER AND
COMPUTE LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
BY LAST_NAME BY FIRST_NAME
WHERE ( ACCTNUMBER NE 000000000) AND (DEPARTMENT EQ 'MIS');
END
```

The output is:

```
LAST_NAME       FIRST_NAME  ACCTNUMBER  LAST3_ACCT
---------       ----------  ----------  ----------
BLACKWOOD       ROSEMARIE    122850108         108
CROSS           BARBARA      163800144         144
GREENSPAN       MARY         150150302         302
JONES           DIANE        040950036         036
MCCOY           JOHN         109200096         096
SMITH           MARY         027300024         024
```

EXP: Raising "e" to the Nth Power

Available Operating Systems: All

Available Languages: reporting, Maintain

The EXP function raises the value "e" (approximately 2.72) to any power you specify. This function is the inverse of the LOG function, which returns an argument's logarithm.

The EXP function calculates the answer by adding terms of an infinite series. If a term adds less than .000001 percent to the sum, the function ends the calculation and returns the result as a double-precision number.

Syntax How to Raise "e" to the Nth Power

```
EXP(power, outfield)
```

where:

power

Numeric

Is the power that "e" is being raised to.

outfield

Double-precision

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

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

Example Raising "e" to the Nth Power

In the following example, EXP raises "e" to the power designated by the &POW variable, specified as 3 here. The result is then rounded to the nearest integer with the .5 rounding constant.

```
-SET &POW = '3';
-SET &RESULT = EXP(&POW, 'D15.3') + 0.5;
-TYPE E TO THE &POW POWER IS APPROXIMATELY &RESULT
```

The output is:

```
E TO THE 3 POWER IS APPROXIMATELY 20
```

EXPN: Evaluating a Number in Scientific Notation

Available Operating Systems: AS/400, OpenVMS, OS/390, Windows NT/2000

Available Languages: reporting

The EXPN function evaluates an argument expressed in scientific notation.

Syntax

How to Evaluate a Number in Scientific Notation

```
EXPN(n.nn {E|D} {+|-} p)
```

where:

n.nn

Is a numeric constant that consists of a whole number component, followed by a decimal point, followed by a fractional component.

{E|D}

Denotes scientific notation. E and D are interchangeable.

p

Is the power of 10 to which you want to raise the number. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. The expression can also invoke a function.

For example, you can use scientific notation to express 103 as:

```
1.03E+2
```

Then

```
EXPN(1.03+2)
```

returns 103 as the result.

INT: Finding the Greatest Integer

Available Operating Systems: All

Available Languages: reporting, Maintain

The INT function returns the integer part of an argument.

Syntax How to Calculate the Greatest Integer

```
INT(argument)
```

where:

argument

Numeric

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

Example Calculating the Greatest Integer in a Field

In the following example, INT returns the greatest integer in the DED_AMT field.

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND COMPUTE
INT_DED_AMT/I9=INT(DED_AMT);
BY LAST_NAME BY FIRST_NAME
WHERE (DEPARTMENT EQ 'MIS') AND (PAY_DATE EQ 820730);
END
```

The output is:

```
LAST_NAME       FIRST_NAME          DED_AMT  INT_DED_AMT
---------       ----------          -------  -----------
BLACKWOOD       ROSEMARIE         $1,261.40         1261
CROSS           BARBARA           $1,668.69         1668
GREENSPAN       MARY                $127.50          127
JONES           DIANE               $725.34          725
SMITH           MARY                $334.10          334
```

LOG: Calculating the Natural Logarithm

Available Operating Systems: AS/400, HP, OpenVMS, OS/390, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The LOG function returns the natural logarithm of an argument.

Syntax How to Calculate the Natural Logarithm

```
LOG(argument)
```

where:

argument

Numeric

Is the value on which the function operates. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, make sure you use parentheses as needed to ensure the correct order of evaluation. If you enter an argument less than or equal to 0, LOG returns 0.

Example Calculating the Natural Logarithm

In the following example, LOG calculates the logarithm of the CURR_SAL field.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
LOG_CURR_SAL/D12.2 = LOG(CURR_SAL);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME       FIRST_NAME         CURR_SAL    LOG_CURR_SAL
---------       ----------         --------    ------------
BANNING         JOHN            $29,700.00           10.30
IRVING          JOAN            $26,862.00           10.20
MCKNIGHT        ROGER           $16,100.00            9.69
ROMANS          ANTHONY         $21,120.00            9.96
SMITH           RICHARD          $9,500.00            9.16
STEVENS         ALFRED          $11,000.00            9.31
```

MAX and MIN: Finding the Maximum or Minimum Value

Available Operating Systems: All

Available Languages: reporting, Maintain

The MAX and MIN functions return the maximum or minimum value, respectively, from a list of arguments.

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

Example Determining the Minimum Value

In the following example, MIN returns either the value from the ED_HRS field or the value 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
```

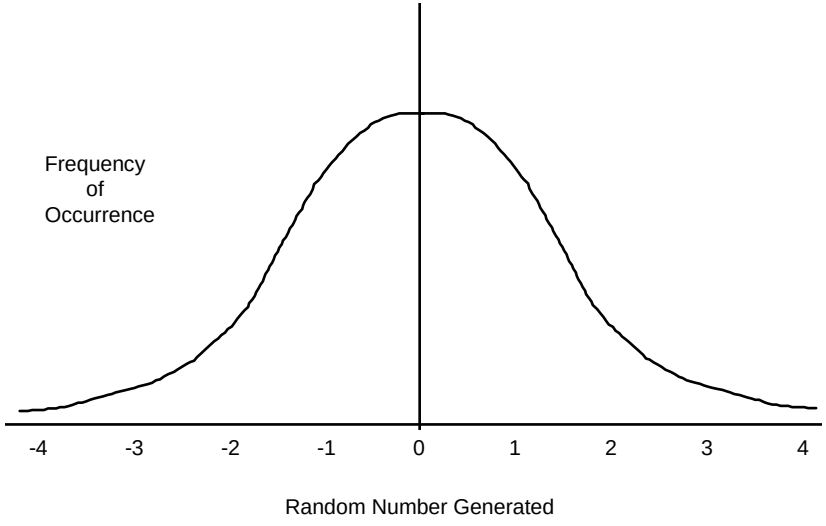
PRDNOR and PRDUNI: Generating Reproducible Random Numbers

Available Operating Systems: All

Available Languages: reporting, Maintain

The PRDNOR and PRDUNI functions generate reproducible random numbers:

- PRDNOR generates reproducible double-precision random numbers that are normally distributed with an arithmetic mean of 0 and a standard deviation of 1. If you use the PRDNOR function to generate a large set of numbers, it has the following properties:
  - The numbers in the set lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, which means that there are more numbers close to 0 than farther away.



  - The average of the set is close to 0.
  - The set can contain numbers of any size, but most of the numbers are between 3 and -3.
- 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).

Syntax How to Use PRDNOR and PRDUNI to Generate Random Numbers

```
{PRDNOR|PRDUNI}(seed, outfield)
```

where:

PRDNOR

Generates reproducible normally distributed random numbers with an arithmetic mean of 0 and a standard deviation of 1.

PRDUNI

Generates reproducible random numbers uniformly distributed between 0 and 1.

seed

Numeric

Is the seed or the field that contains the seed, up to nine bytes. The seed is truncated to an integer. Using the same seed will always produce the same set of numbers.

Note: In the PRDUNI function, VM/CMS behavior differs from OS/390 behavior. In VM/CMS, the seed number changes upon multiple executions as the function is reloaded. In OS/390, the function is loaded once. To keep the function loaded for the duration of the session, we recommend assigning the function to a temporary field using a DEFINE command. The function remains loaded in memory until the DEFINE is cleared.

outfield

Double-precision

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

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

Example Generating Reproducible Random Numbers

In this example, PRDNOR assigns random numbers and stores them in RAND. These values are then used to randomly pick five employee records identified by the values in the LAST NAME and FIRST NAME fields. The seed is 40. To produce a different set of numbers, change the seed.

