ibi™ FOCUS®

Using Functions

Version 9.3.2 | November 2024

Contents

How to Use This Manual

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

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

Available Languages

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

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

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

Operating Systems

Except in cases noted specifically, all functions run on all FOCUS-supported operating systems.

Introducing Functions

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

Using Functions

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

There are three types of functions:

- **Internal functions.** Built into the FOCUS language, requiring no extra work to access or use. The following functions are internal functions. You cannot replace any of these internal functions with your own functions of the same name. All other functions are external.
    - ABS
    - ASIS
    - DMY, MDY, and YMD
    - DECODE
    - EDIT
    - FIND
    - LAST
    - LOG
    - LOOKUP
    - MAX and MIN
    - SQRT
- **External functions.** Stored in an external library that must be accessed. When

invoking these functions, an argument specifying the output field or format of the result is required. External functions are distributed with FOCUS. You can replace these functions with your own functions of the same name. However, in this case, you must set USERFNS=LOCAL.

- **Subroutines.** Written by the user and stored externally. For details, see Creating a Subroutine.

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

Types of Functions

You can access any of the following types of functions:

- **Simplified analytic functions.** Perform calculations using multiple rows in the internal matrix. For details, see Types of Functions.
- **Simplified character functions.** Character functions with streamlined parameter lists and no output arguments, similar to those used by SQL functions. For details, see Types of Functions.
- **Character functions.** Manipulate alphanumeric fields or character strings. For details, see Types of Functions.
- **Variable length character functions.** Manipulate AnV fields or character strings. For details, see Types of Functions.
- **Character functions for DBCS code pages.** Manipulate alphanumeric fields or character strings on DBCS code pages. For details, see Types of Functions.
- **Data source and decoding functions.** Search for or retrieve data source records or values, and assign values. For details, see Types of Functions.
- **Simplified date and date-time functions.** Date and date-time functions with streamlined parameter lists and no output arguments, similar to those used by SQL functions. For details, see Types of Functions.
- **Date functions.** Manipulate dates. For details, see Types of Functions.
- **Date-time functions.** Manipulate date-time values. For details, see Types of Functions.
- **Simplified conversion functions.** Convert fields from one format to another using

streamlined parameter lists. For details, see Types of Functions.

- **Format conversion functions.** Convert fields from one format to another. For details, see Types of Functions.
- **Simplified numeric functions.** Perform calculations on numeric constants and fields using streamlined parameter lists. For details, see Types of Functions.
- **Numeric functions.** Perform calculations on numeric constants and fields. For details, see Types of Functions.
- **Simplified statistical functions.** Perform statistical calculations. For details, see Types of Functions.
- **Simplified system functions.** Call the operating system to obtain information about the operating environment or to use a system service, using streamlined parameter lists. For details, see Types of Functions.
- **System functions.** Call the operating system to obtain information about the operating environment or to use a system service. For details, see Types of Functions.
- **Trigonometric functions.** Perform trigonometric calculations, inverse trigonometric calculations, and angle conversion functions For details, see Trigonometric Functions.

WebFOCUS-Specific Functions

Most supplied functions are available in both WebFOCUS and FOCUS. However, some functions are available only in WebFOCUS. They are:

- SPAWN
- SYSTEM

For details on these functions, see the individual topics.

Simplified Analytic Functions

The following functions perform calculations based on multiple rows in the internal matrix. For details, see Simplified Analytic Functions.

FORECAST_MOVAVE

Calculates a simple moving average column.

FORECAST_EXPAVE

Calculates a single exponential smoothing column.

FORECAST_DOUBLEXP

Calculates a double exponential smoothing column.

FORECAST_SEASONAL

Calculates a triple exponential smoothing column.

FORECAST_LINEAR

Calculates a linear regression column.

PARTITION_AGGR

Creates rolling calculations.

PARTITION_REF

Retrieves prior or subsequent fields.

INCREASE

Calculates the difference between a value in the current row and a prior row within a partition.

PCT_INCREASE

Calculates the percent difference between a value in the current row and a prior row within a partition.

PREVIOUS

Retrieves a prior value within a partition.

RUNNING_AVE

Calculate the average over a group of rows within a partition.

RUNNING_MIN

Calculate the minimum over a group of rows within a partition.

RUNNING_MAX

Calculate the maximum over a group of rows within a partition.

RUNNING_SUM

Calculate the sum over a group of rows within a partition.

Simplified Character Functions

The following functions manipulate alphanumeric fields or character strings and have simplified parameter lists. For details, see Simplified Character Functions.

CHAR_LENGTH

Returns the length, in characters, of a string.

DIGITS

Converts a number to a character string of the specified length.

GET_TOKEN

Extracts a token (substring) based on a token number and a string listing acceptable delimiter characters.

INITCAP

Capitalizes the first letter of every word in a string and makes all other letters lowercase, where a word starts at the beginning of the string, after a blank space, or after a special character.

LAST_NONBLANK

retrieves the last field value that is neither blank nor missing. If all previous values are either blank or missing, returns a missing value.

LOWER

Translates a string to lowercase.

LPAD

Left-pads a string with a given character.

LTRIM

Removes all blanks from the left end of a string.

PATTERNS

Returns a pattern that represents the structure of the source string.

POSITION

Returns the first position (in characters) of a substring in a source string.

REGEX

Matches a string to a regular expression and returns true (1) or false (0).

RPAD

Right-pads a string with a given character.

RTRIM

Removes all blanks from the right end of a string.

SUBSTRING

Extracts a substring from a source string.

TOKEN

Extracts a token (substring) based on a token number and a delimiter string.

TRIM_

Removes all occurrences of a single character from either the beginning or end of a string, or both.

UPPER

Translates a string to uppercase.

Character Functions

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

ARGLEN

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

ASIS

Distinguishes between a blank and a zero in Dialogue Manager.

BITSON

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

BITVAL

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

BYTVAL

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

CHKFMT

Checks a character string for incorrect characters or character types.

CTRAN

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

CTRFLD

Centers a character string within a field.

EDIT

Extracts characters from or adds characters to a character string.

GETTOK

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

LCWORD

Converts the letters in a character string to mixed case.

LCWORD2

Converts the letters in a character string to mixed case.

LCWORD3

Converts the letters in a character string to mixed case.

LJUST

Left-justifies a character string within a field.

LOCASE

Converts alphanumeric text to lowercase.

OVRLAY

Overlays a base character string with a substring.

PARAG

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

POSIT

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

REVERSE

Reverses the characters in a character string.

RJUST

Right-justifies a character string.

SOUNDEX

Searches for a character string phonetically without regard to spelling.

SPELLNM

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

SQUEEZ

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

STRIP

Removes all occurrences of a specific character from a string.

STRREP

Replaces all occurrences of a specific character string.

SUBSTR

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

TRIM

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

UPCASE

Converts a character string to uppercase.

Variable Length Character Functions

The following functions manipulate variable length alphanumeric fields or character strings. For details, see Variable Length Character Functions.

LENV

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

LOCASV

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

POSITV

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

SUBSTV

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

TRIMV

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

UPCASV

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

Character Functions for DBCS Code Pages

The following functions manipulate character strings for DBCS code pages. For details, see Character Functions for DBCS Code Pages.

DCTRAN

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

DEDIT

Extracts characters from or adds characters to a string.

DSTRIP

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

DSUBSTR

Extracts a substring based on its length and position in the source string.

JPTRANS

Converts Japanese specific characters.

Maintain-specific Character Functions

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

CHAR2INT

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

INT2CHAR

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

LCWORD and LCWORD2

Converts the letters in a character string to mixed case.

LENGTH

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

LJUST

Left-justifies a character string within a field.

LOWER

Converts a character string to lowercase.

MASK

Extracts characters from or adds characters to a character string.

MNTGETTOK

Divides a character string into substrings, called tokens.

NLSCHR

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

OVRLAY

Overlays a base character string with a substring.

POSIT

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

RJUST

Right-justifies a character string.

SELECTS

Decodes a value from a stack.

STRAN

Substitutes a substring for another substring in a character string.

STRCMP

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

STRICMP

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

STRNCMP

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

SUBSTR

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

TRIM

Removes trailing occurrences of a pattern within a character string.

TRIMLEN

Determines the length of a character string excluding trailing spaces.

UPCASE

Converts a character string to uppercase.

Data Source and Decoding Functions

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

COALESCE

Returns the value of the first non-missing argument.

DB_EXPR

Inserts an SQL expression into the SQL generated for a request against a relational data source.

DB_INFILE

Compares values in a source file to values in a target file, or if the source file is a relational data source, to values retrieved by a subquery.

CHECKMD5

Computes an MD5 hash check value of its input parameter.

CHECKSUM

Computes hash sum of its input parameter.

DB_LOOKUP

Retrieves a data value from a lookup data source.

DECODE

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

FIND

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

IMPUTE

Replaces missing values with aggregated values.

LAST

Retrieves the preceding value for a field.

LOOKUP

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

Available Languages: MODIFY

NULLIF

Returns a missing value when its parameters have equal values.

FOCUSPAGES

Returns the size of a FOCUS or XFOCUS file as number of pages.

Simplified Date and Date-Time Functions

The following functions manipulate date and date- time values. For details see Simplified Date and Date-Time Functions.

DT_CURRENT_DATE

Returns the current date.

DT_CURRENT_DATETIME

Returns the current date and time.

DT_CURRENT_TIME

Returns the current time.

DTADD

Returns a new date after adding the specified number of a supported component

DTDIFF

Returns the number of given component boundaries between the two dates.

DTIME

Extracts time components from a date-time value.

DTPART

Returns a component value in integer format.

DTRUNC

Returns the first date within a period

Date Functions

The following functions manipulate dates. For details see Date Functions.

Standard Date Functions

DATEADD

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

DATECVT

Converts date formats.

DATEDIF

Returns the difference between two dates in units.

DATEMOV

Moves a date to a significant point on the calendar.

DATETRAN

Formats dates in international formats.

DPART

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

FIYR

Returns the financial year, also known as the fiscal year, corresponding to a given calendar date based on the financial year starting date and the financial year numbering convention.

FIQTR

Returns the financial quarter corresponding to a given calendar date based on the financial year starting date and the financial year numbering convention.

FIYYQ

Returns a financial date containing both the financial year and quarter that corresponds to a given calendar date.

HMASK

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

TODAY

Retrieves the current date from the system.

Legacy Date Functions

AYM

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

AYMD

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

CHGDAT

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

DA

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

DADMY converts dates in day-month-year format.

DADYM converts dates in day-year-month format.

DAMDY converts dates in month-day-year format.

DAMYD converts dates in month-year-day format.

DAYDM converts dates in year-day-month format.

DAYMD converts dates in year-month-day format.

DMY, MDY, and YMD

Calculate the difference between two dates.

DOWK and DOWKL

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

DT

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

DTDMY converts numbers to day-month-year dates.

DTDYM converts numbers to day-year-month dates.

DTMDY converts numbers to month-day-year dates.

DTMYD converts numbers to month-year-day dates.

DTYDM converts numbers to year-day-month dates.

DTYMD converts numbers to year-month-day dates.

GREGDT

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

JULDAT

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

YM

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

Date-Time Functions

The following functions manipulate date-time values. For details see Date-Time Functions.

HADD

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

HCNVRT

Converts a date-time field to a character string.

HDATE

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

HDIFF

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

HDTTM

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

HEXTR

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

HGETC

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

HMASK

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

HHMMSS

Retrieves the current time from the system.

HINPUT

Converts an alphanumeric string to a date-time value.

HMIDNT

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

HNAME

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

HPART

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

HSETPT

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

HTIME

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

HTMTOTS/TIMETOTS

Converts a time to a timestamp.

Accepts a date and time in one of five formats and converts the value to native OpenVMS 64-bit DEC Date/Time format.

Maintain-specific Date and Time Functions

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

Maintain-specific Standard Date and Time Functions

HHMMSS

Retrieves the current time from the system.

Initial_HHMMSS

Retrieves the time that the Maintain module was started.

Initial_TODAY

Retrieves the date that the Maintain module was started.

TODAY

Retrieves the current date from the system.

TODAY2

Retrieves the current date from the system.

Maintain-specific Legacy Date Functions

ADD

Adds a given number of days to a date.

DAY

Extracts the day of the month from a date.

JULIAN

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

MONTH

Extracts the month from a date.

QUARTER

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

SETMDY

Sets a value to a date.

SUB

Subtracts a given number of days from a date.

WEEKDAY

Determines the day of the week for a date.

YEAR

Extracts the year from a date.

Simplified Conversion Functions

The following functions convert fields from one format to another, using streamlined parameter lists. For details, see Simplified Conversion Functions.

CHAR

Returns a character based on a numeric code.

COMPACTFORMAT

Converts a numeric value to an alphanumeric value that represents the number in an abbreviated format, using the characters K, M, B, and T to represent the abbreviation.

CTRLCHAR

Returns a non-printable control character.

DT_FORMAT

Converts a date or date-time value to an alphanumeric string.

FPRINT

Converts a numeric, date, or date-time value to a character string.

HEXTYPE

Returns the hexadecimal view of an input value.

PHONETIC

Returns a phonetic key.

Format Conversion Functions

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

ATODBL

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

EDIT

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

FPRINT

Converts a field to alphanumeric format.

FTOA

Converts a number in a numeric format to alphanumeric format.

HEXBYT

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

ITONUM

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

ITOPACK

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

ITOZ

Converts a number in numeric format to zoned format.

PCKOUT

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

PTOA

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

TSTOPACK

Converts a Microsoft SQL Server or Sybase TIMESTAMP column (which contains an incremented counter) to packed decimal.

UFMT

Converts characters in alphanumeric field values to hexadecimal representation.

XTPACK

Stores a packed number with up to 31 significant digits in an alphanumeric field, retaining decimal data.

Maintain-specific Light Update Support Functions

The following functions retrieve WebFOCUS variable or parameter data implicitly from within a Maintain procedure. These functions are available only in Maintain Data. For details, see Maintain-specific Light Update Support Functions.

IWC.GetAppCGIValue

Imports the value of a WebFOCUS parameter or variable into a Maintain Data variable.

IWC.FindAppCGIValue

Retrieves WebFOCUS parameter or variable values.

Simplified Numeric Functions

The following functions perform calculations on numeric constants or fields, using streamlined parameter lists. For details, see Simplified Numeric Functions.

CEILING

Returns the smallest integer value greater than or equal to a value.

EXPONENT

Raises *e* to a power.

FLOOR

Returns the largest integer value less than or equal to a value.

MOD

Calculates the remainder from a division.

POWER

Raises a value to a power.

Numeric Functions

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

ABS

Returns the absolute value of a number.

ASIS

Distinguishes between a blank and a zero in Dialogue Manager.

BAR

Produces a horizontal bar chart.

CHKPCK

Validates the data in a field described as packed format.

DMOD, FMOD, and IMOD

Calculate the remainder from a division.

EXP

Raises the number "e" to a specified power.

EXPN

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

FMLINFO

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

FMLLIST

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

FMLFOR

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

FMLCAP

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

INT

Returns the integer component of a number.

LOG

Returns the natural logarithm of a number.

MAX and MIN

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

MIRR

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

NORMSDST and NORMSINV

Perform calculations on a standard normal distribution curve.

PRDNOR and PRDUNI

Generate reproducible random numbers.

RDNORM and RDUNIF

Generate random numbers.

SQRT

Calculates the square root of a number.

XIRR

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

Maintain-specific Script Functions

Script functions integrate JavaScript and VBScripts into your Maintain Data applications and perform client-side execution without returning to the WebFOCUS Reporting Server. These functions are available only in Maintain Data. For details, see Maintain-specific Script Functions.

IWCLink

Executes external procedures.

IWCSwitchToSecure and IWCSwitchToUnsecure

Turns the Secure Sockets layer on and off, respectively.

IWCTrigger

Returns control from the script to your application.

Simplified Statistical Functions

The following functions perform statistical functions. For details, see Simplified Statistical Functions.

CORRELATION

Calculates the degree of correlation between two independent sets of data.

KMEANS_CLUSTER

Partitions observations into clusters based on the nearest mean value.

MULTIREGRESS

Calculates a linear regression column based on multiple fields.

OUTLIER

Identifies outliers in a numeric field using the 1.5 * IQR rule.

STDDEV

Calculates the standard deviation in a set of data values.

Machine Learning (Python-based) Functions

The following functions run Python scripts to perform classification and regression. For details, see Machine Learning (Python-based) Functions.

ANOMALY_IF

Detects outliers using an Isolation Forest.

CLASSIFY_BLR

Finds the best linear separation between two classes in the space spanned by the predictors, and returns either a class assignment (0 or 1) or the probability of belonging to class 1.

CLASSIFY_KNN

Assigns a class membership to a data point by assigning the class most common among its *k* nearest neighbors.

CLASSIFY_RF

Creates a random forest (ensemble of decision trees), and returns the prediction that is the average of the individual predictions.

CLASSIFY_XGB

Creates a random forest (ensemble of decision trees), where each new tree attempts to improve on the predictive capabilities of the prior trees.

REGRESS_KNN

Predicts a target value for a data point by assigning the average of the target values of its *k* nearest neighbors.

REGRESS_POLY

Fits the target column to a polynomial expression of the predictor columns.

REGRESS_RF

Creates a random forest (ensemble of decision trees), and returns the prediction that is the majority vote of the individual classification predictions.

REGRESS_XGB

Creates a random forest (ensemble of decision trees), where each new tree attempts to improve on the predictive capabilities of the prior trees.

RUN_MODEL and RUN_MODEL2

RUN_MODEL runs a saved model where the field names in the data source used to run the model are the same as the field names in the data source used to create the model. RUN_MODEL2 runs a saved model where the field names in the data source used to run the model are not the same as the field names in the data source used to create the model.

Simplified System Functions

The following functions call the operating system to obtain information about the operating environment or to use a system service, using streamlined parameter lists. For details, see Simplified System Functions.

EDAPRINT

Inserts a custom message in the EDAPRINT log file.

ENCRYPT

Encrypts a password.

GETENV

Retrieves the value of an environment variable.

PUTENV

Assigns a value to an environment variable.

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 System Functions.

CLSDDREC

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

FEXERR

Retrieves a FOCUS error message.

FINDMEM

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

Available Operating Systems: z/OS

GETPDS

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

Available Operating Systems: z/OS

GETUSER

Retrieves the ID of the connected user.

MVSDYNAM

Transfers a FOCUS DYNAM command to the DYNAM command processor.

Available Operating Systems: z/OS

PUTDDREC

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

SLEEP

Suspends execution for a specified number of seconds.

SYSVAR

Retrieves the value of a z/OS system variable.

Available Operating Systems: z/OS

Available Languages: reporting

Simplified Geography Functions

These functions perform location-based calculations and retrieve geocoded points for various types of location data. For details, see Simplified Geography Functions.

GIS_DISTANCE

Calculates the distance between geography points.

GIS_DRIVE_ROUTE

Calculates the driving directions between geography points.

GIS_POINT

Builds a geometry point.

GIS_GEOCODE_ADDR

Geocodes a complete address.

GIS_GEOCODE_ADDR_CITY

Geocodes an address line, city, and state.

GIS_GEOCODE_ADDR_POSTAL

Geocodes an address line and postal code.

GIS_GEOMETRY

Builds a JSON geometry object.

GIS_IN_POLYGON

Determines whether a point is in a complex polygon.

GIS_LINE

Builds a JSON line.

GIS_REVERSE_COORDINATE

Given a longitude, latitude, and component name, returns the applicable geographic component.

GIS_SERVICE_AREA

Calculates a geometry area around a given point.

GIS_SERV_AREA_XY

Calculates a geometry area around a given coordinate.

Trigonometric Functions

The trigonometric functions provide trigonometric calculations, inverse trigonometric calculations, and angle conversion functions. For details, see Trigonometric Functions.

ACOS

Given the cosine of an angle in radians, ACOS (arccosine) returns an angle between 0 (zero) and pi radians.

ASIN

Given the sine of an angle in radians, ASIN (arcsine) returns an angle between -(pi/2) and pi/2 radians.

ATAN

Given the tangent of an angle in radians, ATAN (arctangent) returns an angle between -(pi/2) and pi/2 radians.

ATAN2

Given the coordinates of the tangent of an angle in radians, ATAN2 (arctangent2) returns an angle between -pi and pi radians.

COS

Given an angle in radians, COS calculates the cosine of the angle.

COT

Given an angle in radians, COT calculates the cotangent of the angle.

DEGREES

Converts an angle in radians to an angle in degrees.

PI

Returns the constant pi as a floating-point number.

RADIANS

Converts an angle in degrees to an angle in radians.

SIN

Given an angle in radians, SIN calculates the sine of the angle.

TAN

Given an angle in radians, TAN calculates the tangent of the angle.

Character Chart for ASCII and EBCDIC

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

Decimal	ASCII		EBCDIC	
33	!	exclamation point		
34	"	quotation mark		
35	#	number sign		
36	$	dollar sign		
37	%	percent		
38	&	ampersand		

Decimal	ASCII		EBCDIC	
39	'	apostrophe		
40	(	left parenthesis		
41	)	right parenthesis		
42	*	asterisk		
43	+	plus sign		
44	,	comma		
45	-	hyphen		
46	.	period		
47	/	slash		
48	0	0		
49	1	1		
50	2	2		
51	3	3		
52	4	4		
53	5	5		
54	6	6		
55	7	7		
56	8	8		
57	9	9		

Decimal	ASCII		EBCDIC	
58	:	colon		
59	;	semicolon		
60	<	less-than sign		
61	=	equal sign		
62	>	greater-than sign		
63	?	question mark		
64	@	at sign		
65	A	A		
66	B	B		
67	C	C		
68	D	D		
69	E	E		
70	F	F		
71	G	G		
72	H	H		
73	I	I		
74	J	J	¢	cent sign
75	K	K	.	period
76	L	L	<	less-than sign

Decimal	ASCII		EBCDIC	
77	M	M	(	left parenthesis
78	N	N	+	plus sign
79	O	O	|	logical or
80	P	P	&	ampersand
81	Q	Q		
82	R	R		
83	S	S		
84	T	T		
85	U	U		
86	V	V		
87	W	W		
88	X	X		
89	Y	Y		
90	Z	Z	!	exclamation point
91	[	opening bracket	$	dollar sign
92	\	back slant	*	asterisk
93	]	closing bracket	)	right parenthesis
94	^	caret	;	semicolon
95	_	underscore	¬	logical not

Decimal	ASCII		EBCDIC	
96	`	grave accent	-	hyphen
97	a	a	/	slash
98	b	b		
99	c	c		
100	d	d		
101	e	e		
102	f	f		
103	g	g		
104	h	h		
105	i	i		
106	j	j		
107	k	k	,	comma
108	l	l	%	percent
109	m	m	_	underscore
110	n	n	>	greater-than sign
111	o	o	?	question mark
112	p	p		
113	q	q		
114	r	r		

Decimal	ASCII		EBCDIC	
115	s	s		
116	t	t		
117	u	u		
118	v	v		
119	w	w		
120	x	x		
121	y	y		
122	z	z	:	colon
123	{	opening brace	#	number sign
124	|	vertical line	@	at sign
125	}	closing brace	'	apostrophe
126	~	tilde	=	equal sign
127			"	quotation mark
129			a	a
130			b	b
131			c	c
132			d	d
133			e	e
134			f	f

Decimal	ASCII		EBCDIC	
135			g	g
136			h	h
137			i	i
145			j	j
146			k	k
147			l	l
148			m	m
149			n	n
150			o	o
151			p	p
152			q	q
153			r	r
162			s	s
163			t	t
164			u	u
165			v	v
166			w	w
167			x	x
168			y	y

Decimal	ASCII		EBCDIC	
169			z	z
185			`	grave accent
193			A	A
194			B	B
195			C	C
196			D	D
197			E	E
198			F	F
199			G	G
200			H	H
201			I	I
209			J	J
210			K	K
211			L	L
212			M	M
213			N	N
214			O	O
215			P	P
216			Q	Q

Decimal	ASCII		EBCDIC	
217			R	R
226			S	S
227			T	T
228			U	U
229			V	V
230			W	W
231			X	X
232			Y	Y
233			Z	Z
240			0	0
241			1	1
242			2	2
243			3	3
244			4	4
245			5	5
246			6	6
247			7	7
248			8	8
249			9	9

Accessing and Calling a Function

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

Note: FOCUSis fully LE compliant, and all FOCUS applications must be LE compliant.

Calling a Function

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

For more information on external functions, see Types of Functions.

Call a Function

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

where:

function

Is the name of the function.

arg1, arg2, ...

Are the arguments.

outfield

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

In Dialogue Manager, you must specify the format.

Store Output in a Field

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

or

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

or

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

where:

DEFINE

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

COMPUTE

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

field

Is the field that contains the result.

file

Is the file in which the virtual field is created.

var

Is the variable that contains the result.

fmt

Is the format of the field that contains the result.

function

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

input1, input2,...

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

outfield

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

In Dialogue Manager, you must specify the format.

Access the Maintain MNTUWS Function Library

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

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

Supplying an Argument in a Function

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

Argument Types

The following are acceptable arguments for a function:

- Numeric constant, such as 6 or 15.

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

  ```
  CURR_SAL * 1.03
  ```

  and

  ```
  FN || LN
  ```

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

Argument Formats

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

- **Alphanumeric argument.** An alphanumeric argument is stored internally as one character per byte. An alphanumeric argument can be a literal, an alphanumeric field, a number or date stored in alphanumeric format, an alphanumeric expression, or the format of an alphanumeric field. A literal is enclosed in single quotation marks, except when specified in operating systems that support Dialogue Manager RUN commands (for example, -MVS RUN).

- **Numeric argument.** A numeric argument is stored internally as a binary or packed number. A numeric argument includes integer (I), floating-point single-precision (F), floating-point double-precision (D), and packed decimal (P) formats. A numeric argument can be a numeric constant, field, or expression, or the format of a numeric field.

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

  > **Note:** With CDN ON, numeric arguments must be delimited by a comma followed by a space.

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

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

Argument Length

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

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

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

- If the specified length is shorter than the actual length, a subset of the string is used. For example, passing the argument 'ABCDEF' and specifying a length of 3 causes the function to process a string of 'ABC'.
- If the specified length is too long, whatever is in memory up to that length is

included. For example, passing an argument of 'ABC' and specifying a length of 6 causes the function to process a string beginning with 'ABC' plus the three characters in the next three positions of memory. Depending on memory utilization, the extra three characters could be anything.

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

Number and Order of Arguments

The number of arguments required varies according to each function. Supplied functions may require up to six arguments. User-written subroutines may require a maximum of 200 arguments including the output argument. If a function requires more than 200 arguments, you must use two or more calls to pass the arguments to the function.