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = PRDNOR(40, RAND);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

The output is:

```
RAND  LAST_NAME        FIRST_NAME
----  ---------        ----------
1.38  STEVENS          ALFRED
1.12  MCCOY            JOHN
 .55  SMITH            RICHARD
 .21  JONES            DIANE
 .01  IRVING           JOAN
```

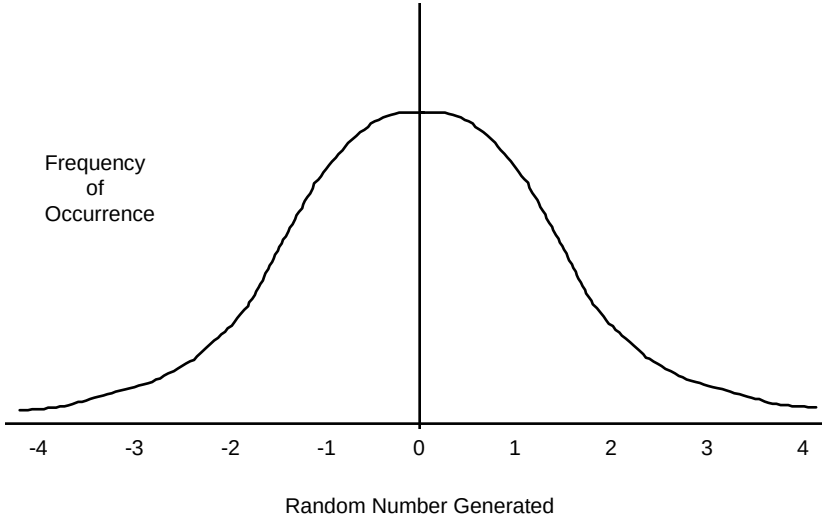
RDNORM and RDUNIF: Generating Random Numbers

Available Operating Systems: All

Available Languages: reporting, Maintain

The RDNORM and RDUNIF functions generate random numbers:

- RDNORM generates double-precision random numbers that are normally distributed with an arithmetic mean of 0 and a standard deviation of 1. If you use the RDNORM function to generate a large set of numbers (between 1 and 32768), it has the following properties:
  - The numbers in the set lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, which means that there are more numbers close to 0 than farther away.



  - The average of the set is close to 0.
  - The set can contain numbers of any size, but most of the numbers are between 3 and -3.
- 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 Use RDNORM and RDUNIF to Generate Random Numbers

```
{RDNORM|RDUNIF}(outfield)
```

where:

`RDNORM`

Generates normally distributed random numbers with an arithmetic mean of 0 and a standard deviation of 1.

`RDUNIF`

Generates random numbers uniformly distributed between 0 and 1.

`outfield`

Double-precision

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

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

Example

Generating Random Numbers

In this example, RDNORM assigns random numbers and stores them in RAND. These values are then used to randomly choose five employee records identified by the values in the LAST NAME and FIRST NAME fields.

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = RDNORM(RAND);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

The request produces output similar to the following:

```
RAND  LAST_NAME        FIRST_NAME
----  ---------        ----------
 .65  CROSS            BARBARA
 .20  BANNING          JOHN
 .19  IRVING           JOAN
 .00  BLACKWOOD        ROSEMARIE
-.14  GREENSPAN        MARY
```

SQRT: Calculating the Square Root

Available Operating Systems: All

Available Languages: reporting, Maintain

The SQRT function calculates the square root of an argument.

Syntax How to Calculate the Square Root

```
SQRT(argument)
```

where:

argument

Numeric

Is the value for which the square root is calculated. You may supply the actual value, the name of a field that contains the value, or an expression that returns the value. If you use an expression, use parentheses as needed to ensure the correct order of evaluation.

Example Calculating Square Root of Movies' List Price

In the following example, 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 FOCUS output is:

```
TITLE                                    LISTPR     SQRT_LISTPR
-----                                    ------     -----------
ALL THAT JAZZ                             19.98            4.47
CABARET                                   19.98            4.47
CHORUS LINE, A                            14.98            3.87
FIDDLER ON THE ROOF                       29.95            5.47
```

CHAPTER 8

System Functions

Topics:

- Alphabetical List of System Functions

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

FEXERR: Retrieving an Error Message

Available Operating Systems: AS/400, OpenVMS, S/390, UNIX, VM/CMS, Windows NT/2000

Available Languages: reporting, Maintain

The FEXERR function retrieves an error message. This function is especially useful in procedures using commands that suppress the display of output messages.

Error messages may consist of up to four lines of text; the first line contains the message and the remaining three may contain a detailed explanation, if one exists. The FEXERR function 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 five digits long.

'A72'

Is the format of the output value, enclosed in single quotation marks. The format is A72 because the maximum length of a FOCUS error message is 72 characters.

Note: In Maintain, you must supply the field name instead.

Example Retrieving an Error Message

In the following example, FEXERR retrieves the error message whose number is contained in the &ERR variable, in this case 650. The result is stored in a field with 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

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

The FINDMEM function, used on OS/390 or batch only, determines if a specific member of a partitioned data set (PDS) exists. This function is especially useful in Dialogue Manager procedures.

In order to use this function, the PDS must be allocated to a ddname because the ddname is specified in the function call. You can search multiple partitioned data sets with one function call if the partitioned data sets 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 argument must be eight characters long or be a variable. If you are using a literal for this argument, enclose it in single quotation marks. If the literal is less than eight characters, pad it with trailing blanks.

member

A8

Is the member you are searching for. This argument must be eight characters long. If you are using a literal for this argument that has less than eight characters, pad the literal with trailing blanks.

outfield

A1

Is the field to which the result is returned, 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. This can occur because 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).

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

Example Finding the Member of a Partitioned Data Set

In the following example, FINDMEM searches for the EMPLOYEE Master File in the PDS allocated to ddname MASTER, and returns the result to a field with 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

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

The GETPDS function determines if a specific member of a partitioned data set (PDS) exists, and returns the PDS name. This function is especially useful in Dialogue Manager procedures.

In order to use this function, the PDS must be allocated to a ddname because the ddname is specified in the function call. You can search multiple partitioned data sets with one function call if the partitioned data sets are concatenated to one ddname.

Note: The FINDMEM function is almost identical to the GETPDS function, except that the GETPDS function provides either the PDS name or different status codes.

Syntax

How to Determine if a Member Exists

`GETPDS(`*`ddname, member, outfield`*`)`

where:

`ddname`

A8

Is the ddname to which the PDS is allocated. This argument must be an eight character literal enclosed in single quotation marks, or a variable that contains the ddname. If the literal is less than eight characters long, you must pad it with trailing blanks.

`member`

A8

Is the member you are searching for. This argument must be eight characters long. If you are using a literal for this argument that has less than eight characters, you must pad the literal with trailing blanks.

`outfield`

A44

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. The value returned to *outfield* is one of the following:

`PDS name` is the PDS name that contains the specified member, if it exists.

`*D` is returned if the ddname is not assigned (allocated) to a data set.

`*M` is returned if the member does not exist in the PDS.

`*E` is returned if an error occurs. This can happen because the data set allocated to the ddname is not a PDS (and may be a sequential file).

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

Example Determining if a Member Exists

In the following example, GETPDS searches for the member specified by &MEMBER in the PDS specified by &DDNAME, and returns the result to the &PNAME variable.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '                                                ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,'A44');
-IF &PNAME EQ '*D' THEN GOTO DDNOAL;
-IF &PNAME EQ '*M' THEN GOTO MEMNOF;
-IF &PNAME EQ '*E' THEN GOTO DDERROR;
-*
-TYPE MEMBER &MEMBER IS FOUND IN
-TYPE THE PDS &PNAME
-TYPE ALLOCATED TO &DDNAME
-*
-EXIT
-DDNOAL
-*
-TYPE DDNAME &DDNAME NOT ALLOCATED
-*
-EXIT
-MEMNOF
-*
-TYPE MEMBER &MEMBER NOT FOUND UNDER DDNAME &DDNAME
-*
-EXIT
-DDERROR
-*
-TYPE ERROR IN GETPDS; DATA SET PROBABLY NOT A PDS.
-*
-EXIT
```

Output similar to the following is produced:

```
MEMBER EMPLOYEE IS FOUND IN
THE PDS USER1.MASTER.DATA
ALLOCATED TO MASTER
```

Example Using GETPDS With TED

In the following example, GETPDS searches for the member specified by &MEMBER in the PDS specified by &DDNAME, and returns the result to the value specified by &PNAME. Then the TED editor enables you to edit the member. The ddnames are allocated earlier in the session: the production PDS is allocated to the ddname MASTER; your local PDS to ddname MYMASTER.