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

Verifying Function Parameters

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

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

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

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

Enable Parameter Verification

Parameter verification can be enabled only for DEFINE FUNCTIONs and supplied functions. If your site has a locally written function with the same name as a supplied function, the USERFNS setting determines which function is used.

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

where:

SYSTEM

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

LOCAL

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

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

Control Function Parameter Verification

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

```
SET USERFCHK = setting
```

where:

setting

Can be one of the following:

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

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

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

- **OFF** does not verify parameters except in the following cases:
  - If a parameter that is too long would overwrite the memory area in which the computational code is stored, the size is automatically reduced without issuing a message.
  - If an alphanumeric parameter is too short, it is padded with blanks to the correct length.

> **Note:**
> - The OFF setting will be deprecated in a future release.
> - We strongly recommend that you not use the OFF setting, as disabling parameter checking can lead to unexpected issues.

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

Note:

- If a parameter provided is the incorrect type, verification fails and processing terminates.
- Errors encountered during subroutine processing, unless fatal at the system level, are communicated to the calling routine through the return of an unchanged return parameter, which is the last parameter in the subroutine call. This is always communicated as spaces for alphanumeric outputs.

Verifying Parameters With Correctable Errors

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

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

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

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

Verifying Parameters With Uncorrectable Errors

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

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

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

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

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

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

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

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

Call a Function From a COMPUTE, DEFINE, or VALIDATE Command

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

where:

filename

Is the data source being used.

tempfield

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

format

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

function

Is the name of the function.

input1, input2, input3...

Are the arguments.

outfield

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

In Dialogue Manager, you must specify the format.

Calling a Function From a Dialogue Manager Command

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

- From a -SET command, storing the result of a function in a variable. For more information, see Calling a Function From a Dialogue Manager Command.
- From an -IF command. For more information, see Calling a Function in WHERE or IF Criteria.
- From an operating system -RUN command. For more information, see Calling a Function From a Dialogue Manager Command.

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

If the function expects an alphanumeric string and the input is a numeric string, incorrect results will occur because of conversion to floating-point double-precision. To resolve this problem, append a non-numeric character to the end of the string, but do not count this extra character in the length of the argument.

Dialogue Manager date variables such as &YYMD return alphanumeric legacy dates, not a date format (an offset from a base date). If a function requires a date offset rather than a legacy date, you must convert any date variable to a date offset (using the DATECVT function) before using it as an argument. You can then convert the result back to a legacy date, again with the DATECVT function. For example:

```
-SET &TODAY_OFFSET=DATECVT(&YYMD , 'I8YYMD' , 'YYMD');
-SET &BEG_CUR_YR=DATEMOV(&TODAY_OFFSET.EVAL , 'BOY');
-SET &CLOSE_DTBOY=DATECVT(&BEG_CUR_YR.EVAL , 'YYMD' , 'I8YYMD')';
```

Assigning the Result of a Function to a Variable

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

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

Assign the Result of a Function to a Variable

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

where:

variable

Is the variable to which the result will be assigned.

function

Is the function.

arg1, arg2

Are the function's arguments.

.LENGTH

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

format

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

Calling a Function From a -SET Command

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

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

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

Branching Based on the Result of a Function

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

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

Branch Based on the Result of a Function

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

where:

function

Is the function.

args

Are the arguments.

relation

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

expression

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

label1, label2

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

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

ELSE GOTO

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

Branching Based on the Result of a Function

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

The procedure processes as follows:

1. It prompts for the start date of a project in YYMMDD format.

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

Calling a Function From an Operating System RUN Command

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

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

Call a Function From an Operating System -RUN Command

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

where:

-TSO|-MVS

Is the operating system.

function

Is the name of the function.

input1, input2,...

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

&output

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

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

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

Calling a Function From an Operating System -RUN Command

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

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

Calling a Function From Another Function

A function can be an argument for another function.

Call a Function From Another Function

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

where:

field

Is the field that contains the result of the function.

function

Is a function.

arguments

Are arguments for *function*.

function2

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

arguments2

Are arguments for *function2*.

Calling a Function From Another Function

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

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

Calling a Function in WHERE or IF Criteria

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

Call a Function in WHERE Criteria

```
WHERE function relation expression
```

where:

function

Is a function.

relation

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

expression

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

Call a Function in IF Criteria

```
IF function relation value
```

where:

function

Is a function.

relation

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

value

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

Calling a Function in WHERE Criteria

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

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

The output is:

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

Using a Calculation or Compound IF Command

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

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

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

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

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

Calling a Function in WHEN Criteria

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

Call a Function in WHEN Criteria

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

or

```
WHEN NOT(function)
```

where:

function

Is a function.

value

Is a value or logical expression.

relation

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

Calling a Function in WHEN Criteria

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

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

The output is:

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

Calling a Function From a RECAP Command

You can call a function from an FML RECAP command.

Call a Function From a RECAP Command

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

where:

name

Is the name of the calculation.

n

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

n,m

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

n,m,i

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

format1

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

function

Is the function.

input1,...

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

format2

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

Calling a Function in a RECAP Command

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

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

The output is:

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

Storing and Accessing an External Function

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

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

Storing and Accessing a Function on z/OS

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

Allocate a Load Library in z/OS Batch

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

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

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

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

Allocate a Load Library in TSO

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

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

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

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

Allocate a Load Library

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

or

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

where:

MVS|TSO

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

USERLIB

Is the ddname to which you allocate a load library.

lib1 lib2 lib3...

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

Allocating the FUSELIB.LOAD Load Library

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

or

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

Concatenating a Load Library to USERLIB In TSO

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

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

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

Concatenating a Load Library to STEPLIB in Batch (JCL)

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

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

Storing and Accessing a Function on UNIX

No extra work is required.

Storing and Accessing a Function on Windows

No extra work is required.

Simplified Analytic Functions

The analytic functions enable you to perform calculations and retrievals using multiple rows in the internal matrix.

FORECAST_MOVAVE: Using a Simple Moving Average

A simple moving average is a series of arithmetic means calculated with a specified number of values from a field. Each new mean in the series is calculated by dropping the first value used in the prior calculation, and adding the next data value to the calculation.

Simple moving averages are sometimes used to analyze trends in stock prices over time. In this scenario, the average is calculated using a specified number of periods of stock prices. A disadvantage to this indicator is that because it drops the oldest values from the calculation as it moves on, it loses its memory over time. Also, mean values are distorted by extreme highs and lows, since this method gives equal weight to each point.

Predicted values beyond the range of the data values are calculated using a moving average that treats the calculated trend values as new data points.

The first complete moving average occurs at the n^{th} data point because the calculation requires *n* values. This is called the lag. The moving average values for the lag rows are calculated as follows: the first value in the moving average column is equal to the first data value, the second value in the moving average column is the average of the first two data values, and so on until the n^{th} row, at which point there are enough values to calculate the moving average with the number of values specified.

Calculate a Simple Moving Average Column

```
FORECAST_MOVAVE(display, infield, interval,
 npredict, npoint1)
```

where:

display

Keyword

Specifies which values to display for rows of output that represent existing data. Valid values are:

- **INPUT_FIELD.** This displays the original field values for rows that represent existing data.
- **MODEL_DATA.** This displays the calculated values for rows that represent existing data.

Note: You can show both types of output for any field by creating two independent COMPUTE commands in the same request, each with a different display option.

infield

Is any numeric field. It can be the same field as the result field, or a different field. It cannot be a date-time field or a numeric field with date display options.

interval

Is the increment to add to each sort field value (after the last data point) to create the next value. This must be a positive integer. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the sort field values is converted to the same format as the sort field.

For date fields, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days. If the format is YM, the 2 is interpreted as meaning two months.

npredict

Is the number of predictions for FORECAST to calculate. It must be an integer greater than or equal to zero. Zero indicates that you do not want predictions, and is only supported with a non-recursive FORECAST.

npoint1

Is the number of values to average for the MOVAVE method.

Calculating a New Simple Moving Average Column

This request defines an integer value named PERIOD to use as the independent variable for the moving average. It predicts three periods of values beyond the range of the retrieved data. The MOVAVE column on the report output shows the calculated moving average numbers for existing data points.

```
DEFINE FILE GGSALES
SDATE/YYM = DATE;
SYEAR/Y = SDATE;
SMONTH/M = SDATE;
PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM UNITS DOLLARS
COMPUTE  MOVAVE/D10.1= FORECAST_MOVAVE(MODEL_DATA, DOLLARS,1,3,3);
BY CATEGORY BY PERIOD
WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is:

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	741,731.5
	3	61608	765078	749,513.7
	4	57050	691274	712,897.3
	5	59229	720444	725,598.7
	6	58466	742457	718,058.3
	7	60771	747253	736,718.0
	8	54633	655896	715,202.0
	9	57829	730317	711,155.3
	10	57012	724412	703,541.7
	11	51110	620264	691,664.3
	12	58981	762328	702,334.7
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9
Food	1	54394	672727	672,727.0
	2	54894	699073	685,900.0
	3	52713	642802	671,534.0
	4	58026	718514	686,796.3
	5	53289	660740	674,018.7
	6	58742	734705	704,653.0
	7	60127	760586	718,677.0
	8	55622	695235	730,175.3
	9	55787	683140	712,987.0
	10	57340	713768	697,381.0
	11	57459	710138	702,348.7
	12	57290	705315	709,740.3
	13	0	0	708,397.8
	14	0	0	707,817.7
	15	0	0	708,651.9

In the report, the number of values to use in the average is 3 and there are no UNITS or DOLLARS values for the generated PERIOD values.

Each average (MOVAVE value) is computed using DOLLARS values where they exist. The calculation of the moving average begins in the following way:

- The first MOVAVE value (801,123.0) is equal to the first DOLLARS value.
- The second MOVAVE value (741,731.5) is the mean of DOLLARS values one and two: (801,123 + 682,340) /2.
- The third MOVAVE value (749,513.7) is the mean of DOLLARS values one through three: (801,123 + 682,340 + 765,078) / 3.
- The fourth MOVAVE value (712,897.3) is the mean of DOLLARS values two through four: (682,340 + 765,078 + 691,274) /3.

For predicted values beyond the supplied values, the calculated MOVAVE values are used as new data points to continue the moving average. The predicted MOVAVE values (starting with 694,975.6 for PERIOD 13) are calculated using the previous MOVAVE values as new data points. For example, the first predicted value (694,975.6) is the average of the data points from periods 11 and 12 (620,264 and 762,328) and the moving average for period 12 (702,334.7). The calculation is: 694,975 = (620,264 + 762,328 + 702,334.7)/3.

Displaying Original Field Values in a Simple Moving Average Column

This request defines an integer value named PERIOD to use as the independent variable for the moving average. It predicts three periods of values beyond the range of the retrieved data. It uses the keyword INPUT_FIELD as the first argument in the FORECAST parameter list. The trend values are not displayed in the report. The actual data values for DOLLARS are followed by the predicted values in the report column.

```
DEFINE FILE GGSALES
SDATE/YYM = DATE;
SYEAR/Y = SDATE;
SMONTH/M = SDATE;
PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM UNITS DOLLARS
COMPUTE MOVAVE/D10.1 = FORECAST_MOVAVE(INPUT_FIELD,DOLLARS,1,3,3);
BY CATEGORY BY PERIOD
WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image:

Category	PERIOD	Unit Sales	Dollar Sales	MOVAVE
Coffee	1	61666	801123	801,123.0
	2	54870	682340	682,340.0
	3	61608	765078	765,078.0
	4	57050	691274	691,274.0
	5	59229	720444	720,444.0
	6	58466	742457	742,457.0
	7	60771	747253	747,253.0
	8	54633	655896	655,896.0
	9	57829	730317	730,317.0
	10	57012	724412	724,412.0
	11	51110	620264	620,264.0
	12	58981	762328	762,328.0
	13	0	0	694,975.6
	14	0	0	719,879.4
	15	0	0	705,729.9
Food	1	54394	672727	672,727.0
	2	54894	699073	699,073.0
	3	52713	642802	642,802.0
	4	58026	718514	718,514.0
	5	53289	660740	660,740.0
	6	58742	734705	734,705.0
	7	60127	760586	760,586.0
	8	55622	695235	695,235.0
	9	55787	683140	683,140.0
	10	57340	713768	713,768.0
	11	57459	710138	710,138.0
	12	57290	705315	705,315.0
	13	0	0	708,397.8
	14	0	0	707,817.7
	15	0	0	708,651.9

FORECAST_EXPAVE: Using Single Exponential Smoothing

The single exponential smoothing method calculates an average that allows you to choose weights to apply to newer and older values.

The following formula determines the weight given to the newest value.

```
k = 2/(1+n)
```

where:

k

Is the newest value.

n

Is an integer greater than one. Increasing *n* increases the weight assigned to the earlier observations (or data instances), as compared to the later ones.

The next calculation of the exponential moving average (EMA) value is derived by the following formula:

```
EMA = (EMA * (1-k)) + (datavalue * k)
```

This means that the newest value from the data source is multiplied by the factor *k* and the current moving average is multiplied by the factor (1-*k*). These quantities are then summed to generate the new EMA.

Note: When the data values are exhausted, the last data value in the sort group is used as the next data value.

Calculate a Single Exponential Smoothing Column

```
FORECAST_EXPAVE(display, infield, interval,
 npredict, npoint1)
```

where:

display

Keyword

Specifies which values to display for rows of output that represent existing data. Valid values are:

- **INPUT_FIELD.** This displays the original field values for rows that represent existing data.

- **MODEL_DATA.** This displays the calculated values for rows that represent existing data.

> **Note:** You can show both types of output for any field by creating two independent COMPUTE commands in the same request, each with a different display option.

infield

Is any numeric field. It can be the same field as the result field, or a different field. It cannot be a date-time field or a numeric field with date display options.

interval

Is the increment to add to each sort field value (after the last data point) to create the next value. This must be a positive integer. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the sort field values is converted to the same format as the sort field.

For date fields, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days. If the format is YM, the 2 is interpreted as meaning two months.

npredict

Is the number of predictions for FORECAST to calculate. It must be an integer greater than or equal to zero. Zero indicates that you do not want predictions, and is only supported with a non-recursive FORECAST.

npoint1

For EXPAVE, this number is used to calculate the weights for each component in the average. This value must be a positive whole number. The weight, k, is calculated by the following formula:

```
k=2/(1+npoint1)
```

Calculating a Single Exponential Smoothing Column

The following defines an integer value named PERIOD to use as the independent variable for the moving average. It predicts three periods of values beyond the range of retrieved data.

```
DEFINE FILE GGSALES
SDATE/YYM = DATE;
SYEAR/Y = SDATE;
SMONTH/M = SDATE;
PERIOD/I2 = SMONTH;
END
TABLE FILE GGSALES
SUM UNITS DOLLARS
COMPUTE EXPAVE/D10.1= FORECAST_EXPAVE(MODEL_DATA,DOLLARS,1,3,3);
BY CATEGORY BY PERIOD
WHERE SYEAR EQ 97 AND CATEGORY NE 'Gifts'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image:

```
Category     PERIOD  Unit Sales  Dollar Sales        EXPAVE
--------     ------  ----------  ------------        ------
Coffee            1       61666        801123     801,123.0
                  2       54870        682340     741,731.5
                  3       61608        765078     753,404.8
                  4       57050        691274     722,339.4
                  5       59229        720444     721,391.7
                  6       58466        742457     731,924.3
                  7       60771        747253     739,588.7
                  8       54633        655896     697,742.3
                  9       57829        730317     714,029.7
                 10       57012        724412     719,220.8
                 11       51110        620264     669,742.4
                 12       58981        762328     716,035.2
                 13           0             0     739,181.6
                 14           0             0     750,754.8
                 15           0             0     756,541.4
Food              1       54394        672727     672,727.0
                  2       54894        699073     685,900.0
```

```
                        3       52713        642802     664,351.0
                        4       58026        718514     691,432.5
                        5       53289        660740     676,086.3
                        6       58742        734705     705,395.6
                        7       60127        760586     732,990.8
                        8       55622        695235     714,112.9
                        9       55787        683140     698,626.5
                       10       57340        713768     706,197.2
                       11       57459        710138     708,167.6
                       12       57290        705315     706,741.3
                       13           0             0     706,028.2
                       14           0             0     705,671.6
                       15           0             0     705,493.3
```

In the report, three predicted values of EXPAVE are calculated within each value of CATEGORY. For values outside the range of the data, new PERIOD values are generated by adding the interval value (1) to the prior PERIOD value.

Each average (EXPAVE value) is computed using DOLLARS values where they exist. The calculation of the moving average begins in the following way:

- The first EXPAVE value (801,123.0) is the same as the first DOLLARS value.
- The second EXPAVE value (741,731.5) is calculated as follows. Note that because of rounding and the number of decimal places used, the value derived in this sample calculation varies slightly from the one displayed in the report output:

  ```
  n=3 (number used to calculate weights)
  ```

  ```
  k = 2/(1+n) = 2/4 = 0.5
  ```

  ```
  EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) = (801123*0.5) +
  (682340*0.50) = 400561.5 + 341170 = 741731.5
  ```

- The third EXPAVE value (753,404.8) is calculated as follows:

  ```
  EXPAVE = (EXPAVE*(1-k))+(new-DOLLARS*k) = (741731.5*0.5)+
  (765078*0.50) = 370865.75 + 382539 = 753404.75
  ```

FORECAST_DOUBLEXP: Using Double Exponential Smoothing

Double exponential smoothing produces an exponential moving average that takes into account the tendency of data to either increase or decrease over time without repeating. This is accomplished by using two equations with two constants.

- The first equation accounts for the current time period and is a weighted average of the current data value and the prior average, with an added component (b) that represents the trend for the previous period. The weight constant is k:

  ```
  DOUBLEXP(t) = k * datavalue(t) + (1-k) * ((DOUBLEXP(t-1) + b(t-1))
  ```

- The second equation is the calculated trend value, and is a weighted average of the difference between the current and previous average and the trend for the previous time period. b(*t*) represents the average trend. The weight constant is g:

  ```
  b(t) = g * (DOUBLEXP(t)-DOUBLEXP(t-1)) + (1 - g) * (b(t-1))
  ```

These two equations are solved to derive the smoothed average. The first smoothed average is set to the first data value. The first trend component is set to zero. For choosing the two constants, the best results are usually obtained by minimizing the mean-squared error (MSE) between the data values and the calculated averages. You may need to use nonlinear optimization techniques to find the optimal constants.

The equation used for forecasting beyond the data points with double exponential smoothing is

```
forecast(t+m) = DOUBLEXP(t) + m * b(t)
```

where:

m

Is the number of time periods ahead for the forecast.

Calculate a Double Exponential Smoothing Column

```
FORECAST_DOUBLEXP(display, infield,
interval, npredict, npoint1, npoint2)
```

where:

display

Keyword

Specifies which values to display for rows of output that represent existing data. Valid values are:

- **INPUT_FIELD.** This displays the original field values for rows that represent existing data.
- **MODEL_DATA.** This displays the calculated values for rows that represent existing data.

> **Note:** You can show both types of output for any field by creating two independent COMPUTE commands in the same request, each with a different display option.

infield

Is any numeric field. It can be the same field as the result field, or a different field. It cannot be a date-time field or a numeric field with date display options.

interval

Is the increment to add to each sort field value (after the last data point) to create the next value. This must be a positive integer. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the sort field values is converted to the same format as the sort field.

For date fields, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days. If the format is YM, the 2 is interpreted as meaning two months.

npredict

Is the number of predictions for FORECAST to calculate. It must be an integer greater than or equal to zero. Zero indicates that you do not want predictions, and is only supported with a non-recursive FORECAST.

npoint1

For DOUBLEXP, this number is used to calculate the weights for each component in the average. This value must be a positive whole number. The weight, k, is calculated by the following formula:

```
k=2/(1+npoint1)
```

npoint2

For DOUBLEXP, this positive whole number is used to calculate the weights for each term in the trend. The weight, g, is calculated by the following formula:

```
g=2/(1+npoint2)
```

Calculating a Double Exponential Smoothing Column

The following sums the TRANSTOT field of the VIDEOTRK data source by TRANSDATE, and calculates a single exponential and double exponential moving average. The report columns show the calculated values for existing data points.

```
TABLE FILE VIDEOTRK
SUM TRANSTOT
COMPUTE EXP/D15.1 = FORECAST_EXPAVE(MODEL_DATA,TRANSTOT,1,0,3);
DOUBLEXP/D15.1 = FORECAST_DOUBLEXP(MODEL_DATA,TRANSTOT,1,0,3,3);
BY TRANSDATE
WHERE TRANSDATE NE '19910617'
ON TABLE SET STYLE *
GRID=OFF,$
END
```

The output is shown in the following image:

TRANSDATE	TRANSTOT	EXP	DOUBLEXP
91/06/18	21.25	21.3	21.3
91/06/19	38.17	29.7	35.0
91/06/20	14.23	22.0	30.7
91/06/21	44.72	33.3	39.7
91/06/24	126.28	79.8	86.2
91/06/25	47.74	63.8	80.2
91/06/26	40.97	52.4	65.7
91/06/27	60.24	56.3	61.9
91/06/28	31.00	43.7	45.0

FORECAST_SEASONAL: Using Triple Exponential Smoothing

Triple exponential smoothing produces an exponential moving average that takes into account the tendency of data to repeat itself in intervals over time. For example, sales data that is growing and in which 25% of sales always occur during December contains both trend and seasonality. Triple exponential smoothing takes both the trend and seasonality into account by using three equations with three constants.

For triple exponential smoothing you, need to know the number of data points in each time period (designated as L in the following equations). To account for the seasonality, a seasonal index is calculated. The data is divided by the prior season index and then used in calculating the smoothed average.

- The first equation accounts for the current time period, and is a weighted average of the current data value divided by the seasonal factor and the prior average adjusted for the trend for the previous period. The weight constant is k:

  ```
  SEASONAL(t) = k * (datavalue(t)/I(t-L)) + (1-k) * (SEASONAL(t-1) +
  b(t-1))
  ```

- The second equation is the calculated trend value, and is a weighted average of the difference between the current and previous average and the trend for the previous time period. b(*t*) represents the average trend. The weight constant is g:

  ```
  b(t) = g * (SEASONAL(t)-SEASONAL(t-1)) + (1-g) * (b(t-1))
  ```

- The third equation is the calculated seasonal index, and is a weighted average of the current data value divided by the current average and the seasonal index for the previous season. I(*t*) represents the average seasonal coefficient. The weight constant is p:

  ```
  I(t) = p * (datavalue(t)/SEASONAL(t)) + (1 - p) * I(t-L)
  ```

These equations are solved to derive the triple smoothed average. The first smoothed average is set to the first data value. Initial values for the seasonality factors are calculated based on the maximum number of full periods of data in the data source, while the initial trend is calculated based on two periods of data. These values are calculated with the following steps:

1. The initial trend factor is calculated by the following formula:

   ```
   b(0) = (1/L) ((y(L+1)-y(1))/L + (y(L+2)-y(2))/L + ... + (y(2L) -
   y(L))/L )
   ```

2. The calculation of the initial seasonality factor is based on the average of the data values within each period, A(j) (1<=j<=N):

   ```
   A(j) = ( y((j-1)L+1) + y((j-1)L+2) + ... + y(jL) ) / L
   ```

3. Then, the initial periodicity factor is given by the following formula, where N is the number of full periods available in the data, L is the number of points per period and n is a point within the period (1<= n <= L):

   ```
   I(n) = ( y(n)/A(1) + y(L+n)/A(2) + ... + y((N-1)L+n)/A(N) ) / N
   ```

The three constants must be chosen carefully. The best results are usually obtained by choosing the constants to minimize the mean-squared error (MSE) between the data values and the calculated averages. Varying the values of npoint1 and npoint2 affect the results, and some values may produce a better approximation. To search for a better approximation, you may want to find values that minimize the MSE.

The equation used to forecast beyond the last data point with triple exponential smoothing is:

```
forecast(t+m) = (SEASONAL(t) + m * b(t)) / I(t-L+MOD(m/L))
```

where:

m

Is the number of periods ahead for the forecast.

Calculate a Triple Exponential Smoothing Column

```
FORECAST_SEASONAL(display, infield,
interval, npredict, nperiod, npoint1, npoint2, npoint3)
```

where:

display

Keyword

Specifies which values to display for rows of output that represent existing data. Valid values are:

- **INPUT_FIELD.** This displays the original field values for rows that represent existing data.
- **MODEL_DATA.** This displays the calculated values for rows that represent existing data.

> **Note:** You can show both types of output for any field by creating two independent COMPUTE commands in the same request, each with a different display option.

infield

Is any numeric field. It can be the same field as the result field, or a different field. It cannot be a date-time field or a numeric field with date display options.

interval

Is the increment to add to each sort field value (after the last data point) to create the next value. This must be a positive integer. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the sort field values is converted to the same format as the sort field.

For date fields, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days. If the format is YM, the 2 is interpreted as meaning two months.

npredict

Is the number of predictions for FORECAST to calculate. It must be an integer greater than or equal to zero. Zero indicates that you do not want predictions, and is only supported with a non-recursive FORECAST. For the SEASONAL method, npredict is the number of *periods* to calculate. The number of *points* generated is:

```
nperiod * npredict
```

nperiod

For the SEASONAL method, is a positive whole number that specifies the number of data points in a period.

npoint1

For SEASONAL, this number is used to calculate the weights for each component in the average. This value must be a positive whole number. The weight, k, is calculated by the following formula:

```
k=2/(1+npoint1)
```

npoint2

For SEASONAL, this positive whole number is used to calculate the weights for each term in the trend. The weight, g, is calculated by the following formula:

```
g=2/(1+npoint2)
```

npoint3

For SEASONAL, this positive whole number is used to calculate the weights for each term in the seasonal adjustment. The weight, p, is calculated by the following formula:

```
p=2/(1+npoint3)
```

Calculating a Triple Exponential Smoothing Column

In the following, the data has seasonality but no trend. Therefore, *npoint2* is set high (1000) to make the trend factor negligible in the calculation:

```
TABLE FILE VIDEOTRK
SUM TRANSTOT
COMPUTE SEASONAL/D10.1 = FORECAST_SEASONAL(MODEL_
DATA,TRANSTOT,1,3,3,3,1000,1);
BY TRANSDATE
WHERE TRANSDATE NE '19910617'
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

In the output, *npredict* is 3. Therefore, three periods (nine points, *nperiod * npredict*) are generated.

TRANSDATE	TRANSTOT	SEASONAL
91/06/18	21.25	21.3
91/06/19	38.17	31.0
91/06/20	14.23	34.6
91/06/21	44.72	53.2
91/06/24	126.28	75.3
91/06/25	47.74	82.7
91/06/26	40.97	73.7
91/06/27	60.24	62.9
91/06/28	31.00	66.3
91/06/29		45.7
91/06/30		94.1
91/07/01		53.4
91/07/02		72.3
91/07/03		140.0
91/07/04		75.8
91/07/05		98.9
91/07/06		185.8
91/07/07		98.2

FORECAST_LINEAR: Using a Linear Regression Equation

The linear regression equation estimates values by assuming that the dependent variable (the new calculated values) and the independent variable (the sort field values) are related by a function that represents a straight line:

```
y = mx + b
```

where:

y

Is the dependent variable.

x

Is the independent variable.

m

Is the slope of the line.

b

Is the y-intercept.

FORECAST_LINEAR uses a technique called Ordinary Least Squares to calculate values for *m* and *b* that minimize the sum of the squared differences between the data and the resulting line.

The following formulas show how *m* and *b* are calculated.

$$m = \frac{(\sum xy - (\sum x \bullet \sum y)/n)}{(\sum x^2 - (\sum x)^2/n)}$$

$$b = (\sum y)/n - (m \bullet (\sum x)/n)$$

where:

n

Is the number of data points.

y

Is the data values (dependent variables).

x

Is the sort field values (independent variables).

Trend values, as well as predicted values, are calculated using the regression line equation.

Calculate a Linear Regression Column

```
FORECAST_LINEAR(display, infield, interval,
 npredict)
```

where:

display

Keyword

Specifies which values to display for rows of output that represent existing data. Valid values are:

- **INPUT_FIELD.** This displays the original field values for rows that represent existing data.
- **MODEL_DATA.** This displays the calculated values for rows that represent existing data.

> **Note:** You can show both types of output for any field by creating two independent COMPUTE commands in the same request, each with a different display option.

infield

Is any numeric field. It can be the same field as the result field, or a different field. It cannot be a date-time field or a numeric field with date display options.

interval

Is the increment to add to each sort field value (after the last data point) to create the next value. This must be a positive integer. To sort in descending order, use the BY HIGHEST phrase. The result of adding this number to the sort field values is converted to the same format as the sort field.

For date fields, the minimal component in the format determines how the number is interpreted. For example, if the format is YMD, MDY, or DMY, an interval value of 2 is interpreted as meaning two days. If the format is YM, the 2 is interpreted as meaning two months.

npredict

Is the number of predictions for FORECAST to calculate. It must be an integer greater than or equal to zero. Zero indicates that you do not want predictions, and is only supported with a non-recursive FORECAST.

Calculating a New Linear Regression Field

The following request calculates a regression line using the VIDEOTRK data source of QUANTITY by TRANSDATE. The interval is one day, and three predicted values are calculated.

```
TABLE FILE VIDEOTRK
SUM QUANTITY
COMPUTE FORTOT=FORECAST_LINEAR(MODEL_DATA,QUANTITY,1,3);
BY TRANSDATE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image:

TRANSDATE	QUANTITY	FORTOT
06/17/91	12	6.63
06/18/91	2	6.57
06/19/91	5	6.51
06/20/91	3	6.45
06/21/91	7	6.39
06/24/91	12	6.21
06/25/91	8	6.15
06/26/91	2	6.09
06/27/91	9	6.03
06/28/91	3	5.97
06/29/91		5.91
06/30/91		5.85
07/01/91		5.79

Note:

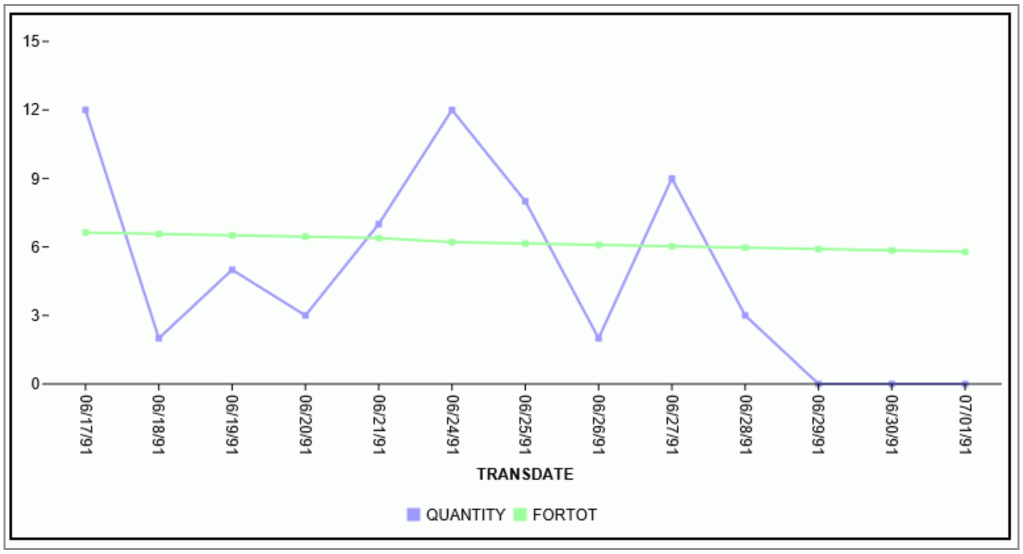
- Three predicted values of FORTOT are calculated. For values outside the range of the data, new TRANSDATE values are generated by adding the interval value (1) to the prior TRANSDATE value.
- There are no QUANTITY values for the generated FORTOT values.
- Each FORTOT value is computed using a regression line, calculated using all of the actual data values for QUANTITY.

  TRANSDATE is the independent variable (x) and QUANTITY is the dependent variable (y). The equation is used to calculate QUANTITY FORECAST trend and predicted values.

The following version of the request charts the data values and the regression line.

```
GRAPH FILE VIDEOTRK
SUM QUANTITY
COMPUTE FORTOT=FORECAST_LINEAR(MODEL_DATA,QUANTITY,1,3);
BY TRANSDATE
ON GRAPH HOLD FORMAT JSCHART
ON GRAPH SET LOOKGRAPH VLINE
END
```

The output is shown in the following image.



PARTITION_AGGR: Creating Rolling Calculations

Using the PARTITION_AGGR function, you can generate rolling calculations based on a block of rows from the internal matrix of a TABLE request. In order to determine the limits of the rolling calculations, you specify a partition of the data based on either a sort field or the entire TABLE. Within either type of break, you can start calculating from the beginning of the break or a number of rows prior to or subsequent to the current row. You can stop the rolling calculation at the current row, a row past the start point, or the end of the partition.

By default, the field values used in the calculations are the summed values of a measure in the request. Certain prefix operators can be used to add a column to the internal matrix and use that column in the rolling calculations. The rolling calculation can be SUM, AVE, CNT, MIN, MAX, FST, or LST.

Generate Rolling Calculations Using PARTITION_AGGR

```
PARTITION_AGGR([prefix.]measure,reset_key,lower,upper,operation)
```

where:

prefix.

Defines an aggregation operator to apply to the measure before using it in the rolling calculation. Valid operators are:

- **SUM.** which calculates the sum of the measure field values. SUM is the default operator.
- **CNT.** which calculates a count of the measure field values.
- **AVE.** which calculates the average of the measure field values.
- **MIN.** which calculates the minimum of the measure field values.
- **MAX.** which calculates the maximum of the measure field values.
- **FST.** which retrieves the first value of the measure field.
- **LST.** which retrieves the last value of the measure field.
- **STDP.** which calculates the population standard deviation.
- **STDS.** which calculates the sample standard deviation.

Note: The operators PCT., RPCT., TOT., MDN., and DST. are not supported. COMPUTEs that reference those unsupported operators are also not supported.

measure

Is the measure field to be aggregated. It can be a real field in the request or a calculated value generated with the COMPUTE command, as long as the COMPUTE does not reference an unsupported prefix operator.

reset_key

Identifies the point at which the calculation restarts. Valid values are:

- The name of a sort field in the request.
- PRESET, which uses the value of the PARTITION_ON parameter, as described in Specify the Partition Size for Simplified Statistical Functions.
- TABLE, which indicates that there is no break on a sort field.

The sort field may use BY HIGHEST to indicate a HIGH-TO-LOW sort. ACROSS COLUMNS AND is supported. BY ROWS OVER and FOR are not supported.

lower

Identifies the starting point for the rolling calculation. Valid values are:

- **n, -n**, which starts the calculation *n* rows forward or back from the current row.
- **B**, which starts the calculation at the beginning of the current sort break (the first line with the same sort field value as the current line).

upper

Identifies the ending point of the rolling calculation. The *lower* row value must precede the *upper* row value.

Valid values are:

- **C**, which ends the rolling calculation at the current row in the internal matrix.
- **n, -n**, which ends the calculation *n* rows forward or back from the current row.
- **E**, which ends the rolling calculation at the end of the sort break (the last line with the same sort value as the current row.)

Note: The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

operation

Specifies the rolling calculation used on the values in the internal matrix. Supported operations are:

- **SUM,** which calculates a rolling sum.
- **AVE,** which calculates a rolling average.
- **CNT,** which counts the rows in the partition.
- MEDIAN.
- **MIN,** which returns the minimum value in the partition.
- **MAX,** which returns the maximum value in the partition.
- **MEDIAN,** which returns the median value in the partition.
- **MODE,** which returns the mode value in the partition.
- **FST,** which returns the first value in the partition.
- **LST,** which returns the last value in the partition.
- **STDP,** which returns the population standard deviation in the partition. Requires

using the verb PRINT to avoid duplicate aggregation.

- **STDS,** which returns the sample standard deviation in the partition. Requires using the verb PRINT to avoid duplicate aggregation.

The calculation is performed prior to any WHERE TOTAL tests, but after any WHERE_GROUPED tests.

Calculating a Rolling Average

The following request calculates a rolling average of the current line and the previous line in the internal matrix, within the quarter.

```
TABLE FILE WFLITE
SUM COGS_US
COMPUTE AVE1/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR, -1, C, AVE);
BY BUSINESS_REGION
BY TIME_QTR
BY TIME_MTH
WHERE BUSINESS_REGION EQ 'North America' OR 'South America'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. Within each quarter, the first average is just the value from Q1, as going back 1 would cross a boundary. The second average is calculated using the first two rows within that quarter, and the third average is calculated using rows 2 and 3 within the quarter.

Customer Business Region	Sale Quarter	Sale Month	Cost of Goods	AVE1
North America	1	1	$26,361,956.00	$26,361,956.00
		2	$24,348,729.00	$25,355,342.50
		3	$26,118,420.00	$25,233,574.50
	2	4	$23,776,352.00	$23,776,352.00
		5	$24,717,633.00	$24,246,992.50
		6	$24,284,736.00	$24,501,184.50
	3	7	$25,317,633.00	$25,317,633.00
		8	$25,916,286.00	$25,616,959.50
		9	$24,968,297.00	$25,442,291.50
	4	10	$30,717,478.00	$30,717,478.00
		11	$30,055,782.00	$30,386,630.00
		12	$32,225,143.00	$31,140,462.50
South America	1	1	$3,216,999.00	$3,216,999.00
		2	$2,745,677.00	$2,981,338.00
		3	$3,163,526.00	$2,954,601.50
	2	4	$2,852,809.00	$2,852,809.00
		5	$2,952,020.00	$2,902,414.50
		6	$2,918,017.00	$2,935,018.50
	3	7	$2,961,406.00	$2,961,406.00
		8	$3,077,824.00	$3,019,615.00
		9	$2,895,280.00	$2,986,552.00
	4	10	$3,642,505.00	$3,642,505.00
		11	$3,482,327.00	$3,562,416.00
		12	$3,517,651.00	$3,499,989.00

The following changes the rolling average to start from the beginning of the sort break.

```
COMPUTE AVE1/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR ,B, C, AVE);
```

The output is shown in the following image. Within each quarter, the first average is just the value from Q1, as going back would cross a boundary. The second average is calculated using the first two rows within that quarter, and the third average is calculated using rows 1 through 3 within the quarter.

Customer Business Region	Sale Quarter	Sale Month	Cost of Goods	AVE1
North America	1	1	$26,361,956.00	$26,361,956.00
		2	$24,348,729.00	$25,355,342.50
		3	$26,118,420.00	$25,609,701.67
	2	4	$23,776,352.00	$23,776,352.00
		5	$24,717,633.00	$24,246,992.50
		6	$24,284,736.00	$24,259,573.67
	3	7	$25,317,633.00	$25,317,633.00
		8	$25,916,286.00	$25,616,959.50
		9	$24,968,297.00	$25,400,738.67
	4	10	$30,717,478.00	$30,717,478.00
		11	$30,055,782.00	$30,386,630.00
		12	$32,225,143.00	$30,999,467.67
South America	1	1	$3,216,999.00	$3,216,999.00
		2	$2,745,677.00	$2,981,338.00
		3	$3,163,526.00	$3,042,067.33
	2	4	$2,852,809.00	$2,852,809.00
		5	$2,952,020.00	$2,902,414.50
		6	$2,918,017.00	$2,907,615.33
	3	7	$2,961,406.00	$2,961,406.00
		8	$3,077,824.00	$3,019,615.00
		9	$2,895,280.00	$2,978,170.00
	4	10	$3,642,505.00	$3,642,505.00
		11	$3,482,327.00	$3,562,416.00
		12	$3,517,651.00	$3,547,494.33

The following command uses the partition boundary TABLE.

```
COMPUTE AVE1/D12.2M = PARTITION_AGGR(COGS_US, TABLE, B, C, AVE);
```

The output is shown in the following image. The rolling average keeps adding the next row to the average with no sort field break.

Customer Business Region	Sale Quarter	Sale Month	Cost of Goods	AVE1
North America	1	1	$26,361,956.00	$26,361,956.00
		2	$24,348,729.00	$25,355,342.50
		3	$26,118,420.00	$25,609,701.67
	2	4	$23,776,352.00	$25,151,364.25
		5	$24,717,633.00	$25,064,618.00
		6	$24,284,736.00	$24,934,637.67
	3	7	$25,317,633.00	$24,989,351.29
		8	$25,916,286.00	$25,105,218.13
		9	$24,968,297.00	$25,090,004.67
	4	10	$30,717,478.00	$25,652,752.00
		11	$30,055,782.00	$26,053,027.45
		12	$32,225,143.00	$26,567,370.42
South America	1	1	$3,216,999.00	$24,771,188.00
		2	$2,745,677.00	$23,197,937.21
		3	$3,163,526.00	$21,862,309.80
	2	4	$2,852,809.00	$20,674,216.00
		5	$2,952,020.00	$19,631,733.88
		6	$2,918,017.00	$18,703,194.06
	3	7	$2,961,406.00	$17,874,678.89
		8	$3,077,824.00	$17,134,836.15
		9	$2,895,280.00	$16,456,762.05
	4	10	$3,642,505.00	$15,874,295.82
		11	$3,482,327.00	$15,335,514.57
		12	$3,517,651.00	$14,843,103.58

Usage Notes for PARTITION_AGGR

- Fields referenced in the PARTITION_AGGR parameters but not previously mentioned in the request will *not* be counted in column notation or propagated to HOLD files.
- Using the WITHIN phrase for a sum is the same as computing PARTITION_AGGR on the WITHIN sort field from B (beginning of sort break) to E (end of sort break) using SUM, as in the following example.

```
TABLE FILE WFLITE
SUM COGS_US WITHIN TIME_QTR AS 'WITHIN Qtr'
COMPUTE PART_WITHIN_QTR/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR,
```

```
B, E, SUM);
BY BUSINESS_REGION AS Region
BY TIME_QTR
BY TIME_MTH
WHERE BUSINESS_REGION EQ 'North America' OR 'South America'
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Region	Sale Quarter	Sale Month	WITHIN Qtr	PART_WITHIN_QTR
North America	1	1	$76,829,105.00	$76,829,105.00
		2	$76,829,105.00	$76,829,105.00
		3	$76,829,105.00	$76,829,105.00
	2	4	$72,778,721.00	$72,778,721.00
		5	$72,778,721.00	$72,778,721.00
		6	$72,778,721.00	$72,778,721.00
	3	7	$76,202,216.00	$76,202,216.00
		8	$76,202,216.00	$76,202,216.00
		9	$76,202,216.00	$76,202,216.00
	4	10	$92,998,403.00	$92,998,403.00
		11	$92,998,403.00	$92,998,403.00
		12	$92,998,403.00	$92,998,403.00
South America	1	1	$9,126,202.00	$9,126,202.00
		2	$9,126,202.00	$9,126,202.00
		3	$9,126,202.00	$9,126,202.00
	2	4	$8,722,846.00	$8,722,846.00
		5	$8,722,846.00	$8,722,846.00
		6	$8,722,846.00	$8,722,846.00
	3	7	$8,934,510.00	$8,934,510.00
		8	$8,934,510.00	$8,934,510.00
		9	$8,934,510.00	$8,934,510.00
	4	10	$10,642,483.00	$10,642,483.00
		11	$10,642,483.00	$10,642,483.00
		12	$10,642,483.00	$10,642,483.00

With other types of calculations, the results are not the same. For example, the following request calculates the average within quarter using the WITHIN phrase and the average within quarter using PARTITION_AGGR.

```
TABLE FILE WFLITE
SUM  COGS_US AS Cost
CNT.COGS_US AS Count AVE.COGS_US WITHIN TIME_QTR AS 'Ave Within'
COMPUTE PART_WITHIN_QTR/D12.2M = PARTITION_AGGR(COGS_US, TIME_QTR,
B, E, AVE);
BY BUSINESS_REGION AS Region
BY TIME_QTR
ON TIME_QTR SUBTOTAL COGS_US CNT.COGS_US
BY TIME_MTH
WHERE BUSINESS_REGION EQ 'North America'
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The average using the WITHIN phrase divides the total cost for the quarter by the total count of instances for the quarter (for example, $76,829,105.00/ 252850 = $303.85), while PARTITION_AGGR divides the total cost for the quarter by the number of report rows in the quarter (for example, $76,829,105.00/3 = $25,609,701.67).

Region	Sale Quarter	Sale Month	Cost	Count	Ave Within	PART_WITHIN_QTR
North America	1	1	$26,361,956.00	86369	$303.85	$25,609,701.67
		2	$24,348,729.00	79791	$303.85	$25,609,701.67
		3	$26,118,420.00	86690	$303.85	$25,609,701.67
*TOTAL TIME_QTR 1			$76,829,105.00	252850		
	2	4	$23,776,352.00	79093	$303.40	$24,259,573.67
		5	$24,717,633.00	81317	$303.40	$24,259,573.67
		6	$24,284,736.00	79469	$303.40	$24,259,573.67
*TOTAL TIME_QTR 2			$72,778,721.00	239879		
	3	7	$25,317,633.00	82158	$308.06	$25,400,738.67
		8	$25,916,286.00	83941	$308.06	$25,400,738.67
		9	$24,968,297.00	81262	$308.06	$25,400,738.67
*TOTAL TIME_QTR 3			$76,202,216.00	247361		
	4	10	$30,717,478.00	99572	$309.47	$30,999,467.67
		11	$30,055,782.00	97042	$309.47	$30,999,467.67
		12	$32,225,143.00	103898	$309.47	$30,999,467.67
*TOTAL TIME_QTR 4			$92,998,403.00	300512		
TOTAL			$318,808,445.00	1040602		

- If you use PARTITION_AGGR to perform operations for specific time periods using an offset, for example, an operation on the quarters for different years, you must make sure that every quarter is represented. If some quarters are missing for some years, the offset will not access the correct data. In this case, generate a HOLD file that has every quarter represented for every year (you can use BY QUARTER ROWS OVER 1 OVER 2 OVER 3 OVER 4) and use PARTITION_AGGR on the HOLD file.

PARTITION_REF: Using Prior or Subsequent Field Values in Calculations

Use of LAST in a calculation retrieves the LAST value of the specified field the last time this calculation was performed. In contrast, the PARTITION_REF function enables you to specify both how many rows back or forward to go in the output in order to retrieve a value, and a sort break within which the retrieval will be contained.

Retrieve Prior or Subsequent Field Values for Use in a Calculation

```
PARTITION_REF([prefix.]field, reset_key, offset)
```

where:

prefix

Is optional. If used, it can be one of the following aggregation operators:

- **AVE.** Average
- **MAX.** Maximum
- **MIN.** Minimum
- **CNT.** Count
- **SUM.** Sum

field

Is the field whose value is to be retrieved.

reset_key

Identifies the point at which the retrieval break restarts. Valid values are:

- The name of a sort field in the request.
- PRESET, which uses the value of the PARTITION_ON parameter, as described in Specify the Partition Size for Simplified Statistical Functions.
- TABLE, which indicates that there is no break on a sort field.

The sort field may use BY HIGHEST to indicate a HIGH-TO-LOW sort. ACROSS COLUMNS AND is supported. BY ROWS OVER and FOR are not supported.

Note: The values used in the retrieval depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

offset

Is the integer number of records to go forward (for a positive offset) or backward (for a negative offset) to retrieve the value.

If the offset is prior to the partition boundary sort value, the return will be the default value for the field. The calculation is performed prior to any WHERE TOTAL tests, but after WHERE_GROUPED tests.

Retrieving a Previous Record With PARTITION_REF

The following request retrieves the previous record within the sort field PRODUCT_CATEGORY.

```
TABLE FILE WFLITE
SUM DAYSDELAYED
COMPUTE NEWDAYS/I5=PARTITION_REF(DAYSDELAYED, PRODUCT_CATEGORY, -1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value within each sort break is zero because there is no prior record to retrieve.

Product Category	Product Subcategory	Days Delayed	NEWDAYS
Accessories	Charger	12,301	0
	Headphones	26,670	12301
	Universal Remote Controls	20,832	26670
Camcorder	Handheld	29,446	0
	Professional	1,531	29446
	Standard	22,248	1531
Computers	Smartphone	24,113	0
	Tablet	21,293	24113
Media Player	Blu Ray	78,989	0
	DVD Players	31	78989
	Streaming	8,153	31
Stereo Systems	Home Theater Systems	47,214	0
	Receivers	17,999	47214
	Speaker Kits	28,468	17999
	iPod Docking Station	37,556	28468
Televisions	Flat Panel TV	10,941	0
Video Production	Video Editing	23,553	0

The following request retrieves the average cost of goods from two records prior to the current record within the PRODUCT_CATEGORY sort field.

```
TABLE FILE WFLITE
SUM COGS_US AVE.COGS_US AS Average
COMPUTE PartitionAve/D12.2M=PARTITION_REF(AVE.COGS_US, PRODUCT_CATEGORY,
-2);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Product Category	Product Subcategory	Cost of Goods	Average	PartitionAve
Accessories	Charger	$2,052,711.00	$27.48	$.00
	Headphones	$51,663,564.00	$319.05	$.00
	Universal Remote Controls	$36,037,623.00	$285.21	$27.48
Camcorder	Handheld	$20,576,916.00	$116.02	$.00
	Professional	$35,218,308.00	$3,897.56	$.00
	Standard	$49,071,633.00	$359.54	$116.02
Computers	Smartphone	$44,035,774.00	$302.01	$.00
	Tablet	$25,771,890.00	$247.89	$.00
Media Player	Blu Ray	$181,112,921.00	$376.11	$.00
	DVD Players	$3,756,254.00	$281.45	$.00
	DVD Players - Portable	$306,576.00	$77.01	$376.11
	Streaming	$5,064,730.00	$104.99	$281.45
Stereo Systems	Boom Box	$840,373.00	$125.67	$.00
	Home Theater Systems	$56,428,589.00	$199.38	$.00
	Receivers	$40,329,668.00	$377.67	$125.67
	Speaker Kits	$81,396,140.00	$471.02	$199.38
	iPod Docking Station	$26,119,093.00	$118.66	$377.67
Televisions	CRT TV	$1,928,416.00	$590.09	$.00
	Flat Panel TV	$59,077,345.00	$900.19	$.00
	Portable TV	$545,348.00	$95.74	$590.09
Video Production	Video Editing	$40,105,657.00	$283.23	$.00

Replacing the function call with the following syntax changes the partition boundary to TABLE.

```
COMPUTE PartitionAve/D12.2M=PARTITION_REF(AVE.COGS_US, TABLE, -2);
```

The output is shown in the following image.

Product Category	Product Subcategory	Cost of Goods	Average	PartitionAve
Accessories	Charger	$2,052,711.00	$27.48	$.00
	Headphones	$51,663,564.00	$319.05	$.00
	Universal Remote Controls	$36,037,623.00	$285.21	$27.48
Camcorder	Handheld	$20,576,916.00	$116.02	$319.05
	Professional	$35,218,308.00	$3,897.56	$285.21
	Standard	$49,071,633.00	$359.54	$116.02
Computers	Smartphone	$44,035,774.00	$302.01	$3,897.56
	Tablet	$25,771,890.00	$247.89	$359.54
Media Player	Blu Ray	$181,112,921.00	$376.11	$302.01
	DVD Players	$3,756,254.00	$281.45	$247.89
	DVD Players - Portable	$306,576.00	$77.01	$376.11
	Streaming	$5,064,730.00	$104.99	$281.45
Stereo Systems	Boom Box	$840,373.00	$125.67	$77.01
	Home Theater Systems	$56,428,589.00	$199.38	$104.99
	Receivers	$40,329,668.00	$377.67	$125.67
	Speaker Kits	$81,396,140.00	$471.02	$199.38
	iPod Docking Station	$26,119,093.00	$118.66	$377.67
Televisions	CRT TV	$1,928,416.00	$590.09	$471.02
	Flat Panel TV	$59,077,345.00	$900.19	$118.66
	Portable TV	$545,348.00	$95.74	$590.09
Video Production	Video Editing	$40,105,657.00	$283.23	$900.19

Usage Notes for PARTITION_REF

- Fields referenced in the PARTITION_REF parameters but not previously mentioned in the request, will *not* be counted in column notation or propagated to HOLD files.

INCREASE: Calculating the Difference Between the Current and a Prior Value of a Field

Given an aggregated input field and a negative offset, INCREASE calculates the difference between the value in the current row of the report output and one or more prior rows, within a sort break or the entire table. The reset point for the calculation is determined by the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

Note: The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

Calculate the Difference Between the Current and a Prior Value of a Field

```
INCREASE([prefix.]field, offset)
```

where:

prefix

Is one of the following optional aggregation operators to apply to the field before using it in the calculation:

- **SUM.** which calculates the sum of the field values. SUM is the default value.
- **CNT.** which calculates a count of the field values.
- **AVE.** which calculates the average of the field values.
- **MIN.** which calculates the minimum of the field values.
- **MAX.** which calculates the maximum of the field values.
- **FST.** which retrieves the first value of the field.
- **LST.** which retrieves the last value of the field.

field

Numeric

Is the field to be used in the calculation.

offset

Numeric

Is a negative number indicating the number of rows back from the current row to use for the calculation.