```
-* If the MASTER file in question is in the 'production' pds, it must
-* be copied to a 'local' pds, which has been allocated previously to the
-* ddname MYMASTER before any changes can be made.
-* Assume the MASTER in question is supplied via a -CRTFORM, with
-* a length of 8 characters, as &MEMBER
-*
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = &MEMBER;
-SET &PNAME = '                                              ';
-SET &PNAME = GETPDS(&DDNAME,&MEMBER,'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 FOCUS session, allocate the ddnames:

```
>  > tso alloc f(master) da('prod720.master.data') shr
>  > tso alloc f(mymaster) da('user1.master.data') shr
```

After you execute the procedure, specify the EMPLOYEE member. The member is copied to your local PDS and you enter TED.

```
PLEASE SUPPLY VALUES REQUESTED

MEMBER=  > employee


MYMASTER(EMPLOYEE)                    SIZE=37    LINE=0

00000 * * * TOP OF FILE * * *
00001 FILENAME=EMPLOYEE, SUFFIX=FOC
00002 SEGNAME=EMPINFO,  SEGTYPE=S1
00003  FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
00004  FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
00005  FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,      $
00006  FIELDNAME=HIRE_DATE,    ALIAS=HDT,     FORMAT=I6YMD,    $
00007  FIELDNAME=DEPARTMENT,   ALIAS=DPT,     FORMAT=A10,      $
```

Example Using GETPDS With Query Commands

Suppose you wanted to review the attributes of the PDS that contains a specific member. This Dialogue Manager procedure searches for the EMPLOYEE member in the PDS allocated to the ddname MASTER and, based on its existence, allocates the PDS name to the ddname TEMPMAST. Dialogue Manager 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
```

A sample execution follows:

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

When you execute this procedure, it searches the PDS allocated to ddname MASTER for the member EMPLOYEE. Since the procedure locates the member, it displays the attributes for the MASTER PDS.

GETUSER: Retrieving a User ID

Available Operating Systems: All

Available Languages: reporting, Maintain

The GETUSER function retrieves the user ID of the connected user.

In OS/390 FOCUS, it can also retrieve the name of an S/390 batch job if you run it from the batch job. To retrieve a logon ID for MSO, use the MSOINFO function described in the *FOCUS for IBM Mainframe Multi-Session Option Installation and Technical Reference Guide*.

Syntax How to Retrieve a User ID

```
GETUSER(outfield)
```

where:

outfield

A8

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks. The field must be 8 bytes long.

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

Example Retrieving a User ID

In the following example, GETUSER retrieves the user ID of the person executing the request.

```
DEFINE FILE EMPLOYEE
USERID/A8 WITH EMP_ID = GETUSER(USERID);
END
TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES'
BY DEPARTMENT
HEADING
"SALARY REPORT RUN FROM USERID: <USERID"
" "
END
```

The output is:

```
SALARY REPORT RUN FROM USERID: USER1

DEPARTMENT   TOTAL SALARIES
----------   --------------
MIS             $108,002.00
PRODUCTION      $114,282.00
```

HHMMSS: Returning the Current Time

Available Operating Systems: All

Available Languages: reporting, Maintain

The HHMMSS function retrieves the current time from the operating system and returns the time as an eight-character string, separating the hours minutes and seconds with periods for reporting and colons for Maintain.

For details on the HHMMSS functions, see Chapter 5, *Date and Time Functions*.

MVSDYNAM: Passing a DYNAM Command to the Command Processor

Available Operating Systems: OS/390

Available Languages: reporting, Maintain

The MVSDYNAM function transfers a FOCUS DYNAM command to the DYNAM command processor. This is useful to pass allocation commands to the processor in compiled MODIFY procedures 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. This can be the 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 length of the command in characters, between 1 and 256.

outfield

I4

Is the field to which the result is returned, 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 error number corresponding to DYNAM failure.

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

Example Passing a DYNAM Command to the Processor

In the following request, MVSDYNAM transfers the DYNAM FREE command to the processor. Query commands display the results before and after the DYNAM FREE command is specified. The successful return code of zero (0) is stored in the RES field.

```
-* THE RESULT OF ? TSO DDNAME CAR WILL BE BLANK AFTER ENTERING
-* 'FREE FILE CAR' AS YOUR COMMAND
DYNAM ALLOC FILE CAR DS USER1.CAR.FOCUS SHR REUSE
? TSO DDNAME CAR
-RUN
-PROMPT &XX.ENTER A SPACE TO CONTINUE.
MODIFY FILE CAR
COMPUTE LINE/A60=;
        RES/I4 = 0;
CRTFORM
" ENTER DYNAM COMMAND BELOW:"
" <LINE>"
COMPUTE
RES = MVSDYNAM(LINE, 60, RES);
GOTO DISPLAY

 CASE DISPLAY
 CRTFORM LINE 1
" THE RESULT OF DYNAM WAS <D.RES"
GOTO EXIT
ENDCASE
DATA
END
? TSO DDNAME CAR
```

The first query command displays the allocation that results from the DYNAM ALLOCATE command.

```
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 the Enter key to continue. Then enter the DYNAM FREE command. (The DYNAM keyword is assumed.)

```
ENTER DYNAM COMMAND BELOW:
 free file car
```

The function successfully transfers the DYNAM FREE command to the processor and the return code displays.

```
THE RESULT OF DYNAM WAS     0
```

Press the Enter key to continue. The second query command indicates that the allocation has been freed.

```
DDNAME      =  CAR
DSNAME      =
DISP        =
DEVICE      =
VOLSER      =
DSORG       =
RECFM       =
SECONDARY   =  ****
ALLOCATION  =
BLKSIZE     =         0
LRECL       =         0
TRKTOT      =         0
EXTENTSUSED =         0
BLKSPERTRK  =         0
TRKSPERCYL  =         0
CYLSPERDISK =         0
BLKSWRITTEN =         0
>
```

TODAY: Returning the Current Date

Available Operating Systems: All

Available Languages: reporting, Maintain

The TODAY function retrieves the current date from the system in the format MM/DD/YY or MM/DD/YYYY for reporting, and in YY/MM/DD or YYYY/MM/DD for Maintain.

For details on the TODAY function, see Chapter 5, *Date and Time Functions*.

APPENDIX A

Creating Your Own Subroutines

Topics:

- Process Overview
- Considerations for Writing Subroutines
- Compilation and Storage
- Testing the Subroutine
- Example of a Custom Subroutine: The MTHNAM Subroutine
- Subroutines Written in REXX

This topic discusses how to create your own private collection of subroutines to use with FOCUS.

Process Overview

The process of creating a subroutine involves four steps:

1. Write the subroutine for FOCUS the same way you would for a program. Use any language that supports subroutine calls; among the most common languages are FORTRAN, COBOL, PL/I, Assembler, and C.
2. Store the subroutine in a separate file; do not include it in the main program.
3. Compile the subroutine. In OS/390, link-edit it; in VM/CMS, add the subroutine to a load library using the GENSUBLL command.
4. Test the subroutine; specify it in a FOCUS command, report request, or procedure.

For example, suppose you write a program named INTCOMP that calculates the amount of money in an account earning simple interest. The program reads a record, tests if the data is acceptable, and then calls a subroutine called SIMPLE that computes the amount of money. The program and the subroutine are stored together in the same file.

The program and the subroutine shown here are written in pseudocode (a method of representing computer code in a general way):

```
Begin program INTCOMP.
Execute this loop until end-of-file.
    Read next record, fields: PRINCPAL, DATE_PUT, YRRATE.
    If PRINCPAL is negative or greater than 100,000,
        reject record.
    If DATE_PUT is before January 1, 1975, reject record.
    If YRRATE is negative or greater than 20%, reject record.
    Call subroutine SIMPLE (PRINCPAL, DATE_PUT, YRRATE, TOTAL).
    Print PRINCPAL, YEARRATE, TOTAL.
End of loop.
End of program.

Subroutine SIMPLE (AMOUNT, DATE, RATE, RESULT).
Retrieve today's date from the system.
Let NO_DAYS = Days from DATE until today's date.
Let DAY_RATE = RATE / 365 days in a year.
Let RESULT = AMOUNT * (NO_DAYS * DAY_RATE + 1).
End of subroutine.
```

If you move the SIMPLE subroutine into a file separate from the main program and compile it, you can call the subroutine from FOCUS. The following report request shows how much money employees would accrue if they invested their salaries in accounts paying 12%:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME DAT_INC SALARY AND COMPUTE
    INVESTED/D10.2 = SIMPLE (SALARY, DAT_INC, 0.12, INVESTED);
BY EMP_ID
END
```

Note: The subroutine is designed to return only the amount of the investment, not today's date. This is because a subroutine can return only a single value to FOCUS each time it is called.

Considerations for Writing Subroutines

When you write a subroutine for FOCUS, there are requirements and limits that you need to consider. The topic provides information about:

- Naming conventions
- Argument considerations
- Programming considerations
- Language considerations
- A programming technique that uses entry points. Entry points enable you to use one algorithm to produce different results.
- A programming technique that allows multiple subroutine calls. Multiple calls enable the subroutine to process more than 28 arguments.

Naming Conventions

The subroutine name may consist of up to eight characters, unless the language you are using to write the subroutine supports a shorter naming convention. Each character can be a letter or number. The first character of the name must be a letter (A-Z). Special symbols are not permitted.

Argument Considerations

When you create your arguments, consider these points:

- **The argument maximum.** Subroutine calls in FOCUS may contain up to 28 arguments. However, you can bypass this restriction if you create a subroutine that accepts multiple calls, as described in *Programming Technique: Subroutines With More Than 28 Arguments* on page A-9.
- **Types of arguments.** Subroutine calls can serve as arguments in other subroutine calls or in FOCUS functions. For types of acceptable arguments and rules, see Chapter 2, *Accessing and Invoking a Function.*
- **Input arguments.** FOCUS passes input arguments to subroutines using standard conventions. Register 1 points to the list of argument addresses. Each address is a full word.
- **Output arguments.** Subroutines may return only one output argument to the FOCUS request. Place this argument last in the subroutine argument list. You can choose any format for the output argument except in Dialogue Manager statements.
- **Internal processing.** When you specify values for arguments and FOCUS passes the arguments to a subroutine,
  - Alphanumeric arguments remain unchanged.
  - Numeric arguments are converted to 8-byte, double-precision data (except in -CMS RUN and -MVS RUN statements and amper variables, as discussed below).

    Various languages represent double-precision fields as declarations:

Language	Declaration
Assembler	DS, D
C	Double
COBOL	COMP-2
FORTRAN	REAL*8
PL/I	DECIMAL FLOAT (16)

- **Dialogue Manager requirements.** If you are writing a subroutine specifically for Dialogue Manager, you may need to code your subroutine to perform conversion for these situations:
  - Operating system -RUN statements. FOCUS passes all arguments from -CMS RUN, -TSO RUN, and -MVS RUN statements as alphanumeric data. If your subroutine requires numeric arguments, you may choose to have your subroutine convert these arguments into numeric format. Otherwise, the user can use the ATODBL subroutine to convert the arguments into double-precision format before passing them to the subroutine. The ATODBL subroutine is described in Chapter 6, *Format Conversion Functions*.
  - Operating system -RUN statements and output argument format. If the subroutine is called from a -CMS or -TSO RUN statement, the output argument is stored in the output variable in numeric format. Since FOCUS cannot interpret data stored in Dialogue Manager variables in numeric format, the data is unreadable. To prevent this, have your subroutine convert the output value into a character string.
  - -SET and output argument format. If the output argument is in numeric format, the -SET statement truncates the output value to an integer, converts it to a character string, and stores the value in a specified amper variable. To prevent this, have your subroutine convert the output value into a character string. This enables the numeric value to be passed to Dialogue Manager without being truncated to an integer.

Programming Considerations

When you plan your programming requirements, consider these points:

- Write the subroutine as a proper subroutine, not as a function.
- If the subroutine initializes variables, it must initialize them each time it is executed (serial reusability).
- Since a single FOCUS request may execute a subroutine hundreds or even thousands of times, code the subroutine as efficiently as possible.
- If you create your own subroutines in text files or text libraries, the subroutine must be 31-bit addressable.

Language Considerations

Language considerations include:

- **Available memory.**

  If you write the subroutine in a language that brings libraries into memory (for example, FORTRAN and COBOL), the libraries reduce the amount of memory available to the subroutine.

- **FORTRAN input/output operations (I/O).**

  In VM/CMS, FOCUS does not support FORTRAN input/output operations. If a subroutine written in FORTRAN must read or write data, write the I/O portions in a separate subroutine in another language.

  In TSO, FOCUS does support FORTRAN input/output operations.

- **PL/I notes:**

  - Do not use the RETURNS attribute.
  - Include the following attribute in the procedure (PROC) statement:

    ```
    OPTIONS (COBOL)
    ```

  - Declare alphanumeric arguments received from FOCUS requests as

    ```
    CHARACTER (n)
    ```

    where *n* is the field length as defined by the FOCUS request. Do not use the VARYING attribute.

  - Declare numeric arguments received from FOCUS requests as

    ```
    DECIMAL FLOAT (16)
    ```

    or

    ```
    BINARY FLOAT (53)
    ```

- The format of the output argument to be returned to the FOCUS request depends on how the format is described in the DEFINE or COMPUTE commands:

FOCUS Format	PL/I Declaration
A*n*	`CHARACTER (n)` (Do not use the VARYING attribute.)
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)

- Declare variables that are not arguments with the STATIC attribute. This avoids dynamically allocating these variables every time the subroutine is executed.

- **C language notes:**
  - Do not return a value with the return statement.
  - Declare double-precision fields as ‘double’.
  - The format of the output parameter to be returned to the FOCUS request depends on how the format is defined in the request, as shown by the chart below:

FOCUS Format	C Declaration
A*n*	`char *xxx n` (**Note:** Alphabetical fields are not terminated with a null byte and, therefore, cannot be processed by many of the string manipulation subroutines in the run-time library.)
I	`long *xxx`
F	`float *xxx`
D	`double *xxx`
P	No equivalent in C.

Programming Technique: Entry Points

Normally, subroutines are executed starting from their first statement. However, they can be executed starting from any place in their code if you designate that place as an *entry point*. (How you designate entry points depends on the language you are using.) Each entry point has a name.

To execute a subroutine at an entry point, specify the entry name in the subroutine call instead of the subroutine name. The general syntax is:

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

Entry points enable a subroutine to use one basic algorithm to produce different results. For example, the DOWK subroutine calculates the days of the week on which dates fall. When you specify the subroutine name DOWK, you obtain a 3-letter abbreviation of the day. If you specify the entry name DOWKL, you obtain the full name. The calculation, however, is the same.

Example Entry Point Example

This example illustrates how entry points work. The FTOC subroutine, written in pseudocode below, converts Fahrenheit temperatures to Centigrade. The entry point FTOK (designated by the Entry statement) sets a flag that causes 273 to be subtracted from the Centigrade temperature (Kelvin temperature). The subroutine is:

```
Subroutine FTOC (FAREN, CENTI).
Let FLAG = 0.
Go to label X.
Entry FTOK (FAREN, CENTI).
Let FLAG = 1.
Label X.
Let CENTI = (5/9) * (FAREN - 32).
If FLAG = 1 then CENTI = CENTI - 273.
Return.
End of subroutine.
```

Here is a shorter way to write the subroutine. Notice that the *kelv* output argument listed for the entry point is different from the *centi* output argument listed at the beginning of the subroutine:

```
Subroutine FTOC (FAREN, CENTI).
Entry FTOK (FAREN, KELV).
Let CENTI = (5/9) * (FAREN - 32).
KELV = CENTI - 273.
Return.
End of Subroutine.
```

To obtain the Centigrade temperature, specify the subroutine name FTOC in the subroutine call. For example:

```
CENTIGRADE/D6.2 = FTOC (TEMPERATURE, CENTIGRADE);
```

To obtain the Kelvin temperature, specify the entry name FTOK in the subroutine call. For example:

```
KELVIN/D6.2 = FTOK (TEMPERATURE, KELVIN);
```

Note: In VM/CMS, subroutines can be executed from their entry points only if the subroutines are stored in libraries. You must specify these libraries in the GLOBAL command, as described in *VM/CMS: Compilation and Storage* on page A-13.

Programming Technique: Subroutines With More Than 28 Arguments

Subroutine call syntax cannot specify more than 28 arguments, including the output argument. To process more than 28 arguments, you must write the subroutine so that the user can specify two or more call statements to pass the arguments to the subroutine.

We recommend the following technique for writing subroutines with multiple call statements:

1. Divide the subroutine into segments. Each segment will receive the arguments passed by one corresponding subroutine call.

   The argument list in the beginning of your subroutine must represent the same number of arguments in the subroutine call, including a call number argument and an output argument.

   You may process some of the arguments as dummy arguments if you have an unequal number of arguments. For example, if you divide 32 arguments among six segments, the each segment processes six arguments; the sixth segment processes two arguments and four dummy arguments.

2. Include a statement at the beginning of the subroutine that reads the call number (first argument) and branches to a corresponding segment. Each segment processes the arguments from one call. (For example, number 1 branches to the first segment, number 2 to the second segment, and so on.)

3. Have each segment store the arguments it receives in other variables (which can be processed by the last segment) or accumulate them in a running total.

   End each segment with a statement returning control back to the FOCUS request (RETURN statement).

4. The last segment returns the final output value to the FOCUS request.

The following sample of pseudocode illustrates the four steps:

```
1.  Subroutine name (num, input1, input2, input3, input4, outfield).
2.  If NUM is 1 then goto label ONE
    else goto label TWO.

    Label ONE.
3.  Let variable = input1 + input2.
    Return.

4.  Label TWO
    LET outfield = variable + input3 + input4
    Return
    End of subroutine
```

Note: You can also use the entry point technique, described in *Programming Technique: Entry Points* on page A-8, to write subroutines that process more than 28 arguments.

How to Use Subroutines With Multiple Call Statements

Syntax

To use a subroutine that requires more than 28 arguments, you must specify two or more call statements to pass the arguments to the subroutine.

The syntax for calling a subroutine with multiple call statements is

```
dummy = subroutine (1, group1, dummy);
dummy = subroutine (2, group2, dummy);
    .
    .
    .
outfield = subroutine (n, groupn, outfield);
```

where:

dummy

Is either the name of a dummy field or its format, enclosed in single quotation marks. It must have the same format as the *outfield* argument.

Note: Do not specify the *dummy* argument for the last call statement; use the *outfield* argument.

subroutine

Is the name of the subroutine, up to eight characters long, depending on your programming language.

n

Is a number that identifies each subroutine call. It must be the first argument in each subroutine call. The subroutine uses this call number to branch to segments of code.

group1...

Are lists of input arguments passed by each subroutine call. Each group contains the same number of arguments, but no more than 26 arguments.