Calculating the Increase Between the Current and a Prior Value of a Field

The following request uses the default value of SET PARTITION_ON (PENULTIMATE) to calculate the increase within the PRODUCT_CATEGORY sort field between the current row and the previous row.

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE INC = INCREASE(QUANTITY_SOLD,-1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value for INC is the value in the Accessories category for Quantity Sold, as there is no prior value. The second value for INC is the difference between the values for Headphones and Charger, the third is the difference between Universal Remote Controls and Headphones. Then, the calculations start over for Camcorder, which is the reset point.

Product Category	Product Subcategory	Quantity Sold	INC
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	123,092.00
	Universal Remote Controls	178,061	-50,288.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	-237,295.00
	Standard	192,205	179,333.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	-58,321.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	-660,660.00
	DVD Players - Portable	5,694	-13,141.00
	Streaming	67,910	62,216.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	389,722.00
	Receivers	150,568	-248,524.00
	Speaker Kits	244,199	93,631.00
	iPod Docking Station	311,103	66,904.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	87,863.00
	Portable TV	8,049	-84,452.00
Video Production	Video Editing	199,749	199,749.00

PCT_INCREASE: Calculating the Percentage Difference Between the Current and a Prior Value of a Field

Given an aggregated input field and a negative offset, PCT_INCREASE calculates the percentage difference between the value in the current row of the report output and one or more prior rows, within a sort break or the entire table. The reset point for the calculation is determined by the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

The percentage increase is calculated using the following formula:

```
(current_value - prior_value) / prior_value
```

Note: The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

Calculate the Percentage Difference Between the Current and a Prior Value of a Field

```
PCT_INCREASE([prefix.]field, offset)
```

where:

prefix

Is one of the following optional aggregation operators to apply to the field before using it in the calculation:

- **SUM.** which calculates the sum of the field values. SUM is the default value.
- **CNT.** which calculates a count of the field values.
- **AVE.** which calculates the average of the field values.
- **MIN.** which calculates the minimum of the field values.
- **MAX.** which calculates the maximum of the field values.
- **FST.** which retrieves the first value of the field.
- **LST.** which retrieves the last value of the field.

field

Numeric

The field to be used in the calculation.

offset

Numeric

Is a negative number indicating the number of rows back from the current row to use for the calculation.

PCT_INCREASE: Calculating the Percent Increase Between the Current and a Prior Value of a Field

The following request uses the default value of SET PARTITION_ON (PENULTIMATE) to calculate the percent increase within the PRODUCT_CATEGORY sort field between the current row and the previous row.

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE PCTINC/D8.2p = PCT_INCREASE(QUANTITY_SOLD,-1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value for PCTINC is zero percent, as there is no prior value. The second value for PCTINC is the percent difference between the values for Headphones and Charger, the third is the percent difference between Universal Remote Controls and Headphones. Then, the calculations start over for Camcorder, which is the reset point.

Product Category	Product Subcategory	Quantity Sold	PCTINC
Accessories	Charger	105,257	.00%
	Headphones	228,349	116.94%
	Universal Remote Controls	178,061	-22.02%
Camcorder	Handheld	250,167	.00%
	Professional	12,872	-94.85%
	Standard	192,205	1,393.20%
Computers	Smartphone	205,049	.00%
	Tablet	146,728	-28.44%
Media Player	Blu Ray	679,495	.00%
	DVD Players	18,835	-97.23%
	DVD Players - Portable	5,694	-69.77%
	Streaming	67,910	1,092.66%
Stereo Systems	Boom Box	9,370	.00%
	Home Theater Systems	399,092	4,159.25%
	Receivers	150,568	-62.27%
	Speaker Kits	244,199	62.19%
	iPod Docking Station	311,103	27.40%
Televisions	CRT TV	4,638	.00%
	Flat Panel TV	92,501	1,894.42%
	Portable TV	8,049	-91.30%
Video Production	Video Editing	199,749	.00%

PREVIOUS: Retrieving a Prior Value of a Field

Given an aggregated input field and a negative offset, PREVIOUS retrieves the value in a prior row, within a sort break or the entire table. The reset point for the calculation is determined by the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

Note: The values used in the retrieval depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

Retrieve a Prior Value of a Field

```
PREVIOUS([prefix.]field, offset)
```

where:

prefix

Is one of the following optional aggregation operators to apply to the field before using it in the calculation:

- **SUM.** which calculates the sum of the field values. SUM is the default value.
- **CNT.** which calculates a count of the field values.
- **AVE.** which calculates the average of the field values.
- **MIN.** which calculates the minimum of the field values.
- **MAX.** which calculates the maximum of the field values.
- **FST.** which retrieves the first value of the field.
- **LST.** which retrieves the last value of the field.

field

Numeric or an alphanumeric field that contains all numeric digits.

The field to be used in the calculation.

offset

Numeric

Is a negative number indicating the number of rows back from the current row to use for the retrieval.

Retrieving a Prior Value of a Field

The following request sets the PARITITON_ON parameter to TABLE and retrieves the value of the QUANTITIY_SOLD field two rows back from the current row.

```
SET PARTITION_ON=TABLE
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE PREV = PREVIOUS(QUANTITY_SOLD,-2);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The value of PREV in the first two rows is zero, as there are no prior rows for retrieval. From then on, each value of PREV is from the QUANTITY_SOLD value from two rows prior, with no reset points.

Product Category	Product Subcategory	Quantity Sold	PREV
Accessories	Charger	105,257	.00
	Headphones	228,349	.00
	Universal Remote Controls	178,061	105,257.00
Camcorder	Handheld	250,167	228,349.00
	Professional	12,872	178,061.00
	Standard	192,205	250,167.00
Computers	Smartphone	205,049	12,872.00
	Tablet	146,728	192,205.00
Media Player	Blu Ray	679,495	205,049.00
	DVD Players	18,835	146,728.00
	DVD Players - Portable	5,694	679,495.00
	Streaming	67,910	18,835.00
Stereo Systems	Boom Box	9,370	5,694.00
	Home Theater Systems	399,092	67,910.00
	Receivers	150,568	9,370.00
	Speaker Kits	244,199	399,092.00
	iPod Docking Station	311,103	150,568.00
Televisions	CRT TV	4,638	244,199.00
	Flat Panel TV	92,501	311,103.00
	Portable TV	8,049	4,638.00
Video Production	Video Editing	199,749	92,501.00

RUNNING_AVE: Calculating an Average Over a Group of Rows

Given an aggregated input field and a negative offset, RUNNING_AVE calculates the average of the values between the current row of the report output and one or more prior rows, within a sort break or the entire table. The reset point for the calculation is determined by the sort field specified, the entire table, or the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

Calculate Running Average Between the Current and a Prior Value of a Field

```
RUNNING_AVE(field, reset_key, lower)
```

where:

field

Numeric

The field to be used in the calculation.

reset_key

Identifies the point at which the running average restarts. Valid values are:

- The name of a sort field in the request.
- PRESET, which uses the value of the PARTITION_ON parameter, as described in Specify the Partition Size for Simplified Statistical Functions.
- TABLE, which indicates that there is no break on a sort field.

> **Note:** The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

lower

Is the starting point in the partition for the running average. Valid values are:

- A negative number, which identifies the offset from the current row.
- B, which specifies the beginning of the sort group.

Calculating a Running Average

The following request calculates a running average of QUANTITY_SOLD within the PRODUCT_CATEGORY sort field, always starting from the beginning of the sort break.

```
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE RAVE = RUNNING_AVE(QUANTITY_SOLD,PRODUCT_CATEGORY,B);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value for RAVE is the value in the Accessories category for Quantity Sold, as there is no prior value. The second value for RAVE is the average of the values for Headphones and Charger, the third is the average of the values for Headphones, Charger, and Universal Remote Controls. Then, the calculations start over for Camcorder, which is the reset point.

Product Category	Product Subcategory	Quantity Sold	RAVE
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	166,803.00
	Universal Remote Controls	178,061	170,555.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	131,519.00
	Standard	192,205	151,748.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	175,888.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	349,165.00
	DVD Players - Portable	5,694	234,674.00
	Streaming	67,910	192,983.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	204,231.00
	Receivers	150,568	186,343.00
	Speaker Kits	244,199	200,807.00
	iPod Docking Station	311,103	222,866.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	48,569.00
	Portable TV	8,049	35,062.00
Video Production	Video Editing	199,749	199,749.00

RUNNING_MAX: Calculating a Maximum Over a Group of Rows

Given an aggregated input field and an offset, RUNNING_MAX calculates the maximum of the values between the current row of the report output and one or more prior rows, within a sort break or the entire table. The reset point for the calculation is determined by the sort field specified, the entire table, or the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

Calculate Running Maximum Between the Current and a Prior Value of a Field

```
RUNNING_MAX(field, reset_key, lower)
```

where:

field

Numeric or an alphanumeric field that contains all numeric digits.

The field to be used in the calculation.

reset_key

Identifies the point at which the running maximum restarts. Valid values are:

- The name of a sort field in the request.
- PRESET, which uses the value of the PARTITION_ON parameter, as described in Specify the Partition Size for Simplified Statistical Functions.
- TABLE, which indicates that there is no break on a sort field.

> **Note:** The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

lower

Is the starting point in the partition for the running maximum. Valid values are:

- A negative number, which identifies the offset from the current row.
- B, which specifies the beginning of the sort group.

Calculating a Running Maximum

The following request calculates a running maximum for the rows from the beginning of the table to the current value of QUANTITY_SOLD, with no reset point.

```
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE RMAX = RUNNING_MAX(QUANTITY_SOLD,TABLE,B);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value for RMAX is the value in the Accessories category for Quantity Sold, as there is no prior value. The second value for RMAX is the value for Headphones, as that value is larger. The third value for RMAX is still the value for Headphones, as that value is larger than the Quantity Sold value in the third row. Since the maximum value in the table occurs for Blu Ray, that value is repeated on all future rows, as there is no reset point.

Product Category	Product Subcategory	Quantity Sold	RMAX
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	228,349.00
	Universal Remote Controls	178,061	228,349.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	250,167.00
	Standard	192,205	250,167.00
Computers	Smartphone	205,049	250,167.00
	Tablet	146,728	250,167.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	679,495.00
	DVD Players - Portable	5,694	679,495.00
	Streaming	67,910	679,495.00
Stereo Systems	Boom Box	9,370	679,495.00
	Home Theater Systems	399,092	679,495.00
	Receivers	150,568	679,495.00
	Speaker Kits	244,199	679,495.00
	iPod Docking Station	311,103	679,495.00
Televisions	CRT TV	4,638	679,495.00
	Flat Panel TV	92,501	679,495.00
	Portable TV	8,049	679,495.00
Video Production	Video Editing	199,749	679,495.00

RUNNING_MIN: Calculating a Minimum Over a Group of Rows

Given an aggregated input field and an offset, RUNNING_MIN calculates the minimum of the values between the current row of the report output and one or more prior rows, within a sort break or the entire table. The reset point for the calculation is determined by the sort field specified, the entire table, or the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

Calculate Running Minimum Between the Current and a Prior Value of a Field

```
RUNNING_MIN(field, reset_key, lower)
```

where:

field

Numeric or an alphanumeric field that contains all numeric digits.

The field to be used in the calculation.

reset_key

Identifies the point at which the running minimum restarts. Valid values are:

- The name of a sort field in the request.
- PRESET, which uses the value of the PARTITION_ON parameter, as described in Specify the Partition Size for Simplified Statistical Functions.
- TABLE, which indicates that there is no break on a sort field.

> **Note:** The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

lower

Is the starting point in the partition for the running minimum. Valid values are:

- A negative number, which identifies the offset from the current row.
- B, which specifies the beginning of the sort group.

Calculating a Running Minimum

The following request calculates a running minimum of QUANTITY_SOLD within the PRODUCT_CATEGORY sort field (the sort break defined by SET PARTITION_ON =

PENULTIMATE), always starting from the beginning of the sort break.

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE RMIN = RUNNING_MIN(QUANTITY_SOLD,PRESET,B);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value for RMIN is the value in the Accessories category for Quantity Sold, as there is no prior value. The second value for RMIN is the value from the first row again (Charger), as that is smaller than the value in the second row. The third is the same again, as it is still the smallest. Then, the calculations start over for Camcorder, which is the reset point.

Product Category	Product Subcategory	Quantity Sold	RMIN
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	105,257.00
	Universal Remote Controls	178,061	105,257.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	12,872.00
	Standard	192,205	12,872.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	146,728.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	18,835.00
	DVD Players - Portable	5,694	5,694.00
	Streaming	67,910	5,694.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	9,370.00
	Receivers	150,568	9,370.00
	Speaker Kits	244,199	9,370.00
	iPod Docking Station	311,103	9,370.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	4,638.00
	Portable TV	8,049	4,638.00
Video Production	Video Editing	199,749	199,749.00

RUNNING_SUM: Calculating a Sum Over a Group of Rows

Given an aggregated input field and an offset, RUNNING_SUM calculates the sum of the values between the current row of the report output and one or more prior rows, within a sort break or the entire table. The reset point for the calculation is determined by the sort field specified, the entire table, or the value of the PARTITION_ON parameter described in Specify the Partition Size for Simplified Statistical Functions.

Calculate Running Sum Between the Current and a Prior Value of a Field

```
RUNNING_SUM(field, reset_key, lower)
```

where:

field

Numeric

The field to be used in the calculation.

reset_key

Identifies the point at which the running sum restarts. Valid values are:

- The name of a sort field in the request.
- PRESET, which uses the value of the PARTITION_ON parameter, as described in Specify the Partition Size for Simplified Statistical Functions.
- TABLE, which indicates that there is no break on a sort field.

> **Note:** The values used in the calculations depend on the sort sequence (ascending or descending) specified in the request. Be aware that displaying a date or time dimension in descending order may produce different results than those you may expect.

lower

Is the starting point in the partition for the running sum. Valid values are:

- A negative number, which identifies the offset from the current row.
- B, which specifies the beginning of the sort group.

Calculating a Running Sum

The following request calculates a running sum of the current value and previous value of QUANTITY_SOLD within the reset point set by the PARTITION_ON parameter, which is the sort field PRODUCT_CATEGORY.

```
SET PARTITION_ON=PENULTIMATE
TABLE FILE wflite
SUM QUANTITY_SOLD
COMPUTE RSUM = RUNNING_SUM(QUANTITY_SOLD,PRESET,-1);
BY PRODUCT_CATEGORY
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The first value for RSUM is the value in the Accessories category for Quantity Sold, as there is no prior value. The second value for RSUM is the sum of the values for Headphones and Charger, the third is the sum of the values for Headphones and Universal Remote Controls. Then, the calculations start over for Camcorder, which is the reset point.

Product Category	Product Subcategory	Quantity Sold	RSUM
Accessories	Charger	105,257	105,257.00
	Headphones	228,349	333,606.00
	Universal Remote Controls	178,061	406,410.00
Camcorder	Handheld	250,167	250,167.00
	Professional	12,872	263,039.00
	Standard	192,205	205,077.00
Computers	Smartphone	205,049	205,049.00
	Tablet	146,728	351,777.00
Media Player	Blu Ray	679,495	679,495.00
	DVD Players	18,835	698,330.00
	DVD Players - Portable	5,694	24,529.00
	Streaming	67,910	73,604.00
Stereo Systems	Boom Box	9,370	9,370.00
	Home Theater Systems	399,092	408,462.00
	Receivers	150,568	549,660.00
	Speaker Kits	244,199	394,767.00
	iPod Docking Station	311,103	555,302.00
Televisions	CRT TV	4,638	4,638.00
	Flat Panel TV	92,501	97,139.00
	Portable TV	8,049	100,550.00
Video Production	Video Editing	199,749	199,749.00

Simplified Character Functions

Simplified character functions have streamlined parameter lists, similar to those used by SQL functions. In some cases, these simplified functions provide slightly different functionality than previous versions of similar functions.

The simplified functions do not have an output argument. Each function returns a value that has a specific data type.

When used in a request against a relational data source, these functions are optimized (passed to the RDBMS for processing).

CHAR_LENGTH: Returning the Length in Characters of a String

The CHAR_LENGTH function returns the length, in characters, of a string. In Unicode environments, this function uses character semantics, so that the length in characters may not be the same as the length in bytes. If the string includes trailing blanks, these are counted in the returned length. Therefore, if the format source string is type A**n**, the returned value will always be **n**.

Return the Length of a String in Characters

```
CHAR_LENGTH(string)
```

where:

string

Alphanumeric

Is the string whose length is returned.

The data type of the returned length value is Integer.

Returning the Length of a String

The following request against the EMPLOYEE data source creates a virtual field named LASTNAME of type A15V that contains the LAST_NAME with the trailing blanks removed. It then uses CHAR_LENGTH to return the number of characters.

```
DEFINE FILE EMPLOYEE
LASTNAME/A15V = RTRIM(LAST_NAME);
END
TABLE FILE EMPLOYEE
SUM LAST_NAME NOPRINT AND COMPUTE
NAME_LEN/I3 = CHAR_LENGTH(LASTNAME);
BY LAST_NAME
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  LAST_NAME        NAME_LEN
  ---------       --------
 BANNING                 7
 BLACKWOOD               9
 CROSS                   5
 GREENSPAN               9
 IRVING                  6
 JONES                   5
 MCCOY                   5
 MCKNIGHT                8
 ROMANS                  6
 SMITH                   5
 STEVENS                 7
```

CONCAT: Concatenating Strings

CONCAT concatenates two strings. The output is returned as a variable-length alphanumeric.

Concatenate Strings

```
CONCAT(string1, string2)
```

where:

string2

Alphanumeric

Is a string to be concatenated.

string1

Alphanumeric

Is a string to be concatenated.

Concatenating Strings

The following request concatenates city names with state names. Note that the city and state names are converted to fixed length alphanumeric fields before concatenation.

```
DEFINE FILE WFLITE
CITY/A50 = CITY_NAME;
STATE/A50 = STATE_PROV_NAME;
CONCAT_CS/A100 = CONCAT(CITY,STATE);
END

TABLE FILE WFLITE
SUM CITY AS City STATE AS State CONCAT_CS AS Concatenation
BY STATE_PROV_NAME NOPRINT
WHERE COUNTRY_NAME EQ 'United States'
WHERE STATE LE 'Louisiana'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

City	State	Concatenation
Montgomery	Alabama	Montgomery Alabama
Anchorage	Alaska	Anchorage Alaska
Phoenix	Arizona	Phoenix Arizona
Little Rock	Arkansas	Little Rock Arkansas
Saratoga	California	Saratoga California
Colorado Springs	Colorado	Colorado Springs Colorado
Old Lyme	Connecticut	Old Lyme Connecticut
Wyoming	Delaware	Wyoming Delaware
Washington	District of Columbia	Washington District of Columbia
Orlando	Florida	Orlando Florida
Atlanta	Georgia	Atlanta Georgia
Honolulu	Hawaii	Honolulu Hawaii
Boise	Idaho	Boise Idaho
Chicago	Illinois	Chicago Illinois
Indianapolis	Indiana	Indianapolis Indiana
Dubuque	Iowa	Dubuque Iowa
Wichita	Kansas	Wichita Kansas
Lexington	Kentucky	Lexington Kentucky
New Orleans	Louisiana	New Orleans Louisiana

DIFFERENCE: Measuring the Phonetic Similarity Between Character Strings

DIFFERENCE returns an integer value measuring the difference between the SOUNDEX or METAPHONE values of two character expressions.

Measure the Phonetic Similarity Between Character String

```
DIFFERENCE(chrexp1, chrexp2)
```

where:

chrexp1, chrexp2

Alphanumeric

Are the character strings to be compared.

Zero (0) represents the least similarity. For SOUNDEX, 4 represents the most similarity, and for METAPHONE, 16 represents the most similarity.

The use of SOUNDEX or METAPHONE depends on the PHONETIC_ALGORITHM setting. METAPHONE is the default algorithm.

Measuring the Phonetic Similarity Between Character Strings

The following request uses DIFFERENCE with the default phonetic algorithm (METAPHONE) to compare first names in the data source with the names JOHN and MARY.

```
TABLE FILE VIDEOTRK
PRINT FIRSTNAME
COMPUTE
JOHN_DIFF/I5 = DIFFERENCE(FIRSTNAME,'JOHN') ;
MARY_DIFF/I5 = DIFFERENCE(FIRSTNAME, 'MARY');
BY LASTNAME NOPRINT
WHERE RECORDLIMIT EQ 30
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. Note that the names JOANN and JOHN have the highest scores for matching with JOHN, and that MARCIA, MICHAEL, and MARTHA have the highest scores for matching with MARY.

FIRSTNAME	JOHN_DIFF	MARY_DIFF
NATALIA	3	5
MARCIA	3	10
IVY	0	0
JASON	6	6
JANET	10	6
JOANN	16	4
JOHN	16	4
WESTON	0	3
GEORGIA	6	6
EVAN	0	0
JESSICA	5	5
MICHAEL	3	10
JAMES	6	6
CHERYL	3	10
DAVID	3	6
JOSHUA	8	8
JOHN	16	4
CATHERINE	2	4
PATRICK	3	3
DONALD	5	5
GLENDA	0	0
RICHARD	3	5
MICHAEL	3	10
LESLIE	3	3
TOM	5	4
MICHAEL	3	10
PATRICIA	2	2
KENNETH	6	6
KELLY	4	8
MARTHA	3	10

DIGITS: Converting a Number to a Character String

Given a number, DIGITS converts it to a character string of the specified length. The format of the field that contains the number must be Integer.

Convert a Number to a Character String

```
DIGITS(number,length)
```

where:

number

Integer

Is the number to be converted, stored in a field with the data type Integer.

length

Integer between 1 and 10

Is the length of the returned character string. If the *length* is longer than the number of digits in the number being converted, the returned value is padded on the left with zeros. If the *length* is shorter than the number of digits in the number being converted, the returned value is truncated on the left.

Converting a Number to a Character String

The following request against the WFLITE data source converts -123.45 and ID_PRODUCT to character strings:

```
DEFINE FILE WFLITE
MEAS1/I8=-123.45;
DIG1/A6=DIGITS(MEAS1,6) ;
DIG2/A6=DIGITS(ID_PRODUCT,6) ;
END
TABLE FILE WFLITE
PRINT MEAS1 DIG1
ID_PRODUCT DIG2
BY PRODUCT_SUBCATEG
WHERE PRODUCT_SUBCATEG EQ 'Flat Panel TV'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

Product Subcategory	MEAS1	DIG1	ID Product	DIG2
Flat Panel TV	-123	000123	4012	004012
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4014	004014
	-123	000123	4016	004016
	-123	000123	4016	004016
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4016	004016
	-123	000123	4018	004018
	-123	000123	4016	004016
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4018	004018
	-123	000123	4017	004017
	-123	000123	4017	004017
	-123	000123	4014	004014
	-123	000123	4018	004018

Usage Notes for DIGITS

- Only I format numbers will be converted. D, P, and F formats generate error messages and should be converted to I before using the DIGITS function. The limit for the number that can be converted is 2 GB.
- Negative integers are turned into positive integers.

- Integer formats with decimal places are truncated.
- DIGITS is not supported in Dialogue Manager.

GET_TOKEN: Extracting a Token Based on a String of Delimiters

GET_TOKEN extracts a token (substring) based on a string that can contain multiple characters, each of which represents a single-character delimiter.

Extract a Token Based on a String of Delimiters

```
GET_TOKEN(string, delimiter_string, occurrence)
```

where:

string

Alphanumeric

Is the input string from which the token will be extracted. This can be an alphanumeric field or constant.

delimiter_string

Alphanumeric constant

Is a string that contains the list of delimiter characters. For example, '; ,' contains three delimiter characters, semi-colon, blank space, and comma.

occurrence

Integer constant

Is a positive integer that specifies the token to be extracted. A negative integer will be accepted in the syntax, but will not extract a token. The value zero (0) is not supported.

Extracting a Token Based on a String of Delimiters

The following request defines an input string and two tokens based on a list of delimiters that contains the characters comma (,), semicolon (;), and slash (/).

```
DEFINE FILE EMPLOYEE
InputString/A20 = 'ABC,DEF;GHI/JKL';
FirstToken/A20 WITH DEPARTMENT = GET_TOKEN(InputString, ',;/', 1);
FourthToken/A20 WITH DEPARTMENT = GET_TOKEN(InputString, ',;/', 4);
END
TABLE FILE EMPLOYEE
PRINT InputString FirstToken FourthToken
WHERE READLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID = OFF,$
END
```

The output is shown in the following image. The first token was extracted using the comma (,) as the delimiter. The fourth token was extracted using the slash (/) as the delimiter.

InputString	FirstToken	FourthToken
ABC,DEF;GHI/JKL	ABC	JKL

INITCAP: Capitalizing the First Letter of Each Word in a String

INITCAP capitalizes the first letter of each word in an input string and makes all other letters lowercase. A word starts at the beginning of the string, after a blank space, or after a special character.

Capitalize the First Letter of Each Word in a String

```
INITCAP(input_string)
```

where:

input_string

Alphanumeric

Is the string to capitalize.

Capitalizing the First Letter of Each Word in a String

The following request changes the last names in the EMPLOYEE data source to initial caps and capitalizes the first letter after each blank or special character in the NewName field.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
Caps1/A30 = INITCAP(LAST_NAME);
NewName/A30 = 'abc,def!ghi'jKL MNO';
Caps2/A30 = INITCAP(NewName);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

LAST_NAME	Caps1	NewName	Caps2
STEVENS	Stevens	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
SMITH	Smith	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
JONES	Jones	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
SMITH	Smith	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
BANNING	Banning	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
IRVING	Irving	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
ROMANS	Romans	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
MCCOY	Mccoy	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
BLACKWOOD	Blackwood	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
MCKNIGHT	Mcknight	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
GREENSPAN	Greenspan	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno
CROSS	Cross	abc,def!ghi'jKL MNO	Abc,Def!Ghi'Jkl Mno

LAST_NONBLANK: Retrieving the Last Field Value That is Neither Blank nor Missing

LAST_NONBLANK retrieves the last field value that is neither blank nor missing. If all previous values are either blank or missing, LAST_NONBLANK returns a missing value.

Return the Last Value That is Neither Blank nor Missing

```
LAST_NONBLANK(field)
```

where:

field

Is the field name whose last non-blank value is to be retrieved. If the current value is not blank or missing, the current value is returned.

Note: LAST_NONBLANK cannot be used in a compound expression, for example, as part of an IF condition.

Retrieving the Last Non-Blank Value

Consider the following delimited file named input1.csv that has two fields named FIELD_1 and FIELD_2.

```
,
A,
,
 ,
B,
C,
```

The input1 Master File follows.

```
FILENAME=INPUT1, SUFFIX=DFIX    ,
 DATASET=baseapp/input1.csv(LRECL 15 RECFM V, BV_NAMESPACE=OFF, $
  SEGMENT=INPUT1, SEGTYPE=S0, $
    FIELDNAME=FIELD_1, ALIAS=E01, USAGE=A1V, ACTUAL=A1V,
      MISSING=ON, $
    FIELDNAME=FIELD_2, ALIAS=E02, USAGE=A1V, ACTUAL=A1V,
      MISSING=ON, $
```

The input1 Access File follows.

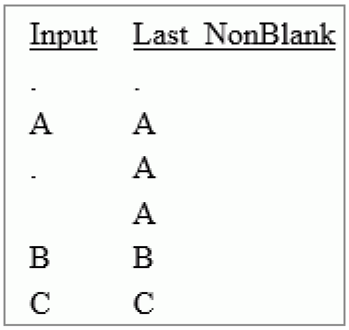
```
SEGNAME=INPUT1,
  DELIMITER=',',
  HEADER=NO,
  PRESERVESPACE=NO,
  CDN=COMMAS_DOT,
  CONNECTION=<local>, $
```

The following request displays the FIELD_1 values and computes the last non-blank value for each FIELD_1 value.

```
TABLE FILE baseapp/INPUT1
PRINT FIELD_1 AS Input
COMPUTE
```

```
Last_NonBlank/A1 MISSING ON = LAST_NONBLANK(FIELD_1);
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.



Input	Last_NonBlank
.	.
A	A
.	A
	A
B	B
C	C

LEFT: Returning Characters From the Left of a Character String

Given a source character string, or an expression that can be converted to varchar (variable-length alphanumeric), and an integer number, LEFT returns that number of characters from the left end of the string.

Return Characters From the Left of a Character String

```
LEFT(chr_exp, int_exp)
```

where:

chr_exp

Alphanumeric or an expression that can be converted to variable-length alphanumeric.

Is the source character string.

int_exp

Integer

Is the number of characters to be returned.

Returning Characters From the Left of a Character String

The following request computes the length of the first name in the FULLNAME field and returns that number of characters to FIRST.

```
TABLE FILE WF_RETAIL_EMPLOYEE
PRINT FULLNAME AND
COMPUTE LEN/I5 = ARGLEN(54, GET_TOKEN(FULLNAME, ' ', 1), LEN); NOPRINT
COMPUTE FIRST/A20 = LEFT(FULLNAME, LEN);
WHERE RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Full Name	FIRST
Steven Wagoner	Steven
Adan Geoghegan	Adan
Candace Aguilar	Candace
Dianna Turpin	Dianna
John Blankinship	John
John Chang	John
John Mackey	John
Elaine Duran	Elaine
Douglas Sanders	Douglas
Linda Whitlow	Linda
Phyllis Carey	Phyllis
Alfred Amerson	Alfred
Jeremy Maness	Jeremy
David Christopher	David
Alice Flemming	Alice
Delia Tennison	Delia
Diane Eads	Diane
Wilfredo Delacruz	Wilfredo
Dorothy Newman	Dorothy
Delia Tennison	Delia

LOWER: Returning a String With All Letters Lowercase

The LOWER function takes a source string and returns a string of the same data type with all letters translated to lowercase.

Return a String With All Letters Lowercase

```
LOWER(string)
```

where:

string

Alphanumeric

Is the string to convert to lowercase.

The returned string is the same data type and length as the source string.

Converting a String to Lowercase

In the following request against the EMPLOYEE data source, LOWER 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 = LOWER(LAST_NAME);
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  LAST_NAME        LOWER_NAME
  ---------        ----------
  STEVENS          stevens
  SMITH            smith
  JONES            jones
  SMITH            smith
  BANNING          banning
  IRVING           irving
  ROMANS           romans
  MCCOY            mccoy
  BLACKWOOD        blackwood
  MCKNIGHT         mcknight
  GREENSPAN        greenspan
  CROSS            cross
```

LPAD: Left-Padding a Character String

LPAD uses a specified character and output length to return a character string padded on the left with that character.

Pad a Character String on the Left

```
LPAD(string, out_length, pad_character)
```

where:

string

Fixed length alphanumeric

Is a string to pad on the left side.

out_length

Integer

Is the length of the output string after padding.

pad_character

Fixed length alphanumeric

Is a single character to use for padding.

Left-Padding a String

In the following request against the WF_RETAIL data source, LPAD left-pads the PRODUCT_CATEGORY column with @ symbols:

```
DEFINE FILE WFLITE
LPAD1/A25 = LPAD(PRODUCT_CATEGORY,25,'@');
DIG1/A4 = DIGITS(ID_PRODUCT,4);
END
TABLE FILE WFLITE
SUM DIG1 LPAD1
```

```
BY PRODUCT_CATEGORY
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
TYPE=DATA,FONT=COURIER,SIZE=11,COLOR=BLUE,$
END
```

The output is:

Product Category	DIG1	LPAD1
Accessories	5005	@@@@@@@@@@@@@@Accessories
Camcorder	3006	@@@@@@@@@@@@@@@@Camcorder
Computers	6016	@@@@@@@@@@@@@@@@Computers
Media Player	1003	@@@@@@@@@@@@@Media Player
Stereo Systems	2155	@@@@@@@@@@@Stereo Systems
Televisions	4018	@@@@@@@@@@@@@@Televisions
Video Production	7005	@@@@@@@@@Video Production

Usage Notes for LPAD

- To use the single quotation mark (') as the padding character, you must double it and enclose the two single quotation marks within single quotation marks (LPAD (COUNTRY, 20,'''')). You can use an amper variable in quotation marks for this parameter, but you cannot use a field, virtual or real.
- Input can be fixed or variable length alphanumeric.
- Output, when optimized to SQL, will always be data type VARCHAR.
- If the output is specified as shorter than the original input, the original data will be truncated, leaving only the padding characters. The output length can be specified as a positive integer or an unquoted &variable (indicating a numeric).

LTRIM: Removing Blanks From the Left End of a String

The LTRIM function removes all blanks from the left end of a string.

Remove Blanks From the Left End of a String

```
LTRIM(string)
```

where:

string

Alphanumeric

Is the string to trim on the left.

The data type of the returned string is AnV, with the same maximum length as the source string.

Removing Blanks From the Left End of a String

In the following request against the MOVIES data source, the DIRECTOR field is right-justified and stored in the RDIRECTOR virtual field. Then LTRIM removes the leading blanks from the RDIRECTOR field:

```
DEFINE FILE MOVIES
RDIRECTOR/A17 = RJUST(17, DIRECTOR, 'A17');
 END
TABLE FILE MOVIES
PRINT RDIRECTOR AND
COMPUTE
TRIMDIR/A17 = LTRIM(RDIRECTOR);
WHERE DIRECTOR CONTAINS 'BR'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
RDIRECTOR          TRIMDIR
---------          -------
      ABRAHAMS J.  ABRAHAMS J.
        BROOKS R.  BROOKS R.
      BROOKS J.L.  BROOKS J.L.
```

OVERLAY: Replacing Characters in a String

Given a starting position, length, source string, and insertion string, OVERLAY replaces the number of characters defined by *length* in the source string with the insertion string, starting from the starting position.

Replace Characters in a String

```
OVERLAY(src, ins, start, len)
```

where:

src

Alphanumeric

Is the source string whose characters will be replaced.

ins

Alphanumeric

Is the insertion string with the replacement characters.

start

Numeric

Is the starting position for the replacement in the source string.

len

Numeric

Is the number of characters to replace in the source string with the entire insertion string.

Replacing Characters in a String

The following request replaces the first three characters in the last name with the first four characters of the first name.

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE FIRST4/A4 = LEFT(FIRST_NAME,4);
NEWNAME/A20 = OVERLAY(LAST_NAME, FIRST4, 1, 3);
BY LAST_NAME
BY FIRST_NAME
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

LAST_NAME	FIRST_NAME	FIRST4	NEWNAME
BANNING	JOHN	JOHN	JOHNNING
BLACKWOOD	ROSEMARIE	ROSE	ROSECKWOOD
CROSS	BARBARA	BARB	BARBSS
GREENSPAN	MARY	MARY	MARYENSPAN
IRVING	JOAN	JOAN	JOANING
JONES	DIANE	DIAN	DIANES
MCCOY	JOHN	JOHN	JOHNOY
MCKNIGHT	ROGER	ROGE	ROGENIGHT
ROMANS	ANTHONY	ANTH	ANTHANS
SMITH	MARY	MARY	MARYTH
	RICHARD	RICH	RICHTH
STEVENS	ALFRED	ALFR	ALFRVENS

PATTERNS: Returning a Pattern That Represents the Structure of the Input String

PATTERNS returns a string that represents the structure of the input argument. The returned pattern includes the following characters:

- **A** is returned for any position in the input string that has an uppercase letter.
- **a** is returned for any position in the input string that has a lowercase letter.
- **9** is returned for any position in the input string that has a digit.

Note that special characters (for example, +-/=%) are returned exactly as they were in the input string.

The output is returned as a variable length alphanumeric.

Return a String That Represents the Pattern Profile of the Input Argument

```
PATTERNS(string)
```

where:

string

Alphanumeric

Is a string whose pattern will be returned.

Returning a Pattern Representing an Input String

The following request returns patterns that represent customer addresses.

```
DEFINE FILE WFLITE
Address_Pattern/A40V = PATTERNS(ADDRESS_LINE_1);
END
```

```
TABLE FILE WFLITE
PRINT FST.ADDRESS_LINE_1 OVER
Address_Pattern
BY ADDRESS_LINE_1 NOPRINT SKIP-LINE
WHERE COUNTRY_NAME EQ 'United States'
WHERE CITY_NAME EQ 'Houston' OR 'Indianapolis' OR 'Chapel Hill' OR
'Bronx'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The partial output is shown in the following image. Note that the special characters (#-,) in an address are represented in the pattern as is.

FST Customer Address Line 1	1010 Milam St # Ifp-2352
Address_Pattern	9999 Aaaaa Aa # Aaa-9999
FST Customer Address Line 1	10700 Richmond Ave
Address_Pattern	99999 Aaaaaaaa Aaa
FST Customer Address Line 1	10777 North Fwy
Address_Pattern	99999 Aaaaa Aaa
FST Customer Address Line 1	11 E Greenway Plz Ste 100
Address_Pattern	99 A Aaaaaaaa Aaa Aaa 999
FST Customer Address Line 1	111 Monument Cir
Address_Pattern	999 Aaaaaaaa Aaa
FST Customer Address Line 1	111 Monument Circle - Ste 2100
Address_Pattern	999 Aaaaaaaa Aaaaaa - Aaa 9999
FST Customer Address Line 1	1205 Dart St, Rm 219
Address_Pattern	9999 Aaaa Aa, Aa 999

POSITION: Returning the First Position of a Substring in a Source String

The POSITION function returns the first position (in characters) of a substring in a source string.

Return the First Position of a Substring in a Source String

```
POSITION(pattern, string)
```

where:

pattern

Alphanumeric

Is the substring whose position you want to locate. The string can be as short as a single character, including a single blank.

string

Alphanumeric

Is the string in which to find the pattern.

The data type of the returned value is Integer.

Returning the First Position of a Substring

In the following request against the EMPLOYEE data source, POSITION determines the position of the first capital letter I in LAST_NAME and stores the result in I_IN_NAME:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
I_IN_NAME/I2 = POSITION('I', LAST_NAME);
```

```
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
LAST_NAME        I_IN_NAME
---------        ---------
STEVENS                  0
SMITH                    3
JONES                    0
SMITH                    3
BANNING                  5
IRVING                   1
ROMANS                   0
MCCOY                    0
BLACKWOOD                0
MCKNIGHT                 5
GREENSPAN                0
CROSS                    0
```

Regular Expression Functions

A regular expression is a sequence of special meta-characters and literal characters that you can combine to form a search pattern.

Note: You can search online for information about the symbols used to create a regular expression pattern. For example, Wikipedia has a good introduction at:

https://en.wikipedia.org/wiki/Regular_expression

The following list summarizes common meta-characters used in regular expressions.

- . represents any single character
- * represents zero or more occurrences
- + represents one or more occurrences
- ? represents zero or one occurrence
- ^ represents beginning of line

- $ represents end of line
- [] represents any one character in the set listed within the brackets
- [^] represents any one character not in the set listed within the brackets
- | represents the Or operator
- \ is the Escape Special Character
- () contains a character sequence

For example, the regular expression '^Ste(v|ph)en$' matches values starting with *Ste* followed by either *ph* or *v*, and ending with *en*.

Using Regular Expressions on z/OS

On z/OS, depending on the code page you are using, some of the meta-characters used to create a regular expression may not be interpreted correctly when inserted directly from a Windows keyboard.

If you are using the Unicode code page 65002, the meta-characters will be interpreted correctly. In this environment, you need to be sure the files you are referencing, such as FOCUS data sources, have been built using this code page.

If you are not using a Unicode code page, you can use the CHAR function to return the correct meta-characters, based on the decimal code for the EBCDIC character. For example, to insert:

- The circumflex (^) meta-character, use CHAR(95).
- The left bracket ([) meta-character, use CHAR(173).
- The right bracket (]) meta-character, use CHAR(189).
- The left brace ({) meta-character, use CHAR(192).
- The right brace (}) meta-character, use CHAR(208).

Create a Dialogue Manager variable that contains the pattern. To insert the meta-characters, use the CHAR function, and then use that variable as the argument in the regular expression function. For example, to generate the regular expression '[AEIOUaeiou]', which matches all uppercase and lowercase vowels, issue a -SET command similar to the following, which creates a variable named &VCWSTRING:

```
-SET &VCWSTRING=CHAR(173) || 'AEIOUaeiou' || CHAR(189);
```

Then use the &VCWSTRING variable as the regular expression argument in the function call. For example:

```
VowelCnt/I5=REGEXP_COUNT(PRODUCT, '&VCWSTRING');
```

REGEX: Matching a String to a Regular Expression

The REGEX function matches a string to a regular expression and returns true (1) if it matches and false (0) if it does not match.

A regular expression is a sequence of special characters and literal characters that you can combine to form a search pattern.

Many references for regular expressions exist on the web.

Match a String to a Regular Expression

```
REGEX(string, regular_expression)
```

where:

string

Alphanumeric

Is the character string to match.

regular_expression

Alphanumeric

Is a regular expression, enclosed in single quotation marks, constructed using literals and meta-characters. The following meta-characters are supported

- . represents any single character
- * represents zero or more occurrences

- + represents one or more occurrences
- ? represents zero or one occurrence
- ^ represents beginning of line
- $ represents end of line
- [] represents any one character in the set listed within the brackets
- [^] represents any one character not in the set listed within the brackets
- | represents the Or operator
- \ is the Escape Special Character
- () contains a character sequence

For example, the regular expression '^Ste(v|ph)en$' matches values starting with *Ste* followed by either *ph* or *v*, and ending with *en*.

> **Note:** The output value is numeric.

Matching a String Against a Regular Expression

The following request matches the FIRSTNAME field against the regular expression 'PATRIC[(I?)K]', which matches PATRICIA or PATRICK:

```
DEFINE FILE VIDEOTRK
PNAME/I5=REGEX(FIRSTNAME,'PATRIC[(I?)K]');
END
TABLE FILE VIDEOTRK
PRINT FIRSTNAME PNAME
BY LASTNAME
WHERE LASTNAME GE 'M'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

LASTNAME	FIRSTNAME	PNAME
MCMAHON	JOHN	0
MONROE	CATHERINE	0
	PATRICK	1
NON-MEMBER		0
O'BRIEN	DONALD	0
PARKER	GLENDA	0
	RICHARD	0
RATHER	MICHAEL	0
RIESLER	LESLIE	0
SPIVEY	TOM	0
STANDLER	MICHAEL	0
STEWART	MAUDE	0
WHITE	PATRICIA	1
WILLIAMS	KENNETH	0
WILSON	KELLY	0
WU	MARTHA	0

The following version of the request runs on z/OS. The variable &REG1 contains the regular expression string with the circumflex character (^) inserted as CHAR(95), the left bracket character ([) inserted as CHAR(173), and the right bracket character (]) inserted as CHAR (189). The other meta-characters are interpreted correctly.

```
-SET &REG1 = CHAR(95) || 'PATRIC' || CHAR(173) ||
- '(I?)K' || CHAR(189);
DEFINE FILE VIDEOTRK
PNAME/I5 = REGEX(FIRSTNAME,'&REG1') ;
END
TABLE FILE VIDEOTRK
PRINT FIRSTNAME PNAME
BY LASTNAME
WHERE LASTNAME GE 'M'
ON TABLE SET PAGE NOLEAD
 END
```

The output follows.

```
LASTNAME          FIRSTNAME   PNAME
--------          ---------   -----
MCMAHON           JOHN            0
MONROE            CATHERINE       0
                  PATRICK         1
NON-MEMBER                        0
```

```
O'BRIEN          DONALD          0
PARKER           GLENDA          0
                 RICHARD         0
RATHER           MICHAEL         0
RIESLER          LESLIE          0
SPIVEY           TOM             0
STANDLER         MICHAEL         0
STEWART          MAUDE           0
WHITE            PATRICIA        1
WILLIAMS         KENNETH         0
WILSON           KELLY           0
WU               MARTHA          0
```

REGEXP_COUNT: Counting the Number of Matches to a Pattern in a String

REGEXP_COUNT returns the integer count of matches to a specified regular expression pattern within a source string.

Count the Number of Matches to a Pattern in a String

```
REGEXP_COUNT(string, pattern)
```

where:

string

Alphanumeric

Is the input string to be searched.

pattern

Alphanumeric

Is a regular expression, enclosed in single quotation marks, constructed using literals and meta-characters. The following meta-characters are supported

- . represents any single character
- * represents zero or more occurrences
- + represents one or more occurrences
- ? represents zero or one occurrence
- ^ represents beginning of line
- $ represents end of line
- [] represents any one character in the set listed within the brackets
- [^] represents any one character not in the set listed within the brackets
- | represents the Or operator
- \ is the Escape Special Character
- () contains a character sequence

Counting the Number of Matches to a Pattern in a String

The following example uses the following Regular Expression symbols.

- $, which searches for a specified expression that occurs at the end of a string.
- ^, which searches for a specified expression that occurs at the beginning of a string.
- \s*, which matches any number of whitespace characters, such as blank characters.
- [T,t], which matches the characters 'T' and 't'.

In the following request, REG1 is the number of occurrences of the expression 'iscotti', with any number of following whitespace characters, that occur the end of the PRODUCT field. REG2 is the number of occurrences of the characters 'T' and 't' in the PRODUCT field.

```
TABLE FILE GGSALES
SUM DOLLARS AND COMPUTE
REG1/I5 = REGEXP_COUNT(PRODUCT, 'iscotti\s*$');
REG2/I5 = REGEXP_COUNT(PRODUCT, '[T,t]');
BY PRODUCT
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
```

```
ENDSTYLE
END
```

The output is shown in the following image.

Product	Dollar Sales	REG1	REG2
Biscotti	5263317	1	2
Capuccino	2381590	0	0
Coffee Grinder	2337567	0	0
Coffee Pot	2449585	0	1
Croissant	7749902	0	1
Espresso	3906243	0	0
Latte	10943622	0	2
Mug	4522521	0	0
Scone	4216114	0	0
Thermos	2385829	0	1

Using REGEXP_COUNT on Windows and z/OS

The following request uses REGEXP_COUNT to return the number of vowels and number of consonants in each product name, on Windows. VowelCnt is the count of vowels, and ConsonantCnt is the count of non-vowels.

```
DEFINE FILE GGSALES
VowelCnt/I5=REGEXP_COUNT(PRODUCT,'[AEIOUaeiou]');
ConsonantCnt/I5=REGEXP_COUNT(PRODUCT,'[^AEIOUaeiou]');
END
TABLE FILE GGSALES
SUM MAX.VowelCnt AS 'Vowels'
    MAX.ConsonantCnt AS 'Consonants'
BY PRODUCT
END
```

Note:

- Brackets are used to enclose a list of characters that will match the regular expression pattern.
- When the circumflex character (^) prefaces the list of characters within the brackets, the regular expression matches any character not on the list.

The output is shown in the following image.

Product	Vowels	Consonants
Biscotti	3	5
Capuccino	4	5
Coffee Grinder	5	8
Coffee Pot	4	5
Croissant	3	6
Espresso	3	5
Latte	2	3
Mug	1	2
Scone	2	3
Thermos	2	5

The following version of the request uses REGEXP_COUNT to return the number of vowels and number of consonants in each product name, on z/OS. The -SET commands create the regular expressions by using the CHAR function to insert the meta-characters into the expressions. VowelCnt is the count of vowels, and ConsonantCnt is the count of non-vowels.

```
-SET &VCWSTRING=CHAR(173) || 'AEIOUaeiou' || CHAR(189);
-SET &CONSTRING=CHAR(173) || CHAR(95) || 'AEIOU aeiou' || CHAR(189);
DEFINE FILE GGSALES
VowelCnt/I5=REGEXP_COUNT(PRODUCT, '&VCWSTRING');
ConsonantCnt/I5=REGEXP_COUNT(PRODUCT,'&CONSTRING');
END
TABLE FILE GGSALES
SUM MAX.VowelCnt AS 'Vowels'
    MAX.ConsonantCnt AS 'Consonants'
BY PRODUCT
ON TABLE SET PAGE NOLEAD
END
```

The output follows.

```
  Product            Vowels  Consonants
  -------            ------  ----------
  Biscotti                3           5
  Capuccino               4           5
  Coffee Grinder          5           8
  Coffee Pot              4           5
  Croissant               3           6
  Espresso                3           5
  Latte                   2           3
```

```
  Mug                       1            2
  Scone                     2            3
  Thermos                   2            5
```

REGEXP_INSTR: Returning the First Position of a Pattern in a String

REGEXP_INSTR returns the integer position of the first match to a specified regular expression pattern within a source string. The first character position in a string is indicated by the value 1. If there is no match within the source string, the value 0 is returned.

Return the Position of a Pattern in a String

```
REGEXP_INSTR(string, pattern)
```

where:

string

Alphanumeric

Is the input string to be searched.

pattern

Alphanumeric

Is a regular expression, enclosed in single quotation marks, constructed using literals and meta-characters. The following meta-characters are supported

- . represents any single character
- * represents zero or more occurrences
- + represents one or more occurrences
- ? represents zero or one occurrence
- ^ represents beginning of line

- $ represents end of line
- [] represents any one character in the set listed within the brackets
- [^] represents any one character not in the set listed within the brackets
- | represents the Or operator
- \ is the Escape Special Character
- () contains a character sequence

Finding the Position of a Pattern in a String

The following example uses the following Regular Expression symbols.

- $, which searches for a specified expression that occurs at the end of a string.
- ^, which searches for a specified expression that occurs at the beginning of a string.
- \s*, which matches any number of whitespace characters, such as blank characters.
- [B,C,S], which matches the uppercase letters B, C, and S.

In the following request, REG1 is the position of the expression 'iscotti', with any number of following whitespace characters, that occur the end of the PRODUCT field value. REG2 is the position of the characters 'B' ,C, or 'S' that occur at the beginning of the PRODUCT field value.

```
TABLE FILE GGSALES
SUM DOLLARS AND COMPUTE
REG1/I5 = REGEXP_INSTR(PRODUCT, 'iscotti\s*$');
REG2/I5 = REGEXP_INSTR(PRODUCT, '^[B,C,S]');
BY PRODUCT
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Product	Dollar Sales	REG1	REG2
Biscotti	5263317	2	1
Capuccino	2381590	0	1
Coffee Grinder	2337567	0	1
Coffee Pot	2449585	0	1
Croissant	7749902	0	1
Espresso	3906243	0	0
Latte	10943622	0	0
Mug	4522521	0	0
Scone	4216114	0	1
Thermos	2385829	0	0

The following version of the request runs on z/OS. The first regular expression can be input directly because the characters used are interpreted correctly. For the second regular expression, a variable is created to contain the pattern. This variable is then used in the function call.

```
-SET &REG2STR=CHAR(95) || CHAR(173) || 'B,C,S' || CHAR(189);
TABLE FILE GGSALES
SUM DOLLARS AND COMPUTE
REG1/I5 = REGEXP_INSTR(PRODUCT, 'iscotti\s*$');
REG2/I5 = REGEXP_INSTR(PRODUCT, '&REG2STR');
BY PRODUCT
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output follows.

```
  Product              Dollar Sales    REG1    REG2
  -------              ------------    ----    ----
  Biscotti                  5263317       2       1
  Capuccino                 2381590       0       1
  Coffee Grinder            2337567       0       1
  Coffee Pot                2449585       0       1
  Croissant                 7749902       0       1
  Espresso                  3906243       0       0
  Latte                    10943622       0       0
  Mug                       4522521       0       0
  Scone                     4216114       0       1
  Thermos                   2385829       0       0
```

REGEXP_REPLACE: Replacing All Matches to a Pattern in a String

REGEXP_REPLACE returns a string generated by replacing all matches to a regular expression pattern in the source string with the given replacement string. The replacement string can be a null string.

Replace Matches to a Pattern in a String

```
REGEXP_REPLACE(string, pattern, replacement)
```

where:

string

Alphanumeric

Is the input string to be searched.

pattern

Alphanumeric

Is a regular expression, enclosed in single quotation marks, constructed using literals and meta-characters. The following meta-characters are supported

- . represents any single character
- * represents zero or more occurrences
- + represents one or more occurrences
- ? represents zero or one occurrence
- ^ represents beginning of line
- $ represents end of line
- [] represents any one character in the set listed within the brackets
- [^] represents any one character not in the set listed within the brackets
- | represents the Or operator

- \ is the Escape Special Character
- () contains a character sequence

replacement

Alphanumeric

Is the replacement string.

Replacing Matches to a Pattern in a String

The following example uses the following Regular Expression symbol.

- ^, which searches for a specified expression that occurs at the beginning of a string.

In the following request REG1 replaces the string 'North' at the beginning of the REGION field value with the string 'South', and REG2 replaces the string 'Mid' at the beginning of the REGION field value with a null string.

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
REG1/A25 = REGEXP_REPLACE(REGION, '^North', 'South');
REG2/A25 = REGEXP_REPLACE(REGION, '^Mid', '');
BY REGION
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Region	REG1	REG2
Midwest	Midwest	west
Northeast	Southeast	Northeast
Southeast	Southeast	Southeast
West	West	West

The following version of the request runs on z/OS. The regular expression string is created in a variable using a -SET command. The circumflex meta-character (^) is inserted as CHAR (95).