```
26 + call number + output =28
```

outfield

Is the output field that contains the value returned by the subroutine. It is the fieldname of the field that contains the output or the format of the output value, enclosed in single quotation marks, depending on the application. It is last argument in the last call.

Note:

- Each subroutine call contains the same number of arguments. This is because the argument list in each call must correspond to the argument list in the beginning of the subroutine. The last call may contain several dummy arguments.
- Subroutines may require additional arguments as determined by the programmer who created the subroutine.

Example Creating a Subroutine With 32 Input Arguments

This example illustrates how to create a subroutine with 32 input arguments using the recommended technique. It also shows how the subroutine is specified in a DEFINE command.

The ADD32 subroutine, written in pseudocode, sums 32 numbers. It is divided into six segments, each of which adds six numbers from a subroutine call. (The total number of input arguments is 36 but the last four are dummy arguments.) The sixth segment adds two arguments to the SUM variable and returns the final output value. The sixth segment does not process any values supplied for the four dummy arguments.

The subroutine is:

```
Subroutine ADD32 (NUM, A, B, C, D, E, F, TOTAL).
If NUM is 1 then goto label ONE
else if NUM is 2 then goto label TWO
else if NUM is 3 then goto label THREE
else if NUM is 4 then goto label FOUR
else if NUM is 5 then goto label FIVE
else goto label SIX.

Label ONE.
Let SUM = A + B + C + D + E + F.
Return.

Label TWO
Let SUM = SUM + A + B + C + D + E + F
Return

Label THREE
Let SUM = SUM + A + B + C + D + E + F
Return

Label FOUR
Let SUM = SUM + A + B + C + D + E + F
Return

Label FIVE
Let SUM = SUM + A + B + C + D + E + F
Return

Label SIX
LET TOTAL = SUM + A + B
Return
End of subroutine
```

To use the ADD32 subroutine, list all six call statements; each call specifying six numbers. The last four numbers, represented by zeroes, are dummy arguments. In this example, the DEFINE command stores the total of the 32 numbers in the SUM32 field.

```
DEFINE FILE EMPLOYEE
DUMMY/D10 = ADD32 (1, 5, 7, 13, 9, 4, 2, DUMMY);
DUMMY/D10 = ADD32 (2, 5, 16, 2, 9, 28, 3, DUMMY);
DUMMY/D10 = ADD32 (3, 17, 12, 8, 4, 29, 6, DUMMY);
DUMMY/D10 = ADD32 (4, 28, 3, 22, 7, 18, 1, DUMMY);
DUMMY/D10 = ADD32 (5, 8, 19, 7, 25, 15, 4, DUMMY);
SUM32/D10 = ADD32 (6, 3, 27, 0, 0, 0, 0, SUM32);
END
```

Compilation and Storage

Once you have written your subroutine, you need to compile and store it. This topic discusses compiling and storing your subroutine for VM/CMS and OS/390.

VM/CMS: Compilation and Storage

On VM/CMS, compile the subroutine and use the GENSUBLL command to add the compiled object code to a load library (filetype LOADLIB). Enter:

```
GENSUBLL ?
```

to display for online information about the command. Do not store subroutine in the FUSELIB load library (FUSELIB LOADLIB), as it may be overwritten when your site installs the next release of FOCUS.

You may also compile the subroutine and store the compiled object code either as a text file (filetype TEXT), or as a member in a text library (filetype TXTLIB). Do not store it in the FUSELIB text library (FUSELIB TXTLIB), as it may be overwritten when your site installs the next release of FOCUS.

Individual text files are easier to maintain and control. Text libraries, on the other hand, enable you to build different entry points into the subroutine (as shown in *Programming Technique: Entry Points* on page A-8). Note that there are two VM/CMS commands regarding text libraries:

- The TXTLIB command allows you to create, add to, and delete text libraries.
- The GLOBAL TXTLIB command allows users to specify text libraries to gain access to their subroutines.

If the subroutine is written in PL/I, append this line at the end of the text file

```
ENTRY subroutine
```

where:

subroutine
Is the name of the subroutine. You can do this using your system editor.

Make sure that any subroutines that your subroutine calls are also compiled and placed in text files or libraries.

OS/390: Compilation and Storage

On OS/390, compile and link-edit the subroutine and store the module in a load library. If your subroutine calls other subroutines, compile and link-edit all the subroutines together in a single module.

If the subroutine is written in PL/I, include this link-editor control statement when link-editing the subroutine

```
ENTRY subroutine
```

where:

subroutine
Is the name of the subroutine.

Do not store the subroutine in the FUSELIB load library (FUSELIB.LOAD), as it may be overwritten when your site installs the next release of FOCUS.

Testing the Subroutine

Once you have successfully compiled your subroutine, access it and test it. In order to access the subroutine, you need to issue the GLOBAL command for VM/CMS or the ALLOCATE command for OS/390.

If an error occurs during your testing, check to see if the error is in the FOCUS request or in the subroutine. If you are uncertain about its source, apply this test:

1. Write a dummy subroutine that has the same arguments but only returns a constant.
2. Execute the request with the dummy subroutine.

   If the request executes the dummy subroutine normally, the error is in your subroutine. If the request still generates an error, the error is in the request.

If you intend to make your subroutine available to other users, be sure to document what your subroutine does, what the arguments are, what formats they have, and in what order they must appear in the FOCUS subroutine call.

Example of a Custom Subroutine: The MTHNAM Subroutine

This topic illustrates how a subroutine can be written in FORTRAN, COBOL, PL/I, BAL Assembler, and C, and then executed in a FOCUS request. The subroutine, called MTHNAM, converts a number from 1 to 12 to the full name of the corresponding month (from January to December).

The subroutine performs the following:

1. The subroutine receives the input argument from the FOCUS request as a double-precision number.
2. It adds .000001 to the number. This compensates for rounding errors. (Rounding errors can occur since floating-point numbers are approximations and may be inaccurate in the last significant digit.)
3. It moves the number into an integer field.
4. If the number is less than 1 or greater than 12, it changes the number to 13.
5. It defines a 13-element array containing the names of the months. The last element is an error message.
6. It sets the index of the array equal to the number in the integer field. It then places the corresponding array element into the output argument. If the number is 13, the argument contains the error message.
7. It passes the output argument back to FOCUS.

The MTHNAM Subroutine Written in FORTRAN

This is a FORTRAN version of the MTHNAM subroutine. The fields are:

MTH
: Is the double-precision number passed by FOCUS.

MONTH
: Is the name of the month passed back to FOCUS. Since the character string 'September' contains nine letters, MONTH is a 3-element array. The subroutine passes the three elements back to FOCUS; FOCUS concatenates them into one field.

A
: Is a 2-dimensional, 13 by 3 array containing the names of the months. The last three elements contain the error message.

IMTH
: Is the integer representing the month.

The program is:

```
  SUBROUTINE MTHNAM (MTH,MONTH)
  REAL*8     MTH
  INTEGER*4  MONTH(3),A(13,3),IMTH
  DATA
 +     A( 1,1)/'JANU'/, A( 1,2)/'ARY '/, A( 1,3)/'    '/,
 +     A( 2,1)/'FEBR'/, A( 2,2)/'UARY'/, A( 2,3)/'    '/,
 +     A( 3,1)/'MARC'/, A( 3,2)/'H   '/, A( 3,3)/'    '/,
 +     A( 4,1)/'APRI'/, A( 4,2)/'L   '/, A( 4,3)/'    '/,
 +     A( 5,1)/'MAY '/, A( 5,2)/'    '/, A( 5,3)/'    '/,
 +     A( 6,1)/'JUNE'/, A( 6,2)/'    '/, A( 6,3)/'    '/,
 +     A( 7,1)/'JULY'/, A( 7,2)/'    '/, A( 7,3)/'    '/,
 +     A( 8,1)/'AUGU'/, A( 8,2)/'ST  '/, A( 8,3)/'    '/,
 +     A( 9,1)/'SEPT'/, A( 9,2)/'EMBE'/, A( 9,3)/'R   '/,
 +     A(10,1)/'OCTO'/, A(10,2)/'BER '/, A(10,3)/'    '/,
 +     A(11,1)/'NOVE'/, A(11,2)/'MBER'/, A(11,3)/'    '/,
 +     A(12,1)/'DECE'/, A(12,2)/'MBER'/, A(12,3)/'    '/,
 +     A(13,1)/'**ER'/, A(13,2)/'ROR*'/, A(13,3)/'*   '/
  IMTH=MTH+0.000001
  IF (IMTH .LT. 1 .OR. IMTH .GT. 12) IMTH=13
  DO 1 I=1,3
1 MONTH(I)=A(IMTH,I)
  RETURN
  END
```

The MTHNAM Subroutine Written in COBOL

This is a COBOL version of the MTHNAM subroutine. The fields are:

`MONTH-TABLE`
: Is a field containing the names of the months and the error message.

`MLINE`
: Is a 13-element array that redefines the MONTH-TABLE field. Each element (called A) contains the name of a month; the last element contains the error message.

`A`
: Is one element in the MLINE array.

`IX`
: Is an integer field that indexes MLINE.

`IMTH`
: Is the integer representing the month.

`MTH`
: Is the double-precision number passed by FOCUS.

`MONTH`
: Is the name of the month passed back to FOCUS.

The program is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MTHNAM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STORAGE SECTION.
   01 MONTH-TABLE.
     05 FILLER PIC X(9) VALUE 'JANUARY  '.
     05 FILLER PIC X(9) VALUE 'FEBRUARY '.
     05 FILLER PIC X(9) VALUE 'MARCH    '.
     05 FILLER PIC X(9) VALUE 'APRIL    '.
     05 FILLER PIC X(9) VALUE 'MAY      '.
     05 FILLER PIC X(9) VALUE 'JUNE     '.
     05 FILLER PIC X(9) VALUE 'JULY     '.
     05 FILLER PIC X(9) VALUE 'AUGUST   '.
     05 FILLER PIC X(9) VALUE 'SEPTEMBER'.
     05 FILLER PIC X(9) VALUE 'OCTOBER  '.
     05 FILLER PIC X(9) VALUE 'NOVEMBER '.
     05 FILLER PIC X(9) VALUE 'DECEMBER '.
     05 FILLER PIC X(9) VALUE '**ERROR**'.
   01  MLIST REDEFINES MONTH-TABLE.
     05  MLINE OCCURS 13 TIMES INDEXED BY IX.
         10 A  PIC X(9).
   01  IMTH    PIC S9(5) COMP.
LINKAGE SECTION.
   01  MTH     COMP-2.
   01  MONTH   PIC X(9).
PROCEDURE DIVISION USING MTH, MONTH.
BEG-1.
      ADD 0.000001 TO MTH.
      MOVE MTH TO IMTH.
      IF IMTH < +1 OR > 12
        SET IX TO +13
      ELSE
        SET IX TO IMTH.
      MOVE A (IX) TO MONTH.
      GOBACK.
```

The MTHNAM Subroutine Written in PL/I

This is a PL/I version of the MTHNAM subroutine. The fields are:

MTHNUM
: Is the double-precision number passed by FOCUS.

FULLMTH
: Is the name of the month passed back to FOCUS.

MONTHNUM
: Is the integer representing the month.

MONTH_TABLE
: A 13-element array containing the names of the months. The last element contains the error message.

The program is:

```
MTHNAM:   PROC(MTHNUM,FULLMTH) OPTIONS(COBOL);
DECLARE  MTHNUM  DECIMAL FLOAT (16) ;
DECLARE  FULLMTH CHARACTER (9) ;
DECLARE  MONTHNUM FIXED BIN (15,0)  STATIC ;
DECLARE  MONTH_TABLE(13) CHARACTER (9)   STATIC
                        INIT  ('JANUARY',
                               'FEBRUARY',
                               'MARCH',
                               'APRIL',
                               'MAY',
                               'JUNE',
                               'JULY',
                               'AUGUST',
                               'SEPTEMBER',
                               'OCTOBER',
                               'NOVEMBER',
                               'DECEMBER',
                               '**ERROR**') ;
  MONTHNUM = MTHNUM + 0.00001 ;
  IF MONTHNUM < 1   MONTHNUM > 12 THEN
           MONTHNUM = 13 ;
  FULLMTH = MONTH_TABLE(MONTHNUM) ;
RETURN;
END MTHNAM;
```

The MTHNAM Subroutine Written in BAL Assembler

This is a BAL Assembler version of the MTHNAM subroutine.

```
         START 0
         STM   14,12,12(13)      save registers
         BALR  12,0              load base reg
         USING *,12
*
         L     3,0(0,1)          load addr of first arg into R3
         LD    4,=D'0.0'         clear out FPR4 and FPR5
         LE    6,0(0,3)          FP number in FPR6
         LPER  4,6               abs value in FPR4
         AW    4,=D'0.00001'     add rounding constant
         AW    4,DZERO           shift out fraction
         STD   4,FPNUM           move to memory
         L     2,FPNUM+4         integer part in R2
         TM    0(3),B'10000000'  check sign of original no
         BNO   POS               branch if positive
         LCR   2,2               complement if negative
*
POS      LR    3,2               copy month number into R3
         C     2,=F'0'           is it zero or less?
         BNP   INVALID           yes. so invalid
         C     2,=F'12'          is it greater than 12?
         BNP   VALID             no. so valid
INVALID  LA    3,13(0,0)         set R3 to point to item @13 (error)
*
VALID    SR    2,2               clear out R2
         M     2,=F'9'           multiply by shift in table
*
         LA    6,MTH(3)          get addr of item in R6
         L     4,4(0,1)          get addr of second arg in R4
         MVC   0(9,4),0(6)       move in text
*
         LM    14,12,12(13)      recover regs
         BR    14                return
*
```

```
         DS   0D                  alignment
FPNUM    DS   D                   floating point number
DZERO    DC   X'4E00000000000000' shift constant
MTH      DC   CL9'dummyitem'      month table
         DC   CL9'JANUARY'
         DC   CL9'FEBRUARY'
         DC   CL9'MARCH'
         DC   CL9'APRIL'
         DC   CL9'MAY'
         DC   CL9'JUNE'
         DC   CL9'JULY'
         DC   CL9'AUGUST'
         DC   CL9'SEPTEMBER'
         DC   CL9'OCTOBER'
         DC   CL9'NOVEMBER'
         DC   CL9'DECEMBER'
         DC   CL9'**ERROR**'
         END  MTHNAM
```

The MTHNAM Subroutine Written in C

This is a C language version of the MTHNAM subroutine.

```
void mthnam(double *,char *);
void mthnam(mth,month)
double *mth;
char *month;
{
char *nmonth[13] = {"January  ",
                    "February ",
                    "March    ",
                    "April    ",
                    "May      ",
                    "June     ",
                    "July     ",
                    "August   ",
                    "September",
                    "October  ",
                    "November ",
                    "December ",
                    "**Error**"};
int imth, loop;
imth = *mth + .00001;
imth = (imth < 1 ¦¦ imth > 12 ? 13 : imth);
for (loop=0;loop < 9;loop++)
 month[loop] = nmonth[imth-1][loop];
}
```

The MTHNAM Subroutine Called by a FOCUS Request

The following example demonstrates how a FOCUS request uses the MTHNAM subroutine. The DEFINE command extracts the month portion of the pay date and executes the MTHNAM subroutine to convert it into the full name of the month. The name is stored in the PAY_MONTH field. The report request prints the monthly pay of Alfred Stevens.

The request is as follows:

```
DEFINE FILE EMPLOYEE
MONTH_NUM/M = PAY_DATE;
PAY_MONTH/A12 = MTHNAM (MONTH_NUM, PAY_MONTH);
END
TABLE FILE EMPLOYEE
PRINT PAY_MONTH GROSS
BY EMP_ID BY FIRST NAME BY LAST_NAME
BY PAY_DATE
IF LN IS STEVENS
END
```

This request produces the following report:

```
EMP_ID      FIRST NAME    LAST_NAME     PAY_DATE      PAY_MONTH          GROSS
-------     ----------    ---------     --------      ---------          -----
071382660   ALFRED        STEVENS       81/11/30      NOVEMBER         $833.33
                                        81/12/31      DECEMBER         $833.33
                                        82/01/29      JANUARY          $916.67
                                        82/02/26      FEBRUARY         $916.67
                                        82/03/31      MARCH            $916.67
                                        82/04/30      APRIL            $916.67
                                        82/05/28      MAY              $916.67
                                        82/06/30      JUNE             $916.67
                                        82/07/30      JULY             $916.67
                                        82/08/31      AUGUST           $916.67
```

Subroutines Written in REXX

A FOCUS request can call user-written subroutines coded in REXX. These routines, also called FUSREXX macros, provide a 4GL option to the languages supported for user-written subroutines.