```
-SET &REGSTRING1= CHAR(95) || 'North' ;
-SET &REGSTRING2= CHAR(95) || 'Mid' ;
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
REG1/A25 = REGEXP_REPLACE(REGION, '&REGSTRING1', 'South');
REG2/A25 = REGEXP_REPLACE(REGION, '&REGSTRING2', '');
BY REGION
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output follows.

```
Region       REG1                        REG2
------       ----                        ----
Midwest      Midwest                     west
Northeast    Southeast                   Northeast
Southeast    Southeast                   Southeast
West         West                        West
```

REGEXP_SUBSTR: Returning the First Match to a Pattern in a String

REGEXP_SUBSTR returns a string that contains the first match to a specified regular expression pattern within a source string. If there is no match within the source string, a null string is returned.

Returning the First Match to a Pattern in a String

```
REGEXP_SUBSTR(string, pattern)
```

where:

string

Alphanumeric

Is the input string to be searched.

pattern

Alphanumeric

Is a regular expression, enclosed in single quotation marks, constructed using literals and meta-characters. The following meta-characters are supported

- . represents any single character
- * represents zero or more occurrences
- + represents one or more occurrences
- ? represents zero or one occurrence
- ^ represents beginning of line
- $ represents end of line
- [] represents any one character in the set listed within the brackets
- [^] represents any one character not in the set listed within the brackets
- | represents the Or operator
- \ is the Escape Special Character
- () contains a character sequence

Returning the First Match of a Pattern in a String

The following example uses the following Regular Expression symbols.

- [A-Z], which matches any uppercase letter.
- [a-z], which matches any lowercase letter.

In the following request, REG1 contains the first instance of a string within the REGION field value that starts with an uppercase letter, followed by any number of lowercase letters, followed by the characters 'west'. REG2 contains the first instance of a string within the REGION field value that starts with an uppercase letter, followed by any number of lowercase letters, followed by the characters 'east'.

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
REG1/A25 = REGEXP_SUBSTR(REGION, '[A-Z][a-z]*west');
REG2/A25 = REGEXP_SUBSTR(REGION, '[A-Z][a-z]*east');
BY REGION
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Region	REG1	REG2
Midwest	Midwest	
Northeast		Northeast
Southeast		Southeast
West		

The following version of the request runs on z/OS, where the regular expression is generated as a variable, using the CHAR function to insert the meta-characters. Note that the asterisk meta-character (*) needs to be represented as CHAR(92).

```
-SET &REG1=CHAR(173) || 'A-Z' || CHAR(189) || CHAR(173) || 'a-z'
- || CHAR(189) || CHAR(92) || 'west' ;
-SET &REG2=CHAR(173) || 'A-Z' || CHAR(189)  || CHAR(173) || 'a-z'
- || CHAR(189) || '*east';
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
REG1/A25 = REGEXP_SUBSTR(REGION,'&REG1');
REG2/A25 = REGEXP_SUBSTR(REGION,'&REG2');
BY REGION
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output follows.

```
Region      REG1                        REG2
------      ----                        ----
Midwest     Midwest
Northeast                               Northeast
Southeast                               Southeast
West
```

REPEAT: Repeating a String a Given Number of Times

Given a source string and an integer number, REPEAT returns a string with the source string repeated that number of times. The string containing the repeated strings must be large enough to fit the repetitions or it will contain a truncated value.

Repeat a Character String a Given Number of Times

```
REPEAT(source_str, number)
```

where:

source_str

Alphanumeric

Is the source string to be repeated. If *source_str* is a field, the entire field, including blanks, will be repeated.

number

Numeric

Is the number of times to repeat the source string.

Repeating a String a Given Number of Times

The following request returns a string with FIRST_NAME repeated three times.

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME
COMPUTE REPEAT3/A25 = REPEAT(FIRST_NAME,3);
ON TABLE SET PAGE NOLEAD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The PDF output is shown in the following image.

FIRST_NAME	REPEAT3		
ALFRED	ALFRED	ALFRED	ALFRE
MARY	MARY	MARY	MARY
DIANE	DIANE	DIANE	DIANE
RICHARD	RICHARD	RICHARD	RICHA
JOHN	JOHN	JOHN	JOHN
JOAN	JOAN	JOAN	JOAN
ANTHONY	ANTHONY	ANTHONY	ANTHO
JOHN	JOHN	JOHN	JOHN
ROSEMARIE	ROSEMARIE	ROSEMARIE	ROSEM
ROGER	ROGER	ROGER	ROGER
MARY	MARY	MARY	MARY
BARBARA	BARBARA	BARBARA	BARBA

REPLACE: Replacing a String

REPLACE replaces all instances of a search string in an input string with the given replacement string. The output is always variable length alphanumeric with a length determined by the input parameters.

Replace all Instances of a String

```
REPLACE(input_string , search_string , replacement)
```

where:

input_string

Alphanumeric or text (An, AnV, TX)

Is the input string.

search_string

Alphanumeric or text (An, AnV, TX)

Is the string to search for within the input string.

replacement

Alphanumeric or text (An, AnV, TX)

Is the replacement string to be substituted for the search string. It can be a null string ('').

Replacing a String

REPLACE replaces the string 'South' in the Country Name with the string 'S.'

```
SET TRACEUSER = ON
SET TRACEON = STMTRACE//CLIENT
SET TRACESTAMP=OFF
DEFINE FILE WFLITE
NEWNAME/A20 = REPLACE(COUNTRY_NAME, 'SOUTH', 'S.');
END
TABLE FILE WFLITE
SUM COUNTRY_NAME
BY NEWNAME AS 'New,Name'
WHERE COUNTRY_NAME LIKE 'S%'
ON TABLE SET PAGE NOLEAD
END
```

The output is shown in the following image.

New Name	Customer Country
S. Africa	South Africa
S. Korea	South Korea
Singapore	Singapore
Spain	Spain
Sweden	Sweden
Switzerland	Switzerland

Replacing All Instances of a String

In the following request, the virtual field DAYNAME1 is the string DAY1 with all instances of the string 'DAY’ replaced with the string 'day'. The virtual field DAYNAME2 has all instances of the string 'DAY’ removed.

```
DEFINE FILE WFLITE
DAY1/A30 = 'SUNDAY MONDAY TUESDAY';
DAYNAME1/A30 = REPLACE(DAY1, 'DAY', 'day' );
DAYNAME2/A30 = REPLACE(DAY1, 'DAY', '' );
END
TABLE FILE WFLITE
PRINT DAY1 OVER
DAYNAME1 OVER
DAYNAME2
WHERE EMPLOYEE_NUMBER EQ 'AH118'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  DAY1      SUNDAY MONDAY TUESDAY
  DAYNAME1  SUNday MONday TUESday
  DAYNAME2  SUN MON TUES
```

RIGHT: Returning Characters From the Right of a Character String

Given a source character string, or an expression that can be converted to varchar (variable-length alphanumeric), and an integer number, RIGHT returns that number of characters from the right end of the string.

Return Characters From the Right of a Character String

```
RIGHT(chr_exp, int_exp)
```

where:

chr_exp

Alphanumeric or an expression that can be converted to variable-length alphanumeric.

Is the source character string.

int_exp

Integer

Is the number of characters to be returned.

Returning Characters From the Right of a Character String

The following request computes the length of the last name in the FULLNAME field and returns that number of characters to LAST.

```
TABLE FILE WF_RETAIL_EMPLOYEE
PRINT FULLNAME AND
COMPUTE LEN/I5 = ARGLEN(54, GET_TOKEN(FULLNAME, ' ', 2), LEN); NOPRINT
```

```
COMPUTE LAST/A20 = RIGHT(FULLNAME, LEN);
WHERE RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Full Name	LAST
Steven Wagoner	Wagoner
Adan Geoghegan	Geoghegan
Candace Aguilar	Aguilar
Dianna Turpin	Turpin
John Blankinship	Blankinship
John Chang	Chang
John Mackey	Mackey
Elaine Duran	Duran
Douglas Sanders	Sanders
Linda Whitlow	Whitlow
Phyllis Carey	Carey
Alfred Amerson	Amerson
Jeremy Maness	Maness
David Christopher	Christopher
Alice Flemming	Flemming
Delia Tennison	Tennison
Diane Eads	Eads
Wilfredo Delacruz	Delacruz
Dorothy Newman	Newman
Delia Tennison	Tennison

RPAD: Right-Padding a Character String

RPAD uses a specified character and output length to return a character string padded on the right with that character.

Pad a Character String on the Right

```
RPAD(string, out_length, pad_character)
```

where:

string

Alphanumeric

Is a string to pad on the right side.

out_length

Integer

Is the length of the output string after padding.

pad_character

Alphanumeric

Is a single character to use for padding.

Right-Padding a String

In the following request against the WF_RETAIL data source, RPAD right-pads the PRODUCT_CATEGORY column with @ symbols:

```
DEFINE FILE WFLITE
RPAD1/A25 = RPAD(PRODUCT_CATEGORY,25,'@');
DIG1/A4 = DIGITS(ID_PRODUCT,4);
END
TABLE FILE WFLITE
SUM DIG1 RPAD1
BY PRODUCT_CATEGORY
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
TYPE=DATA,FONT=COURIER,SIZE=11,COLOR=BLUE,$
END
```

The output is:

Product Category	DIG1	RPAD1
Accessories	5005	Accessories@@@@@@@@@@@@@
Camcorder	3006	Camcorder@@@@@@@@@@@@@@@
Computers	6016	Computers@@@@@@@@@@@@@@@
Media Player	1003	Media Player@@@@@@@@@@@@
Stereo Systems	2155	Stereo Systems@@@@@@@@@@
Televisions	4018	Televisions@@@@@@@@@@@@@
Video Production	7005	Video Production@@@@@@@@

Usage Notes for RPAD

- The input string can be data type AnV, VARCHAR, TX, and An.
- Output can only be AnV or An.
- When working with relational VARCHAR columns, there is no need to trim trailing spaces from the field if they are not desired. However, with An and AnV fields derived from An fields, the trailing spaces are part of the data and will be included in the output, with the padding being placed to the right of these positions. You can use TRIM or TRIMV to remove these trailing spaces prior to applying the RPAD function.

RTRIM: Removing Blanks From the Right End of a String

The RTRIM function removes all blanks from the right end of a string.

Remove Blanks From the Right End of a String

```
RTRIM(string)
```

where:

string

Alphanumeric

Is the string to trim on the right.

The data type of the returned string is AnV, with the same maximum length as the source string.

Removing Blanks From the Right End of a String

The following request against the MOVIES data source creates the field DIRSLASH, that contains a slash at the end of the DIRECTOR field. Then it creates the TRIMDIR field, which trims the trailing blanks from the DIRECTOR field and places a slash at the end of that field:

```
TABLE FILE MOVIES
PRINT DIRECTOR NOPRINT AND
COMPUTE
DIRSLASH/A18 = DIRECTOR|'/';
TRIMDIR/A17V = RTRIM(DIRECTOR)|'/';
WHERE DIRECTOR CONTAINS 'BR'
ON TABLE SET PAGE NOPAGE
END
```

On the output, the slashes show that the trailing blanks in the DIRECTOR field were removed in the TRIMDIR field:

```
  DIRSLASH            TRIMDIR
  --------            -------
  ABRAHAMS J.      /  ABRAHAMS J./
  BROOKS R.        /  BROOKS R./
  BROOKS J.L.      /  BROOKS J.L./
```

SPACE: Returning a String With a Given Number of Spaces

Given an integer count, SPACE returns a string consisting of that number of spaces.

Note: To retain the spaces in HTML report output, the SHOWBLANKS parameter must be set to ON.

Return a String With a Given Number of Spaces

```
SPACE(count)
```

where:

count

Numeric

Is the number of spaces to return.

Returning a String With a Given Number of Spaces

The following request inserts 20 spaces between the DOLLARS and UNITS values converted to alphanumeric values. The font used is Courier because it is monospaced and shows the 20 blanks without making them proportional.

```
SET SHOWBLANKS = ON
TABLE FILE GGSALES
SUM DOLLARS NOPRINT  UNITS  NOPRINT AND
COMPUTE ALPHADOLL/A8 = EDIT(DOLLARS); NOPRINT
COMPUTE ALPHAUNIT/A8 = EDIT(UNITS); NOPRINT
COMPUTE Dollars_And_Units_With_Spaces/A60 = ALPHADOLL | SPACE(20) |
ALPHAUNIT;
BY CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET STYLE *
GRID=OFF, FONT=COURIER,$
ENDSTYLE
END
```

The output is shown in the following image.

```
Category  Dollars_And_Units_With_Spaces
Coffee    17231455                  01376266
Food      17229333                  01384845
Gifts     11695502                  00927880
```

SPLIT: Extracting an Element From a String

The SPLIT function returns a specific type of element from a string. The output is returned as a variable length alphanumeric.

Extract an Element From a String

```
SPLIT(element, string)
```

where:

element

Can be one of the following keywords:

- **EMAIL_DOMAIN.** Is the domain name portion of an email address in the string.
- **EMAIL_USERID.** Is the user ID portion of an email address in the string.
- **URL_PROTOCOL.** Is the URL protocol for a URL in the string.
- **URL_HOST.** Is the host name of the URL in the string.
- **URL_PORT.** Is the port number of the URL in the string.
- **URL_PATH.** Is the URL path for a URL in the string.
- **NAME_FIRST.** Is the first token (group of characters) in the string. Tokens are delimited by blanks.
- **NAME_LAST.** Is the last token (group of characters) in the string. Tokens are delimited by blanks.

string

Alphanumeric

Is the string from which the element will be extracted.

Extracting an Element From a String

The following request defines strings and extracts elements from them.

```
DEFINE FILE WFLITE
STRING1/A50 WITH COUNTRY_NAME= 'http://www.informationbuilders.com';
STRING2/A20 = 'user1@ibi.com';
STRING3/A20 = 'Louisa May Alcott';
Protocol/A20 = SPLIT(URL_PROTOCOL, STRING1);
Path/A50 = SPLIT(URL_PATH, STRING1);
Domain/A20 = SPLIT(EMAIL_DOMAIN, STRING2);
User/A20 =  SPLIT(EMAIL_USERID, STRING2);
First/A10 = SPLIT(NAME_FIRST, STRING3);
Last/A10 = SPLIT(NAME_LAST, STRING3);
END
TABLE FILE WFLITE
SUM Protocol Path User Domain First Last
ON TABLE SET PAGE NOLEAD
END
```

The output is shown in the following image.

Protocol	Path	User	Domain	First	Last
http	http://www.informationbuilders.com	user1	ibi.com	Louisa	Alcott

SUBSTRING: Extracting a Substring From a Source String

The SUBSTRING function extracts a substring from a source string. If the ending position you specify for the substring is past the end of the source string, the position of the last character of the source string becomes the ending position of the substring.

Extract a Substring From a Source String

```
SUBSTRING(string, position, length)
```

where:

string

Alphanumeric

Is the string from which to extract the substring. It can be a field, a literal in single quotation marks (‘), or a variable.

position

Positive Integer

Is the starting position of the substring in **string**.

length

Integer

Is the limit for the length of the substring. The ending position of the substring is calculated as **position** + **length** - 1. If the calculated position beyond the end of the source string, the position of the last character of **string** becomes the ending position.

The data type of the returned substring is AnV.

Extracting a Substring From a Source String

In the following request, POSITION determines the position of the first letter I in LAST_NAME and stores the result in I_IN_NAME. SUBSTRING, then extracts three characters beginning with the letter I from LAST_NAME and stores the results in I_SUBSTR.

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE
I_IN_NAME/I2 = POSITION('I', LAST_NAME); AND
COMPUTE
I_SUBSTR/A3 =
SUBSTRING(LAST_NAME, I_IN_NAME, I_IN_NAME+2);
BY LAST_NAME
```

```
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  LAST_NAME        I_IN_NAME  I_SUBSTR
  ---------        ---------  --------
  BANNING                  5  ING
  BLACKWOOD                0  BL
  CROSS                    0  CR
  GREENSPAN                0  GR
  IRVING                   1  IRV
  JONES                    0  JO
  MCCOY                    0  MC
  MCKNIGHT                 5  IGH
  ROMANS                   0  RO
  SMITH                    3  ITH
                           3  ITH
  STEVENS                  0  ST
```

TOKEN: Extracting a Token From a String

The token function extracts a token (substring) from a string of characters. The tokens are separated by a delimiter consisting of one or more characters and specified by a token number reflecting the position of the token in the string.

Extract a Token From a String

```
TOKEN(string, delimiter, number)
```

where:

string

Fixed length alphanumeric

Is the character string from which to extract the token.

delimiter

Fixed length alphanumeric

Is a delimiter consisting of one or more characters.

TOKEN can be optimized if the delimiter consists of a single character.

number

Integer

Is the token number to extract.

Extracting a Token From a String

TOKEN extracts the second token from the PRODUCT_SUBCATEG column, where the delimiter is the letter P:

```
DEFINE FILE WFLITE
TOK1/A20 = TOKEN(PRODUCT_SUBCATEG,'P',2);
END
TABLE FILE WFLITE
SUM TOK1 AS Token
BY PRODUCT_SUBCATEG
ON TABLE SET PAGE NOPAGE
END
```

The output is:

Product Subcategory	Token
Blu Ray	
Boom Box	
CRT TV	
Charger	
DVD Players	layers
DVD Players - Portable	layers -
Flat Panel TV	anel TV
Handheld	
Headphones	hones
Home Theater Systems	
Portable TV	ortable TV
Professional	rofessional
Receivers	
Smartphone	hone
Speaker Kits	eaker Kits
Standard	
Streaming	
Tablet	
Universal Remote Controls	
Video Editing	
iPod Docking Station	od Docking Station

TRIM_: Removing a Leading Character, Trailing Character, or Both From a String

The TRIM_ function removes all occurrences of a single character from either the beginning or end of a string, or both.

Note:

- Leading and trailing blanks count as characters. If the character you want to remove is preceded (for leading) or followed (for trailing) by a blank, the character will not be removed. Alphanumeric fields that are longer than the number of characters stored within them are padded with trailing blanks.
- The function will be optimized when run against a relational DBMS that supports trimming the character and location specified.

Remove a Leading Character, Trailing Character, or Both From a String

```
TRIM_(where, pattern, string)
```

where:

where

Keyword

Defines where to trim the source string. Valid values are:

- **LEADING,** which removes leading occurrences.
- **TRAILING,** which removes trailing occurrences.
- **BOTH,** which removes leading and trailing occurrences.

pattern

Alphanumeric

Is a single character, enclosed in single quotation marks ('), whose occurrences are to be removed from **string**. For example, the character can be a single blank (‘ ‘).

string

Alphanumeric

Is the string to be trimmed.

The data type of the returned string is AnV.

Trimming a Character From a String

In the following request, TRIM_ removes leading occurrences of the character ‘B’ from the DIRECTOR field:

```
TABLE FILE MOVIES
PRINT DIRECTOR AND
COMPUTE
TRIMDIR/A17 = TRIM_(LEADING, 'B', DIRECTOR);
WHERE DIRECTOR CONTAINS 'BR'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  DIRECTOR           TRIMDIR
  --------           -------
  ABRAHAMS J.        ABRAHAMS J.
  BROOKS R.          ROOKS R.
  BROOKS J.L.        ROOKS J.L.
```

Trimming With Trailing Blanks

The following request trims a trailing period (.) from the director name. The field DIRECTOR has format A17, so there are trailing blanks in most of the instances of the field. To create a field (DIRECTORV) without trailing blanks, SQUEEZ converts the trailing blanks in DIRECTOR to a single blank, then TRIMV removes the remaining trailing blank and stores it with format A17V, so the length of the actual characters is known. Then TRIM_ is called against DIRECTOR and DIRECTORV, creating the fields TRIMDIR (trimmed DIRECTOR) and TRIMDIRV (trimmed DIRECTORV) :

```
DEFINE FILE MOVIES
DIRECTORV/A17V = TRIMV('T', SQUEEZ(17, DIRECTOR, 'A17V'), 17, ' ', 1,
DIRECTORV) ;
TRIMDIR/A17 = TRIM_(TRAILING, '.', DIRECTOR);
TRIMDIRV/A17V = TRIM_(TRAILING, '.', DIRECTORV);
```

```
END
TABLE FILE MOVIES
PRINT DIRECTOR TRIMDIR DIRECTORV TRIMDIRV
ON TABLE SET PAGE NOPAGE
END
```

The partial output shows that the trimmed DIRECTOR field still has the trailing periods because the period is not the last character in the field. In the trimmed DIRECTORV field, the trailing periods have been removed:

```
DIRECTOR          TRIMDIR           DIRECTORV         TRIMDIRV
--------          -------           ---------         --------
SPIELBERG S.      SPIELBERG S.      SPIELBERG S.      SPIELBERG S
KAZAN E.          KAZAN E.          KAZAN E.          KAZAN E
WELLES O.         WELLES O.         WELLES O.         WELLES O
LUMET S.          LUMET S.          LUMET S.          LUMET S
```

UPPER: Returning a String With All Letters Uppercase

The UPPER function takes a source string and returns a string of the same data type with all letters translated to uppercase.

Return a String With All Letters Uppercase

```
UPPER(string)
```

where:

string

Alphanumeric

Is the string to convert to uppercase.

The returned string is the same data type and length as the source string.

Converting Letters to Uppercase

In the following request, LCWORD converts LAST_NAME to mixed case. Then UPPER converts the LAST_NAME_MIXED field to uppercase:

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=LCWORD(15, LAST_NAME, 'A15');
LAST_NAME_UPPER/A15=UPPER(LAST_NAME_MIXED) ;
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME_UPPER AND FIRST_NAME
BY LAST_NAME_MIXED
WHERE CURR_JOBCODE EQ 'B02' OR 'A17' OR 'B04';
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  LAST_NAME_MIXED  LAST_NAME_UPPER  FIRST_NAME
  ---------------  ---------------  ----------
  Banning          BANNING          JOHN
  Blackwood        BLACKWOOD        ROSEMARIE
  Cross            CROSS            BARBARA
  Mccoy            MCCOY            JOHN
  Mcknight         MCKNIGHT         ROGER
  Romans           ROMANS           ANTHONY
```

Character Functions

Character functions manipulate alphanumeric fields and character strings.

Character Function Notes

For many functions, the *output* argument can be supplied either as a field name or as a format enclosed in single quotation marks. However, if a function is called from a Dialogue Manager command, this argument must always be supplied as a format. For detailed information about calling a function and supplying arguments, see Accessing and Calling a Function.

ARGLEN: Measuring the Length of a String

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

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

Measure the Length of a Character String

```
ARGLEN(length, source_string, output)
```

where:

length

Integer

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

source_string

Alphanumeric

Is the name of the field containing the character string.

output

Integer

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

Measuring the Length of a Character String

ARGLEN determines the length of the character string in LAST_NAME and stores the result in NAME_LEN:

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

The output is:

LAST_NAME	NAME_LEN
---------	--------
SMITH	5

JONES	5
MCCOY	5
BLACKWOOD	9
GREENSPAN	9
CROSS	5

ASIS: Distinguishing Between Space and Zero

The ASIS function distinguishes between a space and a zero in Dialogue Manager. It differentiates between a numeric string, a constant or variable defined as a numeric string (a 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.

Distinguish Between a Space and a Zero

```
ASIS(argument)
```

where:

argument

Alphanumeric

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

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

Distinguishing Between a Space and a Zero

The first request does not use ASIS. No difference is detected between variables defined as a space and 0.

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

The output is:

```
VAR1 EQ VAR2 0 TRUE
```

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

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

The output is:

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

Usage Notes for ASIS

In general, Dialogue Manager variables are treated as alphanumeric values. However, a Dialogue Manager variable with the value of '.' may be treated as an alphanumeric value

('.') or a number (0) depending on the context used.

- If the Dialogue Manager variable '.' is used in a mathematical expression, its value will be treated as a number. For example, in the following request, &DMVAR1 is used in an arithmetic expression and is evaluated as zero (0).

```
-SET &DMVAR1='.';
-SET &DMVAR2=10 + &DMVAR1;
-TYPE DMVAR2 = &DMVAR2
```

  The output is;

```
DMVAR2 = 10
```

- If the Dialogue Manager variable value '.' is used in an IF test and is compared to the values ' ', '0', or '.', the result will be TRUE even if ASIS is used, as shown in the following example. The following IF tests all evaluate to TRUE.

```
-SET &DMVAR1='.';
-SET &DMVAR2=IF &DMVAR1 EQ ' ' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR3=IF &DMVAR1 EQ '.' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR4=IF &DMVAR1 EQ '0' THEN 'TRUE' ELSE 'FALSE';
```

- If the Dialogue Manager variable is used with ASIS, the result of the ASIS function will be always be considered alphanumeric and will distinguish between the space (‘ ‘), zero (‘0’), or period (‘.’), as in the following example. The following IF tests all evaluate to TRUE.

```
-SET &DMVAR2=IF ASIS('.') EQ '.' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR3=IF ASIS(' ') EQ ' ' THEN 'TRUE' ELSE 'FALSE';
-SET &DMVAR4=IF ASIS('0') EQ '0' THEN 'TRUE' ELSE 'FALSE';
```

- Comparing ASIS('0') to ' ' and ASIS(' ') to '0' always evaluates to FALSE.

BITSON: Determining If a Bit Is On or Off

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

Determine If a Bit Is On or Off

```
BITSON(bitnumber, source_string, output)
```

where:

bitnumber

Integer

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

source_string

Alphanumeric

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

output

Integer

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

Evaluating a Bit in a Field

BITSON evaluates the 24th bit of LAST_NAME and stores the result in BIT_24:

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

The output is:

```
LAST_NAME        BIT_24
---------        ------
```

```
SMITH                1
JONES                1
MCCOY                1
BLACKWOOD            1
GREENSPAN            1
CROSS                0
```

BITVAL: Evaluating a Bit String as an Integer

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 subset of bits in the string as an integer value.

If the number of bits is:

- Less than 1, the returned value is 0.
- Between 1 and 31 (the recommended range), the returned value is a zero or positive number representing the bits specified, extended with high-order zeroes for a total of 32 bits.
- Exactly 32, the returned value is the positive, zero, or the complement value of negative two, of the specified 32 bits.
- Greater than 32 (33 or more), the returned value is the positive, zero, or the complement value of negative two, of the rightmost 32 bits specified.

Evaluate a Bit String

```
BITVAL(source_string, startbit, number, output)
```

where:

source_string

Alphanumeric

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

startbit

Integer

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

number

Integer

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

output

Integer

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

Evaluating a Bit String

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

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

The output is:

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

BYTVAL: Translating a Character to Decimal

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

Translate a Character

```
BYTVAL(character, output)
```

where:

character

Alphanumeric

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

output

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.

Translating the First Character of a Field

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

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

The output on an ASCII platform is:

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

The output on an EBCDIC platform is:

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

Returning the EBCDIC Value With Dialogue Manager

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

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

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

```
ENTER THE CHARACTER TO BE DECODED
!
THE EQUIVALENT VALUE IS 90
>
```

CHKFMT: Checking the Format of a String

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

If the mask is shorter than the character string, the function checks only the portion of the character string corresponding to the mask. For example, if you are using a four-character mask to test a nine-character string, only the first four characters in the string are checked; the rest is returned as a no match with CHKFMT giving the first non-matching position as the result.

Check the Format of a Character String

```
CHKFMT(numchar, source_string, 'mask', output)
```

where:

numchar

Integer

Is the number of characters being compared to the mask.

string

Alphanumeric

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

'mask'

Alphanumeric

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

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

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

`9` is any digit between 0–9.

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

`$` is any character.

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

output

Integer

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

Checking the Format of a Field

CHKFMT examines EMP_ID for nine numeric characters starting with 11 and stores the result in CHK_ID:

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

The output is:

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

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

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

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

A sample execution is:

```
>
 EMPLOYEEFOCUS   A ON 12/05/96 AT 15.42.03
 DATA FOR TRANSACTION    1
 EMP_ID      =
111w2
 LAST_NAME   =
johnson
 FIRST_NAME  =
greg
 DEPARTMENT  =
production
 BAD EMPLOYEE ID: 111W2
 INVALID CHARACTER IN POSITION   4
 DATA FOR TRANSACTION    2
 EMP_ID      =
end
```

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

The procedure processes as follows:

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

CHKNUM: Checking a String for Numeric Format

The CHKNUM function checks a character string for numeric format. If the string contains a valid numeric format, CHKNUM returns the value 1. If the string contains characters that are not valid in a number, CHKNUM returns zero (0).

Check the Format of a Character String

```
CHKNUM(numchar, source_string, output)
```

where:

numchar

Integer

Is the number of characters in the string.

string

Alphanumeric

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

output

Numeric

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

Checking a String for Numeric Format

CHKNUM examines the strings STR1, STR2, and STR3 for numeric format.

```
DEFINE FILE WFLITE
STR1/A8 = '12345E01';
STR2/A8 = 'ABCDEFG';
STR3/A8 = '1234.567';
CHK1/I1= CHKNUM(8,STR1,CHK1);
CHK2/I1= CHKNUM(8,STR2,CHK2);
CHK3/I1= CHKNUM(8,STR3,CHK3);
END
TABLE FILE WFLITE
PRINT STR1 IN 20 CHK1 STR2 CHK2 STR3 CHK3
BY PRODUCT_CATEGORY
WHERE PRODUCT_CATEGORY EQ 'Video Production'
ON  TABLE SET PAGE NOPAGE
ON TABLE PCHOLD FORMAT WP
END
```

The output is:

```
Product

Category            STR1      CHK1  STR2      CHK2  STR3      CHK3

--------            ----      ----  ----      ----  ----      ----

Video Production    12345E01     1  ABCDEFG      0  1234.567     1
                    12345E01     1  ABCDEFG      0  1234.567     1
```

```
                12345E01      1  ABCDEFG       0  1234.567      1
                12345E01      1  ABCDEFG       0  1234.567      1
                12345E01      1  ABCDEFG       0  1234.567      1
                12345E01      1  ABCDEFG       0  1234.567      1
```

CTRAN: Translating One Character to Another

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

To use CTRAN, you must know the decimal equivalent of the characters in internal machine representation. Note that the coding chart for conversion is platform-dependent, hence your platform and configuration option determines whether ASCII, EBCDIC, or Unicode coding is used.

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

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

Translate One Character to Another

```
CTRAN(length, source_string, decimal, decvalue, output)
```

where:

length

Integer

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

source_string

Alphanumeric

Is the character string to be translated enclosed in single quotation marks ('), or the field or variable that contains the character string.

decimal

Integer

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

decvalue

Integer

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

output

Alphanumeric

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

Translating Spaces to Underscores on an ASCII Platform

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

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

The output is:

```
EMP_ID     ADDRESS_LN3           ALT_ADDR
------     -----------           --------
117593129  RUTHERFORD NJ 07073   RUTHERFORD_NJ_07073_
119265415  NEW YORK NY 10039     NEW_YORK_NY_10039___
119329144  FREEPORT NY 11520     FREEPORT_NY_11520___
123764317  NEW YORK NY 10001     NEW_YORK_NY_10001___
126724188  FREEPORT NY 11520     FREEPORT_NY_11520___
451123478  ROSELAND NJ 07068     ROSELAND_NJ_07068___
543729165  JERSEY CITY NJ 07300  JERSEY_CITY_NJ_07300
818692173  FLUSHING NY 11354     FLUSHING_NY_11354
```

Translating Spaces to Underscores on an EBCDIC Platform

CTRAN translates the spaces in ADDRESS_LN3 (EBCDIC decimal value 64) to underscores (EBCDIC decimal value 109) and stores the result in ALT_ADDR:

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

The output is:

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

Inserting Accented Letter E's With MODIFY

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

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

The request is:

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

A sample execution follows:

```
***** NEW EMPLOYEE ENTRY SCREEN *****

ENTER EMPLOYEE'S ID:  999888777
```

```
ENTER EMPLOYEE'S FIRST AND LAST NAME
SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS

FIRST_NAME:  AD*LE       LAST_NAME:  MOLI*RE

ENTER THE DEPARTMENT ASSIGNMENT:  SALES
```

The request processes as:

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

   EMPLOYEE ID: 999888777

   FIRST_NAME: AD*LE

   LAST_NAME: MOLI*RE

   DEPARTMENT: SALES
3. The procedure searches the data source for the employee ID. If it does not find it, it continues processing the request.
4. CTRAN converts the asterisks into È's in both the first and last names (ADÈLE MOLIÈRE).

   ```
   ***** NEW EMPLOYEE ENTRY SCREEN *****

   ENTER EMPLOYEE'S ID:

   ENTER EMPLOYEE'S FIRST AND LAST NAME
   SUBSTITUTE *'S FOR ALL ACCENTED E CHARACTERS

   FIRST_NAME:              LAST_NAME:

   ENTER THE DEPARTMENT ASSIGNMENT:




   FIRST_NAME: ADÈLE LAST_NAME: MOLIÈRE
   ```
5. The procedure stores the data in the data source.

Inserting Commas With MODIFY

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

The request is:

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

A sample execution follows:

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

 EMP_ID      =
224466880
 LAST_NAME   =
BRADLEY; JR.
 FIRST_NAME  =
JOHN
 DEPARTMENT  =
MIS
 DATA FOR TRANSACTION    2
 EMP_ID      =
end
 TRANSACTIONS:         TOTAL =     1  ACCEPTED=     1  REJECTED=     0
 SEGMENTS:             INPUT =     1  UPDATED =     0  DELETED =     0
>
```

The request processes as:

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

   EMP_ID: 224466880

   LAST_NAME: BRADLEY; JR.

   FIRST_NAME: JOHN

   DEPARTMENT: MIS
2. The request searches the data source for the ID 224466880. If it does not find the ID, it continues processing the transaction.
3. CTRAN converts the semicolon in "BRADLEY; JR." to a comma. The last name is now "BRADLEY, JR."
4. The request adds the transaction to the data source.
5. This request displays the semicolon converted to a comma:

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

The output is:

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

CTRFLD: Centering a Character String

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

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

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

Center a Character String

```
CTRFLD(source_string, length, output)
```

where:

source_string

Alphanumeric

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

length

Integer

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

output

Alphanumeric

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

Centering a Field

CTRFLD centers LAST_NAME and stores the result in CENTER_NAME:

```
SET STYLE=OFF
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
CENTER_NAME/A12 = CTRFLD(LAST_NAME, 12, 'A12');
```

```
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

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

EDIT: Extracting or Adding Characters

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

EDIT compares the characters in a mask to the characters in a source string. When it encounters a nine (9) in the mask, EDIT copies the corresponding character from the source field to the output string. When it encounters a dollar sign ($) in the mask, EDIT ignores the corresponding character in the source string. When it encounters any other character in the mask, EDIT copies that character to the corresponding position in the output string. This process ends when the mask is exhausted.

Note:

- EDIT does not require an output argument because the result is alphanumeric and its size is determined from the mask value.
- EDIT can also convert the format of a field.

Extract or Add Characters

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

where:

source_string

Alphanumeric

Is a character string from which to pick characters. Each 9 in the mask represents one digit, so the size of **source_string** must be at least as large as the number of 9's in the mask.

mask

Alphanumeric

Is a string of mask characters enclosed in single quotation marks or a field containing the character string enclosed in single quotation marks. The length of the mask, excluding characters other than 9 and $, determines the length of the output field.

Extracting and Adding Characters

EDIT extracts the first initial from the FIRST_NAME field and stores the result in FIRST_INIT. EDIT also adds dashes to the EMP_ID field and stores the result in EMPIDEDIT. The mask used to extract the first initial is stored in the virtual field named MASK1:

```
DEFINE FILE EMPLOYEE
MASK1/A10 = '9$$$$$$$$$'
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
FIRST_INIT/A1 = EDIT(FIRST_NAME, MASK1);
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)

The GETTOK function divides a character string into substrings, called tokens. The data must have a specific character, called a delimiter, that occurs in the string and separates the string into tokens. GETTOK returns the token specified by the **token_number** argument. GETTOK ignores leading and trailing blanks in the source character string.

For example, suppose you want to extract the fourth word from a sentence. In this case, use the space character for a delimiter and the number 4 for **token_number**. GETTOK divides the sentence into words using this delimiter, then extracts the fourth word. If the string is not divided by the delimiter, use the PARAG function for this purpose.

Extract a Substring (Token)

```
GETTOK(source_string, inlen, token_number, 'delim', outlen, output)
```

where:

source_string

Alphanumeric

Is the source string from which to extract the token.

inlen

Integer

Is the number of characters in **source_string**. If this argument is less than or equal to 0, the function returns spaces.

token_number

Integer

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

'delim'

Alphanumeric

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

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

outlen

Integer

Is the size of the token extracted. 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.

output

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 that the delimiter is not included in the extracted token.

Extracting a Token

GETTOK extracts the last token from ADDRESS_LN3 and stores the result in LAST_TOKEN.

The delimiter is a space:

```
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

The LCWORD function converts the letters in a character string to mixed-case. It converts every alphanumeric character to lowercase except the first letter of each new word and the first letter after a single or double quotation mark, which it converts to uppercase. For example, O'CONNOR is converted to O'Connor and JACK'S to Jack'S.

LCWORD skips numeric and special characters in the source string and continues to convert the following alphabetic characters. The result of LCWORD is a string in which the initial uppercase characters of all words are followed by lowercase characters.

Convert a Character String to Mixed-Case

```
LCWORD(length, source_string, output)
```

where:

length

Integer

Is the number of characters in **source_string** and **output**.

string

Alphanumeric

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

output

Alphanumeric

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

Converting a Character String to Mixed-Case

LCWORD converts the LAST_NAME field to mixed-case and stores the result in MIXED_CASE.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
MIXED_CASE/A15 = LCWORD(15, LAST_NAME, MIXED_CASE);
WHERE DEPARTMENT EQ 'PRODUCTION'
END
```

The output is:

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

LCWORD2: Converting a String to Mixed-Case

The LCWORD2 function converts the letters in a character string to mixed-case by converting the first letter of each word to uppercase and converting every other letter to lowercase. In addition, a double quotation mark or a space indicates that the next letter should be converted to uppercase.

For example, "SMITH" would be changed to "Smith" and "JACK S" would be changed to "Jack S".

Convert a Character String to Mixed-Case

```
LCWORD2(length, string, output)
```

where:

length

Integer

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

string

Alphanumeric

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

output

Alphanumeric

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

Converting a Character String to Mixed-Case

LCWORD2 converts the string O'CONNOR’s to mixed-case:

```
DEFINE FILE EMPLOYEE
MYVAL1/A10='O'CONNOR'S';
LC2/A10 = LCWORD2(10, MYVAL1, 'A10');
END
TABLE FILE EMPLOYEE
SUM LAST_NAME NOPRINT MYVAL1 LC2
END
```

The output is:

```
MYVAL1      LC2
------      ---
O'CONNOR'S  O'Connor's
```

LCWORD3: Converting a String to Mixed-Case

The LCWORD3 function converts the letters in a character string to mixed-case by converting the first letter of each word to uppercase and converting every other letter to lowercase. In addition, a single quotation mark indicates that the next letter should be converted to uppercase, as long as it is neither followed by a blank nor the last character in the input string.

For example, 'SMITH' would be changed to 'Smith' and JACK'S would be changed to Jack's.

Convert a Character String to Mixed-Case Using LCWORD3

```
LCWORD3(length, string, output)
```

where:

length

Integer

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

string

Alphanumeric

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

output

Alphanumeric

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

Converting a Character String to Mixed-Case Using LCWORD3

LCWORD3 converts the strings O'CONNOR’s and o’connor’s to mixed-case:

```
DEFINE FILE EMPLOYEE
MYVAL1/A10='O'CONNOR'S';
MYVAL2/A10='o'connor's';
LC1/A10 = LCWORD3(10, MYVAL1, 'A10');
LC2/A10 = LCWORD3(10, MYVAL2, 'A10');
END
TABLE FILE EMPLOYEE
SUM LAST_NAME NOPRINT MYVAL1 LC1 MYVAL2 LC2
END
```

On the output, the letter **C** after the first single quotation mark is in uppercase because it is not followed by a blank and is not the final letter in the input string. The letter **s** after the second single quotation mark (') is in lowercase because it is the last character in the input string:

```
MYVAL1      LC1         MYVAL2      LC2
------      ---         ------      ---
O'CONNOR'S  O'Connor's  o'connor's  O'Connor's
```

LJUST: Left-Justifying a String

LJUST 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.

Left-Justify a Character String

```
LJUST(length, source_string, output)
```

where:

length

Integer

Is the number of characters in *source_string* and *output*, or a field that contains the length.

source_string

Alphanumeric

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

output

Alphanumeric

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

Left-Justifying a String

The following request creates the XNAME field in which the last names are not left-justified. Then, LJUST left-justifies the XNAME field and stores the result in YNAME.

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

TABLE FILE EMPLOYEE
PRINT LAST_NAME XNAME YNAME
END
```

The output is:

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

LOCASE: Converting Text to Lowercase

The LOCASE function converts alphanumeric text to lowercase.

It is useful for converting input fields from FIDEL CRTFORMs and non-FOCUS applications to lowercase.

Convert Text to Lowercase

```
LOCASE(length, source_string, output)
```

where:

length

Integer

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

source_string

Alphanumeric

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

output

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 *source_string*.

Converting a String to Lowercase

LOCASE converts the LAST_NAME field to lowercase and stores the result in LOWER_NAME:

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

The output is:

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

OVRLAY: Overlaying a Character String

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

Overlay a Character String

```
OVRLAY(source_string, length, substring, sublen, position, output)
```

where:

source_string

Alphanumeric

Is the base character string.

stringlen

Integer

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

substring

Alphanumeric

Is the substring that will overlay **source_string**.

sublen

Integer

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

position

Integer

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

output

Alphanumeric

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

Note that if the overlaid string is longer than the output field, the string is truncated to fit the field.

Replacing Characters in a Character String

OVRLAY replaces the last three characters of EMP_ID with CURR_JOBCODE to create a new security identification code and stores the result in NEW_ID:

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

The output is:

```
LAST_NAME        FIRST_NAME  EMP_ID     CURR_JOBCODE  NEW_ID
---------        ----------  ------     ------------  ------
BLACKWOOD        ROSEMARIE   326179357  B04           326179B04
CROSS            BARBARA     818692173  A17           818692A17
GREENSPAN        MARY        543729165  A07           543729A07
JONES            DIANE       117593129  B03           117593B03
MCCOY            JOHN        219984371  B02           219984B02
SMITH            MARY        112847612  B14           112847B14
```

Overlaying a Character in a String With MODIFY

This MODIFY procedure prompts for input using a CRTFORM screen and updates first names in the EMPLOYEE data source. The CRTFORM LOWER option enables you to update the names in lowercase, but the procedure ensures that the first letter of each name is capitalized.

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

The COMPUTE command invokes two functions:

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

The procedure processes as:

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

   Enter the employee's ID: 071382660

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

   ```
   ENTER EMPLOYEE'S ID:
   ENTER FIRST_NAME IN LOWER CASE:
   CHANGING FIRST NAME TO Alfred
   ```
5. The procedure updates the first name in the data source.
6. When you exit the procedure with PF3, the transaction message indicates that one update occurred:

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

PARAG: Dividing Text Into Smaller Lines

The PARAG function divides a character string into substrings by marking them with a delimiter. It scans a specific number of characters from the beginning of the string and replaces the last space in the group scanned with the delimiter, thus creating a first substring, also known as a token. It then scans the next group of characters in the line, starting from the delimiter, and replaces its last space with a second delimiter, creating a second token. It repeats this process until it reaches the end of the line.

Once each token is marked off by the delimiter, you can use the function GETTOK to place the tokens into different fields. If PARAG does not find any spaces in the group it scans, it replaces the first character after the group with the delimiter. Therefore, make sure that any group of characters has at least one space. The number of characters scanned is provided as the maximum token size.

For example, if you have a field called 'subtitle' which contains a large amount of text consisting of words separated by spaces, you can cut the field into roughly equal substrings by specifying a maximum token size to divide the field. If the field is 350 characters long, divide it into three substrings by specifying a maximum token size of 120 characters. This technique enables you to print lines of text in paragraph form.

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

Divide Text Into Smaller Lines

```
PARAG(length, source_string, 'delimiter', max_token_size, output)
```

where:

length

Integer

Is the number of characters in *source_string* and *output*, or a field that contains the length.

source_string

Alphanumeric

Is a string to divide into tokens enclosed in single quotation marks, or a field or variable that contains the text.

delimiter

Alphanumeric

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

max_token_size

Integer

Is the upper limit for the size of each token.

output

Alphanumeric

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

Dividing Text Into Smaller Lines

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

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

The output is:

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

PATTERN: Generating a Pattern From a String

The PATTERN function examines a source string and produces a pattern that indicates the sequence of numbers, uppercase letters, and lowercase letters in the source string. This function is useful for examining data to make sure that it follows a standard pattern.

In the output pattern:

- Any character from the input that represents a single-byte digit becomes the character **9**.
- Any character that represents an uppercase letter becomes **A**, and any character that represents a lowercase letter becomes **a**. For European NLS mode (Western Europe, Central Europe), **A** and **a** are extended to apply to accented alphabets.
- For Japanese, double-byte characters and Hankaku-katakana become **C** (uppercase). Note that double-byte includes Hiragana, Katakana, Kanji, full-width alphabets, full-width numbers, and full-width symbols. This means that all double-byte letters such as Chinese and Korean are also represented as **C**.
- Special characters remain unchanged.

- An unprintable character becomes the character **X**.

Generate a Pattern From an Input String

```
PATTERN (length, source_string,  output)
```

where:

length

Numeric

Is the length of *source_string*.

source_string

Alphanumeric

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

output

Alphanumeric

Is the name of the field to contain the result or the format of the field enclosed in single quotation marks.

Producing a Pattern From Alphanumeric Data

The following 19 records are stored in a fixed format sequential file (with LRECL 14) named TESTFILE:

```
212-736-6250
212 736 4433
123-45-6789
800-969-INFO
10121-2898
10121
2 Penn Plaza
917-339-6380
```

```
917-339-4350
(212) 736-6250
(212) 736-4433
212-736-6250
212-736-6250
212-736-6250
(212) 736 5533
(212) 736 5533
(212) 736 5533
10121 Æ
800-969-INFO
```

The Master File is:

```
FILENAME=TESTFILE, SUFFIX=FIX     ,
  SEGMENT=TESTFILE, SEGTYPE=S0, $
    FIELDNAME=TESTFLD, USAGE=A14, ACTUAL=A14, $
```

The following request generates a pattern for each instance of TESTFLD and displays them by the pattern that was generated. It shows the count of each pattern and its percentage of the total count. The PRINT command shows which values of TESTFLD generated each pattern.

```
DYNAM ALLOC DD TESTFILE DA USER1.TESTFILE.FTMDEFINE FILE TESTFILE
   PATTERN/A14 = PATTERN (14, TESTFLD, 'A14' ) ;
END
TABLE FILE TESTFILE
  SUM CNT.PATTERN AS 'COUNT' PCT.CNT.PATTERN AS 'PERCENT'
   BY PATTERN
 PRINT TESTFLD
   BY PATTERN
ON TABLE COLUMN-TOTAL
END
```

Note that the next to last line produced a pattern from an input string that contained an unprintable character, so that character was changed to X. Otherwise, each numeric digit generated a 9 in the output string, each uppercase letter generated the character ‘A’, and each lowercase letter generated the character ‘a’. The output is:

```
PATTERN         COUNT  PERCENT  TESTFLD
-------         -----  -------  -------
(999) 999 9999      3    15.79  (212) 736 5533
```

```
                             (212) 736 5533
                             (212) 736 5533
(999) 999-9999      2    10.53  (212) 736-6250
                             (212) 736-4433
9 Aaaa Aaaaa        1     5.26  2 Penn Plaza
999 999 9999        1     5.26  212 736 4433
999-99-9999         1     5.26  123-45-6789
999-999-AAAA        2    10.53  800-969-INFO
                                800-969-INFO
999-999-9999        6    31.58  212-736-6250
                                917-339-6380
                                917-339-4350
                                212-736-6250
                                212-736-6250
                                212-736-6250
99999               1     5.26  10121
99999 X             1     5.26  10121 Æ
99999-9999          1     5.26  10121-2898
TOTAL              19   100.00
```

POSIT: Finding the Beginning of a Substring

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

Find the Beginning of a Substring

```
POSIT(source_string, length, substring, sublength, output)
```

where:

source_string

Alphanumeric

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

length

Integer

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

substring

Alphanumeric

Is the substring whose position you want to find. This can be the substring enclosed in single quotation marks, or the field that contains the string.

sublength

Integer

Is the number of characters in *substring*. If this argument is less than or equal to 0, or if it is greater than *length*, the function returns a 0.

output

Integer

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

Finding the Position of a Letter

POSIT determines the position of the first capital letter I in LAST_NAME and stores the result in I_IN_NAME:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2');
WHERE DEPARTMENT EQ 'PRODUCTION'
END
```

The output is:

```
LAST_NAME        I_IN_NAME
---------        ---------
STEVENS                  0
```

```
SMITH                    3
BANNING                  5
IRVING                   1
ROMANS                   0
MCKNIGHT                 5
```

REVERSE: Reversing the Characters in a String

The REVERSE function reverses the characters in a string. This reversal includes all trailing blanks, which then become leading blanks. However, in an HTML report with SET SHOWBLANKS=OFF (the default value), the leading blanks are not visible.

Reverse the Characters in a String

```
REVERSE(length, source_string, output)
```

where:

length

Integer

Is the number of characters in **source_string** and **output**, or a field that contains the length.

source_string

Alphanumeric

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

output

Alphanumeric

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

Reversing the Characters in a String

In the following request against the EMPLOYEE data source, the REVERSE function is used to reverse the characters in the LAST_NAME field to produce the field named REVERSE_LAST. In this field, the trailing blanks from LAST_NAME have become leading blanks. The TRIM function is used to strip the leading blanks from REVERSE_LAST to produce the field named TRIM_REVERSE:

```
DEFINE FILE EMPLOYEE
REVERSE_LAST/A15 = REVERSE(15, LAST_NAME, REVERSE_LAST);
TRIM_REVERSE/A15 = TRIM('L', REVERSE_LAST, 15, ' ', 1, 'A15');
END
TABLE FILE EMPLOYEE
PRINT REVERSE_LAST TRIM_REVERSE
BY LAST_NAME
END
```

The output is:

```
LAST_NAME        REVERSE_LAST     TRIM_REVERSE
---------        ------------     ------------
BANNING                GNINNAB    GNINNAB
BLACKWOOD            DOOWKCALB    DOOWKCALB
CROSS                    SSORC    SSORC
GREENSPAN            NAPSNEERG    NAPSNEERG
IRVING                  GNIVRI    GNIVRI
JONES                    SENOJ    SENOJ
MCCOY                    YOCCM    YOCCM
MCKNIGHT              THGINKCM    THGINKCM
ROMANS                  SNAMOR    SNAMOR
SMITH                    HTIMS    HTIMS
                         HTIMS    HTIMS
STEVENS                SNEVETS    SNEVETS
```

RJUST: Right-Justifying a Character String

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

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

Right-Justify a Character String

```
RJUST(length, source_string, output)
```

where:

length

Integer

Is the number of characters in *source_string* and *output*, or a field that contains the length. Their lengths must be the same to avoid justification problems.

source_string

Alphanumeric

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

output

Alphanumeric

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

Right-Justifying a String

RJUST right-justifies the LAST_NAME field and stores the result in RIGHT_NAME:

```
SET STYLE=OFF
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
RIGHT_NAME/A15 = RJUST(15, LAST_NAME, RIGHT_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME        RIGHT_NAME
---------        ----------
```

```
SMITH                  SMITH
JONES                  JONES
MCCOY                  MCCOY
BLACKWOOD          BLACKWOOD
GREENSPAN          GREENSPAN
CROSS                  CROSS
```

SOUNDEX: Comparing Character Strings Phonetically

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

To conduct a phonetic search, do the following:

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

Compare Character Strings Phonetically

```
SOUNDEX(length, source_string, output)
```

where:

length

Alphanumeric

Is the number of characters in *source_string*, or a field that contains the length. It can be a number enclosed in single quotation marks, or a field containing the number. The number must be from 01 to 99, expressed with two digits (for example '01'); a number larger than 99 causes the function to return asterisks (*) as output.

source_string

Alphanumeric

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

output

Alphanumeric

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

Comparing Character Strings Phonetically

The following request creates three fields:

- PHON_NAME contains the phonetic code of employee last names.
- PHON_COY contains the phonetic code of your guess, MICOY.
- PHON_MATCH contains YES if the phonetic codes match, NO if they do not.

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

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

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

The output is:

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

SPELLNM: Spelling Out a Dollar Amount

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

Spell Out a Dollar Amount

```
SPELLNM(outlength, number, output)
```

where:

outlength

Integer

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

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

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

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

number

Alphanumeric or Numeric (9.2)

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

output

Alphanumeric

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

Spelling Out a Dollar Amount

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

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

The output is:

```
CURR_SAL    AMT_IN_WORDS
--------    ------------
```

```
$13,200.00  THIRTEEN THOUSAND TWO HUNDRED DOLLARS AND NO CENTS
$18,480.00  EIGHTEEN THOUSAND FOUR HUNDRED EIGHTY DOLLARS AND NO CENTS
$18,480.00  EIGHTEEN THOUSAND FOUR HUNDRED EIGHTY DOLLARS AND NO CENTS
```