Using REXX Subroutines

REXX subroutines are supported in the VM/CMS and OS/390 environments:

- In VM/CMS, a FUSREXX macro can contain either REXX source code or compiled REXX code created by running the source code through the REXX compiler. In addition, you can load either type of FUSREXX macro into memory using the EXECLOAD command. The compilation and load process reduces the CPU requirements and increases speed. Compilation also is a security tool, making private information difficult to read.
- In OS/390, FOCUS supports source versions of REXX subroutines only.

Because of CPU requirements, the use of FUSREXX routines in large production jobs should be monitored carefully.

The following notes apply to the examples in this topic:

- REXX versions are not necessarily the same in all operating environments. Therefore, some of the examples may use REXX functions that are not available in your environment.
- The REXX code is listed, but not fully explained. See your REXX documentation for information about REXX instructions and functions.

Syntax How to Call a REXX User-Written Subroutine

In a DEFINE FILE command:

```
DEFINE FILE filename
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
END
```

In a DEFINE attribute in the Master File:

```
DEFINE fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
```

In a COMPUTE command:

```
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
```

In a Dialogue Manager -SET command:

```
-SET &var = subname(inlen1, inparm1, ..., outlen, outparm);
```

where:

fieldname
Is the name of the field to receive the return value.

An|In
Is the format of the field to receive return value.

subname
Is the name of the REXX routine.

inlen1, inparm1 ...
Are the input parameters. Each parameter consists of a pair of values: a length and an alphanumeric parameter value. You can supply the name of an alphanumeric field, an alphanumeric literal, or an expression that resolves to an alphanumeric value. Up to 13 input parameter pairs are supported by FOCUS. Each parameter value can be up to 256 bytes long.

Note: Dialogue Manager converts input parameters that consist of numeric digits to decimal format, regardless of their original data type. Therefore, you cannot pass numeric input parameters to a REXX routine using -SET.

outlen, outparm
Is the output parameter pair, consisting of a length and a return value. In most cases, the return value should be alphanumeric, but integer return values are also supported. The return value can be the name of the field or Dialogue Manager variable to which the value is returned or its USAGE format enclosed in single quotation marks. The return value can be a minimum of one byte long and a maximum (for an alphanumeric value) of 256 bytes.

Note: If the value returned is integer, *outlen* must be 4 because FOCUS reserves four bytes for integer fields.

&var
Is the name of the Dialogue Manager variable to receive the return value.

REXX subroutines:

- Require input data to be character and should return character output. Integer return values are also supported, but the output length in the subroutine call must be four. FOCUS has a 256-byte limit on character variables. This limit also applies to FUSREXX routines. FUSREXX routines return variable length data. For this reason, you must supply the length of the input arguments and the maximum length of the output data.
- Do *not* require any input parameters, but *do* require one return parameter, which *must* return at least one byte of data. It is possible for a FUSREXX function to need no input, such as a function that returns USERID.
- Do not support floating-point numbers (REXX does not have native floating-point conversion routines). All numeric fields should be converted to character format with no commas using a FOCUS function such as EDIT before being passed to the FUSREXX routine. This prevents FOCUS from converting numbers to floating point before passing them to the FUSREXX routine.
- Are not supported in Dialogue Manager -CMS RUN commands.
- On VM/CMS, the FILETYPE of REXX user-written functions is FUSREXX; they can be stored on any accessed disk.
- On OS/390, DDNAME FUSREXX must be allocated to a PDS, and that library will be searched before other OS/390 libraries.
- The search order for subroutines is:

  1. FUSREXX

  2. Standard VM/CMS or OS/390 search order.

Example Returning the Day of the Week

The FUSREXX routine DOW returns the day of the week an employee was hired. The routine passes one input parameter pair and one return field pair.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. DAY_OF_WEEK/A9 WITH AHDT= DOW(6,AHDT,9,DAY_OF_WEEK) ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAY_OF_WEEK
END
```

1. The input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.

2. The return field is up to nine bytes long and is named DAY_OF_WEEK.

The output is:

```
LAST_NAME        HIRE_DATE  DAY_OF_WEEK
---------        ---------  -----------
STEVENS           80/06/02  Monday
SMITH             81/07/01  Wednesday
JONES             82/05/01  Saturday
SMITH             82/01/04  Monday
BANNING           82/08/01  Sunday
IRVING            82/01/04  Monday
ROMANS            82/07/01  Thursday
MCCOY             81/07/01  Wednesday
BLACKWOOD         82/04/01  Thursday
MCKNIGHT          82/02/02  Tuesday
GREENSPAN         82/04/01  Thursday
CROSS             81/11/02  Monday
```

The FUSREXX macro is displayed below. The FUSREXX routine reads the input date, reformats it to MM/DD/YY format, and returns the day of the week using a REXX DATE call.

```
/* DOW routine. Return WEEKDAY from YYMMDD format date */
Arg ymd .
Return Date('W',Translate('34/56/12',ymd,'123456'),'U')
```

Example Returning Text Format

The REXX function called in this request returns the number of copies of each classic movie in text format. It passes one input parameter and one return field.

```
TABLE FILE MOVIES
PRINT TITLE AND COMPUTE
1. ACOPIES/A3 = EDIT(COPIES); AS 'COPIES'
AND COMPUTE
2. TXTCOPIES/A8 = NUMCNT(3,ACOPIES,8,TXTCOPIES);
WHERE CATEGORY EQ 'CLASSIC'
END
```

1. The input field is 3 bytes long. Data is passed in field ACOPIES. The COPIES field is converted to an alphanumeric field.
2. The return field is up to 8 bytes long and is named TXTCOPIES.

The output is:

```
TITLE                                    COPIES  TXTCOPIES
-----                                    ------  ---------
EAST OF EDEN                             001     One
CITIZEN KANE                             003     Three
CYRANO DE BERGERAC                       001     One
MARTY                                    001     One
MALTESE FALCON, THE                      002     Two
GONE WITH THE WIND                       003     Three
ON THE WATERFRONT                        002     Two
MUTINY ON THE BOUNTY                     002     Two
PHILADELPHIA STORY, THE                  002     Two
CAT ON A HOT TIN ROOF                    002     Two
CASABLANCA                               002     Two
```

The FUSREXX macro is:

```
/* NUMCNT routine. Pass a number from 0 to 10 and return a character value
*/
Arg numbr .
data = 'Zero One Two Three Four Five Six Seven Eight Nine Ten'
numbr = numbr + 1           /* so 0 equals 1 element in array */
Return Word(data,numbr)
```

Example Passing Multiple Arguments

The following example shows how to pass multiple arguments to a FUSREXX routine. It is an interest calculation using the present salary for the employee and the employee start date to calculate a present value. It passes four input parameters and one return field.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6     = EDIT(HIRE_DATE) ;
2. ACSAL/A12   = EDIT(CURR_SAL) ;
3. DCSAL/D12.2 = CURR_SAL ;
4. PV/A12      = INTEREST(6,AHDT,6,'&YMD',3,'6.5',12,ACSAL,12,PV) ;
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE DCSAL PV
END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The current salary is converted to an alphanumeric field for use in the interest calculation.
3. The current salary is converted to a double-precision field to include commas and a decimal point in the output.
4. The second input field is six bytes long. Data is passed as a FOCUS character variable &YMD in YYMMDD format.

   The third input field is a character value of 6.5, which is 3 bytes long to account for the decimal point in the character string.

   The fourth input field is 12 bytes long. This passes the character field ACSAL.

   The return field is up to 12 bytes long and is named PV.

The output is:

```
LAST_NAME       FIRST_NAME  HIRE_DATE          DCSAL           PV
---------       ----------  ---------          -----           --
STEVENS         ALFRED       80/06/02      11,000.00     14055.14
SMITH           MARY         81/07/01      13,200.00     15939.99
JONES           DIANE        82/05/01      18,480.00     21315.54
SMITH           RICHARD      82/01/04       9,500.00     11155.60
BANNING         JOHN         82/08/01      29,700.00     33770.53
IRVING          JOAN         82/01/04      26,862.00     31543.35
ROMANS          ANTHONY      82/07/01      21,120.00     24131.19
MCCOY           JOHN         81/07/01      18,480.00     22315.99
BLACKWOOD       ROSEMARIE    82/04/01      21,780.00     25238.25
MCKNIGHT        ROGER        82/02/02      16,100.00     18822.66
GREENSPAN       MARY         82/04/01       9,000.00     10429.03
CROSS           BARBARA      81/11/02      27,062.00     32081.82
```

The FUSREXX macro is displayed below. The REXX format command is used to format the return value.

```
/* Simple INTEREST program. dates are yymmdd format  */
Arg start_date,now_date,percent,open_balance, .

begin = Date('B',Translate('34/56/12',start_date,'123456'),'U')
stop  = Date('B',Translate('34/56/12',now_date,'123456'),'U')
valnow = open_balance * (((stop - begin) * (percent / 100)) / 365)

Return Format(valnow,9,2)
```

Example Accepting Multiple Tokens in Parameters

FUSREXX routines can accept multiple tokens in a parameter. The following procedure passes employee information (pay date and monthly gross pay) as separate tokens in the first parameter. It passes three input parameters and one return field.

```
   DEFINE FILE EMPLOYEE
1. COMPID/A256 = FN | ' ' | LN | ' ' |  DPT | ' ' | EID ;
2. APD/A6 = EDIT(PAY_DATE) ;
3. APAY/A12 = EDIT(MO_PAY) ;
4. OK4RAISE/A1 = OK4RAISE(256,COMPID,6,APD,12,APAY,1,OK4RAISE) ;
   END

   TABLE FILE EMPLOYEE
   PRINT EMP_ID FIRST_NAME LAST_NAME DEPARTMENT
   IF OK4RAISE EQ '1'
   END
```

1. The first input field is 256 bytes long. Data is passed in field COMPID. COMPID is the concatenation of several character fields passed as the first parameter. Each of the other parameters is a single argument.
2. The second input field is six bytes long. Data is passed in field APD. The pay date is converted to an alphanumeric field.
3. The third input field is 12 bytes long. Data is passed in field APAY. The monthly gross pay is converted to an alphanumeric field.
4. The return field is up to one byte long and is named OK4RAISE.

The output is:

```
EMP_ID     FIRST_NAME         LAST_NAME      DEPARTMENT
------     ----------         ---------      ----------
071382660  ALFRED             STEVENS        PRODUCTION
```

The FUSREXX macro is displayed below. Commas separate FUSREXX parameters. The ARG command specifies multiple variable names before the first comma and, therefore, separates the first FUSREXX parameter into separate REXX variables, using blanks as delimiters between the variables.

```
/* OK4RAISE routine. Parse separate tokens in the 1st parm, then more parms
*/

Arg fname lname dept empid,  pay_date,  gross_pay, .

If dept = 'PRODUCTION' & pay_date < '820000'
Then retvalue = '1'
Else retvalue = '0'

Return retvalue
```

FUSREXX routines *should* use the REXX RETURN function to return data to FOCUS. REXX EXIT is acceptable, but is generally used to end an EXEC, not a FUNCTION.

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

Example Returning an Integer Value

It is possible for REXX to return a value that is *not* character format. The following example shows how REXX returns an integer value. This example also shows how the format of the integer field is used as the last field in the return argument. It passes two input fields and one return field. The FUSREXX routine NUMDAYS returns the number of days between hire date and date of increase. Note that the return value for an integer is *always* four bytes long.

```
     DEFINE FILE EMPLOYEE
1.   AHDT/A6 = EDIT(HIRE_DATE) ;
2.   ADI/A6 = EDIT(DAT_INC) ;
3.   BETWEEN/I6 = NUMDAYS(6,AHDT,6,ADI,4,'I6') ;
     END

     TABLE FILE EMPLOYEE
     PRINT LAST_NAME HIRE_DATE DAT_INC BETWEEN
     IF BETWEEN NE 0
     END
```

1. The first input field is six bytes long. Data is passed in field AHDT. The hire date is converted to an alphanumeric field.
2. The second input field is six bytes long. Data is passed in field ADI. The date of increase is converted to an alphanumeric field.
3. The return field is up to six bytes long and is named BETWEEN.

The output is:

```
LAST_NAME      HIRE_DATE   DAT_INC  BETWEEN
---------      ---------   -------  -------

STEVENS        80/06/02    82/01/01     578
STEVENS        80/06/02    81/01/01     213
SMITH          81/07/01    82/01/01     184
JONES          82/05/01    82/06/01      31
SMITH          82/01/04    82/05/14     130
IRVING         82/01/04    82/05/14     130
MCCOY          81/07/01    82/01/01     184
MCKNIGHT       82/02/02    82/05/14     101
GREENSPAN      82/04/01    82/06/11      71
CROSS          81/11/02    82/04/09     158
```

The FUSREXX macro is displayed below. The return value is converted from REXX character to HEX and formatted to be four bytes long.

```
/* NUMDAYS routine. Return number of days between 2 dates in yymmdd format
*/
/* The value returned will be in hex format
    */

Arg first,second .

base1 = Date('B',Translate('34/56/12',first,'123456'),'U')
base2 = Date('B',Translate('34/56/12',second,'123456'),'U')

Return D2C(base2 - base1,4)
```

Example Returning a Date Field From a FUSREXX Macro

FOCUS smart date fields contain the integer number of days since the base date 12/31/1900. REXX has a date function that can accept and return several types of date formats, including one called Base format (‘B’) that contains the number of days since the REXX base date 01/01/0001 (Jan. 1 of the Year 1).

Because input arguments must be alphanumeric, you cannot pass a smart date field to a REXX subroutine. Therefore, you can either:

- Pass the REXX routine an alphanumeric field with date display options and have it return a smart date value, if you account for the number of days difference between the FOCUS base date and the REXX base date and convert the result to integer.
- Pass the REXX routine a smart date value converted to alphanumeric format. With this technique, you must account for the difference in base dates for both the input and output.

The following example uses the technique of passing the subroutine an alphanumeric field with date display options. The FUSREXX macro called DATEREX1 takes two input arguments: an alphanumeric date in A8YYMD format and a number of days in character format. It returns a smart date in YYMD format that represents the input date plus the number of days. The FOCUS format A8YYMD corresponds to the REXX Standard format ('S').

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX1 routine. Add indate (format A8YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate,'S')+ days - 693959, 4)
```

The following request uses the DATEREX1 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to A8YYMD before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD =  HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8YYMD= ADATE; NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX1(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The output is:

```
LAST_NAME       FIRST_NAME  HIRE_DATE  NEXT_DATE
---------       ----------  ---------  ---------
BANNING         JOHN         82/08/01  1983/08/01
BLACKWOOD       ROSEMARIE    82/04/01  1983/04/01
CROSS           BARBARA      81/11/02  1982/11/02
GREENSPAN       MARY         82/04/01  1983/04/01
IRVING          JOAN         82/01/04  1983/01/04
JONES           DIANE        82/05/01  1983/05/01
MCCOY           JOHN         81/07/01  1982/07/01
MCKNIGHT        ROGER        82/02/02  1983/02/02
ROMANS          ANTHONY      82/07/01  1983/07/01
SMITH           MARY         81/07/01  1982/07/01
SMITH           RICHARD      82/01/04  1983/01/04
STEVENS         ALFRED       80/06/02  1981/06/02
```

The following example uses the technique of passing the subroutine a smart date converted to alphanumeric format. The FUSREXX macro called DATEREX2 takes two input arguments: an alphanumeric number of days that represents a smart date, and a number of days to add. It returns a smart date in YYMD format that represents the input date plus the number of days. Both the input date and output date are in REXX base date ('B') format.

The number 693959 represents the number of days difference between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX2 routine. Add indate (original format YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate+693959,'B') + days - 693959, 4)
```

The following request uses the DATEREX2 macro to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to an alphanumeric number of days before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD =  HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8 = EDIT(ADATE); NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX2(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The report output is the same as that produced by the DATEREX1 macro.

Compiling FUSREXX Macros in VM/CMS

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

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

To compile and compress this FUSREXX macro in VM/CMS, issue the following command. Note that the file identifier must be in upper case:

```
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 FOCUS request, you must rename it to have the file type FUSREXX. The file SUM2 LISTING details the results of the compilation.

To use the compiled version in a FOCUS request, issue the following commands. The EXECLOAD command, which loads the routine into memory and improves performance, is optional:

```
rename sum2 fusrexx a ssum2 fusrexx a
rename sum2 cfusrexx a sum2 fusrexx a
execload sum2 fusrexx a
```

Then, in FOCUS, issue the following request:

```
TABLE FILE EMPLOYEE
PRINT CSAL AND COMPUTE
ASAL/A12 = EDIT(CSAL);
AMOUNT/A4 = '1000';
TOTSAL/I6 = SUM2(12, ASAL, 4, AMOUNT, 4, TOTSAL);
END
```

The output is:

```
CURR_SAL  ASAL          AMOUNT  TOTSAL
--------  ----          ------  ------
$11,000.00  000000011000  1000     12000
$13,200.00  000000013200  1000     14200
$18,480.00  000000018480  1000     19480
$9,500.00  000000009500  1000     10500
$29,700.00  000000029700  1000     30700
$26,862.00  000000026862  1000     27862
$21,120.00  000000021120  1000     22120
$18,480.00  000000018480  1000     19480
$21,780.00  000000021780  1000     22780
$16,100.00  000000016100  1000     17100
$9,000.00  000000009000  1000     10000
$27,062.00  000000027062  1000     28062
```

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Y

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, Two Penn Plaza, New York, NY 10121-2898 **(212) 736-4433**