```
$21,780.00  TWENTY-ONE THOUSAND SEVEN HUNDRED EIGHTY DOLLARS AND NO
CENTS
$9,000.00   NINE THOUSAND DOLLARS AND NO CENTS
$27,062.00  TWENTY-SEVEN THOUSAND SIXTY-TWO DOLLARS AND NO CENTS
```

SQUEEZ: Reducing Multiple Spaces to a Single Space

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

Reduce Multiple Spaces to a Single Space

```
SQUEEZ(length, source_string, output)
```

where:

length

Integer

Is the number of characters in *source_string* and *output*, or a field that contains the length.

source_string

Alphanumeric

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

output

Alphanumeric

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

Reducing Multiple Spaces to a Single Space

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

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

The output is:

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

STRIP: Removing a Character From a String

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

Remove a Character From a String

```
STRIP(length, source_string, char, output)
```

where:

length

Integer

Is the number of characters in **source_string** and **output**, or a field that contains the number.

source_string

Alphanumeric

Is the string from which the character will be removed, or a field containing the string.

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 more than one character is provided, the left-most character 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.

output

Alphanumeric

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

Removing Occurrences of a Character From a String

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

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

The output is:

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

Removing Single Quotation Marks From a String

STRIP removes all occurrences of a single quotation mark (') from the TITLE field and stores the result in a field with the format A39:

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

The output is:

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

STRREP: Replacing Character Strings

The STRREP replaces all instances of a specified string within a source string. It also supports replacement by null strings.

Replace Character Strings

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

where:

inlength

Numeric

Is the number of characters in the source string.

instring

Alphanumeric

Is the source string.

searchlength

Numeric

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

searchstring

Alphanumeric

Is the character string to be replaced.

replength

Numeric

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

repstring

Alphanumeric

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

outlength

Numeric

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

output

Alphanumeric

Is the resulting output string after all replacements and padding.

Usage Note for STRREP Function

The maximum string length is 4095.

Replacing Commas and Dollar Signs

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

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

The output is:

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

```
SMITH           $22,700.00          $22700.00          USD 22700.00
STEVENS         $11,000.00          $11000.00          USD 11000.00
```

SUBSTR: Extracting a Substring

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

Extract a Substring

```
SUBSTR(length, source_string, start, end, sublength, output)
```

where:

length

Integer

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

source_string

Alphanumeric

Is the string from which to extract a substring enclosed in single quotation marks, or the field containing the parent string.

start

Integer

Is the starting position of the substring in the source string. If **start** is less than one or greater than **length**, the function returns spaces.

end

Integer

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

sublength

Integer

Is the number of characters in the substring (normally end - start + 1). If the *sublength* is longer than *end* - *start* +1, the substring is padded with trailing spaces. If it is shorter, the substring is truncated. This value should be the declared length of *output*. Only *sublength* characters will be processed.

output

Alphanumeric

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

Extracting a String

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

```
TABLE FILE EMPLOYEE
PRINT
COMPUTE
    I_IN_NAME/I2 = POSIT(LAST_NAME, 15, 'I', 1, 'I2'); AND
COMPUTE
    I_SUBSTR/A3 =
    SUBSTR(15, LAST_NAME, I_IN_NAME, I_IN_NAME+2, 3, I_SUBSTR);
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION'
END
```

The output is:

```
LAST_NAME      I_IN_NAME  I_SUBSTR
---------      ---------  --------
BANNING                5  ING
IRVING                 1  IRV
MCKNIGHT               5  IGH
```

```
ROMANS                   0
SMITH                    3  ITH
STEVENS                  0
```

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

TRIM: Removing Leading and Trailing Occurrences

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

Remove Leading and Trailing Occurrences

```
TRIM(trim_where, source_string, length, pattern, sublength, output)
```

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.

source_string

Alphanumeric

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

string_length

Integer

Is the number of characters in the source string.

pattern

Alphanumeric

Is the character string pattern to remove enclosed in single quotation marks.

sublength

Integer

Is the number of characters in the pattern.

output

Alphanumeric

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

Removing Leading Occurrences

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

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

The output is:

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

Removing Trailing Occurrences

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

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

The output is:

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

UPCASE: Converting Text to Uppercase

The UPCASE function converts a character string to uppercase. It is useful for sorting on a field that contains both mixed-case and uppercase values. Sorting on a mixed-case field produces incorrect results because the sorting sequence in EBCDIC always places lowercase letters before uppercase letters, while the ASCII sorting sequence always places uppercase letters before lowercase. To obtain correct results, define a new field with all of the values in uppercase, and sort on that field.

In FIDEL, CRTFORM LOWER retains the case of entries exactly as they were typed. Use UPCASE to convert entries for particular fields to uppercase.

Convert Text to Uppercase

```
UPCASE(length, source_string, output)
```

where:

length

Integer

Is the number of characters in *source_string* and *output*.

input

Alphanumeric

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

output

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

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

Converting a Mixed-Case String to Uppercase

UPCASE converts the LAST_NAME_MIXED field to uppercase:

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

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

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

The output is:

```
LAST_NAME_UPPER  LAST_NAME_MIXED  FIRST_NAME
---------------  ---------------  ----------
BANNING          Banning          JOHN
BLACKWOOD        BLACKWOOD        ROSEMARIE
CROSS            CROSS            BARBARA
MCCOY            MCCOY            JOHN
MCKNIGHT         Mcknight         ROGER
ROMANS           Romans           ANTHONY
```

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

Converting a Lowercase Field to Uppercase With MODIFY

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

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

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

The procedure processes as:

1. The procedure prompts you for an employee ID, last name, first name, and department on a CRTFORM screen. The CRTFORM LOWER option retains the case of

entries exactly as typed.

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

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

3. The procedure searches the data source for the ID 444555666. If it does not find the ID, it continues processing the transaction.

4. UPCASE converts the DEPARTMENT entry sales to SALES:

```
ENTER EMPLOYEE'S ID :
ENTER LAST_NAME:                   FIRST_NAME:
TYPE THE NAME EXACTLY AS YOU SEE IT ON THE SHEET
ENTER DEPARTMENT ASSIGNMENT:
DEPARTMENT VALUE CHANGED TO UPPERCASE: SALES
```

5. The procedure adds the transaction to the data source.

6. When you exit the procedure with PF3, the transaction message indicates the number of transactions accepted or rejected:

```
TRANSACTIONS:         TOTAL =     1  ACCEPTED=     1  REJECTED=
0
```

```
SEGMENTS:             INPUT =     1  UPDATED =     0  DELETED =
0
```

XMLDECOD: Decoding XML-Encoded Characters

The XMLDECOD function decodes the following five standard XML-encoded characters when they are encountered in a string:

Character Name	Character	XML-Encoded Representation
ampersand	&	&
greater than symbol	>	>
less than symbol	<	<
double quotation mark	"	"
single quotation mark (apostrophe)	'	'

Decode XML-Encoded Characters

```
XMLDECOD(inlength, source_string, outlength,  output)
```

where:

inlength

Integer

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

source_string

Alphanumeric

Is the name of the field containing the source character string or the string enclosed in single quotation marks (').

outlength

Integer

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

output

Integer

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

Decoding XML-Encoded Characters

The file XMLFUNCS is a .csv file that contains some unencoded characters and some XML-encoded characters. The Master File is:

```
FILE = XMLFUNCS, SUFFIX=COM,$
SEGNAME = SEG01, SEGTYPE=S1,$
FIELD=INSTRING, ALIAS=CHARS, USAGE=A30,ACTUAL=A30,$
```

The contents of the file follow:

```
CHARS: &  < >  ,$
ENCODED: &amp; &gt;  ,$
ENCODED: &quot; &apos;   ,$
MIXED:  &amp;  < &gt;   ,$
```

XMLDECOD decodes any of the supported XML-encoded characters. Note that some viewers automatically decode the encoded values for display, so the output is produced in a plain text format (FORMAT WP):

```
FILEDEF XMLFUNCS DISK xmlfuncs.csv
DEFINE FILE XMLFUNCS
OUTSTRING/A30=XMLDECOD(30,INSTRING,30,'A30');
END
TABLE FILE XMLFUNCS
PRINT INSTRING OUTSTRING
ON TABLE HOLD FORMAT WP
ON TABLE SET PAGE NOPAGE
```

In the output string, XML-encoded characters have been decoded, and characters that were not encoded have been left as they were in the input string:

```
  INSTRING                        OUTSTRING
  --------                        ---------
```

```
CHARS: &  < >                   CHARS: &  < >
ENCODED: &amp; &gt;             ENCODED: & >
ENCODED: &quot; &apos;          ENCODED: " '
MIXED:  &amp;  < &gt;           MIXED:  &  < >
```

XMLENCOD: XML-Encoding Characters

The XMLENCOD function encodes the following five standard characters when they are encountered in a string:

Character Name	Character	Encoded Representation
ampersand	&	&
greater than symbol	>	>
less than symbol	<	<
double quotation mark	"	"
single quotation mark (apostrophe)	'	'

XML-Encode Characters

```
XMLENCOD(inlength, source_string, option, outlength,  output)
```

where:

inlength

Integer

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

source_string

Alphanumeric

Is the name of the field containing the source character string or a string enclosed in single quotation marks (').

option

Integer

Is a code that specifies whether to process a string that already contains XML-encoded characters. Valid values are:

- 0, the default, which cancels processing of a string that already contains at least one XML-encoded character.
- 1, which processes a string that contains XML-encoded characters.

outlength

Integer

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

> **Note:** The output length, in the worst case, could be six times the length of the input.

output

Integer

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

XML-Encoding Characters

The file XMLFUNCS is a .csv file that contains some unencoded characters and some XML-encoded characters. The Master File is:

```
FILE = XMLFUNCS, SUFFIX=COM,$
SEGNAME = SEG01, SEGTYPE=S1,$
FIELD=INSTRING, ALIAS=CHARS, USAGE=A30,ACTUAL=A30,$
```

The contents of the file follow:

```
CHARS: &  < >  ,$
ENCODED: &amp; &gt;  ,$
ENCODED: &quot; &apos;   ,$
MIXED:  &amp;  < &gt;   ,$
```

XMLENCOD XML-encodes any of the supported characters to produce OUTSTRING1, and processes every input string regardless of whether it already contains XML-encoded characters. For OUTSTRING2, it only encodes those strings that do not contain any XML-encoded characters. Note that some viewers automatically decode the encoded values for display, so the output is produced in plain text format (FORMAT WP):

```
FILEDEF XMLFUNCS DISK xmlfuncs.csv
DEFINE FILE XMLFUNCS
OUTSTRING1/A30=XMLENCOD(30,INSTRING,1,30,'A30');
OUTSTRING2/A30=XMLENCOD(30,INSTRING,0,30,'A30');
END
TABLE FILE XMLFUNCS
PRINT INSTRING OUTSTRING1 IN 24 OUTSTRING2 IN 48
ON TABLE SET PAGE NOPAGE
ON TABLE HOLD FORMAT WP
END
```

In OUTSTRING1, the supported characters have been XML-encoded, and output is produced even if the input string contains encoded characters. OUTSTRING2 is only produced when no XML-encoded characters exist in the input string:

```
 INSTRING               OUTSTRING1              OUTSTRING2
 --------               ----------              ----------
 CHARS: &  < >          CHARS: &amp;  &lt; &gt; CHARS: &amp;  &lt; &gt;
 ENCODED: &amp; &gt;    ENCODED: &amp; &gt;
 ENCODED: &quot; &apos; ENCODED: &quot; &apos;
 MIXED:  &amp;  < &gt;  MIXED:  &amp;  &lt; &gt;
```

Variable Length Character Functions

The character format A*n*V is supported in synonyms for FOCUS, XFOCUS, and relational data sources. This format is used to represent the VARCHAR (variable length character) data types supported by relational database management systems.

Overview

For relational data sources, A*n*V keeps track of the actual length of a VARCHAR column. This information is especially valuable when the value is used to populate a VARCHAR column in a different RDBMS. It affects whether trailing blanks are retained in string concatenation and, for Oracle, string comparisons (the other relational engines ignore trailing blanks in string comparisons).

In a ibi™ FOCUS® or XFOCUS data source, A*n*V does not provide true variable length character support. It is a fixed-length character field with an extra two leading bytes to contain the actual length of the data stored in the field. This length is stored as a short integer value occupying two bytes. Because of the two bytes of overhead and the additional processing required to strip them, the A*n*V format is *not* recommended for use with non-relational data sources.

A*n*V fields can be used as arguments to all supplied functions that expect alphanumeric arguments. An A*n*V input parameter is treated as an A*n* parameter and is padded with blanks to its declared size (*n*). If the last parameter specifies an A*n*V format, the function result is converted to type A*n*V with the actual length set equal to its size.

The functions described in this topic are designed to work specifically with the A*n*V data type parameters.

Usage Notes for Using an AnV Field in a Function

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

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

LENV: Returning the Length of an Alphanumeric Field

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

Find the Length of an Alphanumeric Field

```
LENV(source_string, output)
```

where:

source_string

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

Is the source string or field. If it is an A*n* format field, the function returns its size, *n*. For a character string enclosed in quotation marks or a variable, the size of the string or variable is returned. For a field of A*n*V format, its length, taken from the length-in-bytes of the field, is returned.

output

Integer

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

Finding the Length of an A*n*V Field

TRIMV creates an A*n*V field named TITLEV by removing trailing blanks from the TITLE value. Then LENV returns the actual length of each instance of TITLEV to the ALEN field:

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

The output is:

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

LOCASV: Creating a Variable Length Lowercase String

The LOCASV function converts alphabetic characters in the source string to lowercase and is similar to LOCASE. LOCASV returns A*n*V output whose actual length is the lesser of the actual length of the A*n*V source string and the value of the input parameter upper_limit.

Create a Variable Length Lowercase String

```
LOCASV(upper_limit, source_string, output)
```

where:

upper_limit

Integer

Is the limit for the length of the source string.

source_string

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

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

output

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

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

If the output format is A*n*V, the actual length returned is equal to the smaller of the source string length and the upper limit.

Creating a Variable Length Lowercase String

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

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND COMPUTE
LOWCV_NAME/A15V = LOCASV(5, LAST_NAME, LOWCV_NAME);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

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

POSITV: Finding the Beginning of a Variable Length Substring

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

Find the Beginning of a Variable Length Substring

```
POSITV(source_string, upper_limit, substring, sub_limit, output)
```

where:

source_string

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

Is the source string that contains the substring whose position you want to find. It can be the string enclosed in single quotation marks ('), or a field or variable that contains the source string. If it is a field of A*n*V format, its length is taken from the length bytes stored in the field. If the *upper_limit* is smaller than the actual length, the source string is truncated to this upper limit.

upper_limit

Integer

Is a limit for the length of the source string.

substring

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

Is the substring whose position you want to find. This can be the substring enclosed in single quotation marks ('), or the field that contains the string. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If the *sub_limit* is smaller than the actual length, the source string is truncated to this limit.

sub_limit

Integer

Is the limit for the length of the substring.

output

Integer

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

Finding the Starting Position of a Variable Length Pattern

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

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

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

The output is:

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

SUBSTV: Extracting a Variable Length Substring

The SUBSTV function extracts a substring from a string and is similar to SUBSTR. However, the end position for the string is calculated from the starting position and the substring length. Therefore, it has fewer parameters than SUBSTR. Also, the actual length of the output field, if it is an A*n*V field, is determined based on the substring length.

Extract a Variable Length Substring

```
SUBSTV(upper_limit, source_string, start, sub_limit, output)
```

where:

upper_limit

Integer

Is the limit for the length of the source string.

source_string

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

Is the character string that contains the substring you want to extract. It can be the string enclosed in single quotation marks ('), or the field containing the string. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length bytes stored in the field. If the *upper_limit* is smaller than the actual length, the source string is truncated to the upper limit. The final length value determined by this comparison is referred to as *p_length* (see the description of the *output* parameter for related information).

start

Integer

Is the starting position of the substring in the source string. The starting position can exceed the source string length, which results in spaces being returned.

sub_limit

Integer

Is the length, in characters, of the substring. Note that the ending position can exceed the input string length depending on the provided values for *start* and *sub_limit*.

output

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

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

If the format of **output** is A*n*V, and assuming *end* is the ending position of the substring, the actual length, *outlen*, is computed as follows from the values for *end*, *start*, and *p_length* (see the *source_string* parameter for related information):

If *end* > *p_length* or *end* < *start*, then outlen = 0. Otherwise, outlen = *end* - *start* + 1.

Extracting a Variable Length Substring

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

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

The output is:

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

TRIMV: Removing Characters From a String

The TRIMV function removes leading and/or trailing occurrences of a pattern within a character string. TRIMV is similar to TRIM. However, TRIMV allows the source string and the pattern to be removed to have A*n*V format.

TRIMV is useful for converting an A*n* field to an A*n*V field (with the length in bytes containing the actual length of the data up to the last non-blank character).

Remove Characters From a String

```
TRIMV(trim_where, source_string, upper_limit, pattern, pattern_limit,
output)
```

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.

source_string

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

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

upper_limit

Integer

Is the upper limit for the length of the source string.

pattern

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

Is the pattern to remove from the string, enclosed in single quotation marks ('). If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length in bytes stored in the field. If the *pattern_limit* is smaller than the actual length, the pattern is truncated to this limit.

plength_limit

Integer

Is the limit for the length of the pattern.

output

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

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

If the output format is A*n*V, the length is set to the number of characters left after trimming.

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

TRIMV creates an A*n*V field named TITLEV by removing trailing blanks from the TITLE value:

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

Here are the first 10 lines of the output:

```
CATEGORY  DIRECTOR         TITLEV
--------  ---------        ------
ACTION    SPIELBERG S.     JAWS
          VERHOVEN P.      ROBOCOP
          VERHOVEN P.      TOTAL RECALL
          SCOTT T.         TOP GUN
          MCDONALD P.      RAMBO III
```

```
CHILDREN                    SMURFS, THE
          BARTON C.         SHAGGY DOG, THE
                            SCOOBY-DOO-A DOG IN THE RUFF
          GEROMINI          ALICE IN WONDERLAND
                            SESAME STREET-BEDTIME STORIES AND SONGS
```

UPCASV: Creating a Variable Length Uppercase String

UPCASV converts alphabetic characters to uppercase, and is similar to UPCASE. However, UPCASV can return A*n*V output whose actual length is the lesser of the actual length of the A*n*V source string and an input parameter that specifies the upper limit.

Create a Variable Length Uppercase String

```
UPCASV(upper_limit, source_string, output)
```

where:

upper_limit

Integer

Is the limit for the length of the source string. It can be a positive constant or a field whose integer portion represents the upper limit.

source_string

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

is the string to convert to uppercase. It can be the character string enclosed in single quotation marks ('), or the field containing the character string. If it is a field, it can have A*n* or A*n*V format. If it is a field of type A*n*V, its length is taken from the length in bytes stored in the field. If the *upper_limit* is smaller than the actual length, the source string is truncated to the upper limit.

output

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

Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks ('). This can be a field with A*n*V or A*n* format.

If the output format is A*n*V, the length returned is equal to the smaller of the source string length and *upper_limit*.

Creating a Variable Length Uppercase String

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

```
DEFINE FILE EMPLOYEE
LAST_NAME_MIXED/A15=IF DEPARTMENT EQ 'MIS' THEN LAST_NAME ELSE
LCWORD(15, LAST_NAME, 'A15');
LAST_NAME_UPCASV/A15V=UPCASV(5, LAST_NAME_MIXED, 'A15') ;
END
```

Suppose you execute a request that sorts by this field:

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

The output is:

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

Character Functions for DBCS Code Pages

The functions in this topic manipulate strings of DBCS and SBCS characters when your configuration uses a DBCS code page.

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

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

The DCTRAN function can translate single-byte to double-byte characters and double-byte to single-byte characters, as well as single-byte to single-byte characters and double-byte to double-byte characters.

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

```
DCTRAN(length, source_string, indecimal, outdecimal, output)
```

where:

length

Double

Is the number of characters in **source_string**.

source_string

Alphanumeric

Is the character string to be translated.

indecimal

Double

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

outdecimal

Double

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

output

Alphanumeric

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

Using DCTRAN to Translate Double-Byte Characters

In the following:

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

```
For AｱA本B語, the result is AFA本B語.
```

DEDIT: Extracting or Adding Characters

If your configuration uses a DBCS code page, you can use the DEDIT function to extract characters from or add characters to a string.

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

Extract or Add DBCS or SBCS Characters

```
DEDIT(inlength, source_string, mask_length, mask, output)
```

where:

inlength

Integer

Is the number of *bytes* in *source_string*. The string can have a mixture of DBCS and SBCS characters. Therefore, the number of bytes represents the maximum number of characters possible in the source string.

source_string

Alphanumeric

Is the string to edit enclosed in single quotation marks ('), or the field containing the string.

mask_length

Integer

Is the number of *characters* in mask.

mask

Alphanumeric

Is the string of mask characters.

Each nine (9) in the mask causes the corresponding character from the source field to be copied to the new field.

Each dollar sign ($) in the mask causes the corresponding character in the source field to be ignored.

Any other character in the mask is copied to the new field.

output

Alphanumeric

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

Adding and Extracting DBCS Characters

The following example copies alternate characters from the source string to the new field, starting with the first character in the source string, and then adds several new characters at the end of the extracted string:

```
DEDIT( 15, 'あaいiうuえeおo', 16, '9$9$9$9$9$-かきくけこ', 'A30')
The result is あいうえお-かきくけこ.
```

The following example copies alternate characters from the source string to the new field, starting with the second character in the source string, and then adds several new characters at the end of the extracted string:

```
DEDIT( 15, 'あaいiうuえeおo', 16, '$9$9$9$9$9-ABCDE', 'A20')
The result is aiueo-ABCDE.
```

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

The DSTRIP function removes all occurrences of a specific single-byte or double-byte character from a string. The resulting character string has the same length as the original string, but is padded on the right with spaces.

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

```
DSTRIP(length, source_string, char, output)
```

where:

length

Double

Is the number of characters in *source_string* and *outfield*.

source_string

Alphanumeric

Is the string from which the character will be removed.

char

Alphanumeric

Is the character to be removed from the string. If more than one character is provided, the left-most character 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.

output

Alphanumeric

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

Removing a Double-Byte Character From a String

In the following:

```
DSTRIP(9, 'A日A本B語', '日', A9)
```

```
For A日A本B語, the result is AA本B語.
```

DSUBSTR: Extracting a Substring

If your configuration uses a DBCS code page, you can use the DSUBSTR function to extract a substring based on its length and position in the source string.

Extract a Substring

```
DSUBSTR(inlength, source_string, start, end, sublength, output)
```

where:

inlength

Integer

Is the length of the source string in *bytes*, or a field that contains the length. The string can have a mixture of DBCS and SBCS characters. Therefore, the number of bytes represents the maximum number of characters possible in the source string.

source_string

Alphanumeric

Is the string from which the substring will be extracted enclosed in single quotation marks ('), or the field containing the parent string.

start

Integer

Is the starting position (in number of *characters*) of the substring in the source string. If this argument is less than one or greater than *end*, the function returns spaces.

end

Integer

Is the ending position (in number of *characters*) 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, in *characters* (normally *end* - *start* + 1). If the *sublength* is longer than *end* - *start* +1, the substring is padded with trailing spaces. If it is shorter, the substring is truncated. This value should be the declared length of *output*. Only *sublength* characters will be processed.

output

Alphanumeric

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

Extracting a Substring

The following example extracts the 3-character substring in positions 4 through 6 from a 15-byte string of characters:

```
DSUBSTR( 15, 'あaいiうuえeおo', 4, 6, 3, 'A10')
The result is iうu.
```

JPTRANS: Converting Japanese Specific Characters

The JPTRANS function converts Japanese specific characters.

Convert Japanese Specific Characters

```
JPTRANS ('type_of_conversion', length, source_string, 'output_format')
```

where:

type_of_conversion

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

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

Conversion Type	Description
`'KATAHIRA'`	Converts Zenkaku Katakana to Hiragana.
`'930TO939'`	Converts codepage from 930 to 939.
`'939TO930'`	Converts codepage from 939 to 930.

length

Integer

Is the number of characters in the source_string.

source_string

Alphanumeric

Is the string to convert.

output_format

Alphanumeric

Is the name of the field that contains the output, or the format enclosed in single quotation marks (').

Using the JPTRANS Function

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

For ａｂｃ, the result is ＡＢＣ.

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

For ＡＢＣ, the result is ａｂｃ.

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

For AaBbCc123, the result is ＡａＢｂＣｃ１２３.

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

For !@$%,.?, the result is ！＠＄％、。？

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

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

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

For ｱｲｳ, the result is ｱ　ｲ　ｳ

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

For ＡａＢｂＣｃ１２３, the result is AaBbCc123.

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

For ！＠＄％、。？, the result is !@$%,.?

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

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

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

For ア　イ　ウ, the result is ア イ ウ

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

For あいう, the result is アイウ

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

For アイウ, the result is あいう

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

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

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

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

Usage Notes for the JPTRANS Function

- HNZNSIGN and ZNHNSIGN focus on the conversion of symbols.

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

  HNZNSIGN:

  - Double Quote " (U+0022) -> Fullwidth Right Double Quote ” (U+201D)
  - Single Quote ' (U+0027) -> Fullwidth Right Single Quote ’ (U+2019)
  - Comma , (U+002C) -> Fullwidth Ideographic Comma (U+3001)
  - Full Stop . (U+002E) -> Fullwidth Ideographic Full Stop ? (U+3002)
  - Backslash \ (U+005C) -> Fullwidth Backslash \ (U+FF3C)

- Halfwidth Left Corner Bracket (U+FF62) -> Fullwidth Left Corner Bracket (U+300C)
- Halfwidth Right Corner Bracket (U+FF63) -> Fullwidth Right Corner Bracket (U+300D)
- Halfwidth Katakana Middle Dot ? (U+FF65) -> Fullwidth Middle Dot · (U+30FB)

ZNHNSIGN:

- Fullwidth Right Double Quote ” (U+201D) -> Double Quote " (U+0022)
- Fullwidth Left Double Quote “ (U+201C) -> Double Quote " (U+0022)
- Fullwidth Quotation " (U+FF02) -> Double Quote " (U+0022)
- Fullwidth Right Single Quote ’ (U+2019) -> Single Quote ' (U+0027)
- Fullwidth Left Single Quote ‘ (U+2018) -> Single Quote ' (U+0027)
- Fullwidth Single Quote ' (U+FF07) -> Single Quote ' (U+0027)
- Fullwidth Ideographic Comma (U+3001) -> Comma , (U+002C)
- Fullwidth Comma , (U+FF0C) -> Comma , (U+002C)
- Fullwidth Ideographic Full Stop ? (U+3002) -> Full Stop . (U+002E)
- Fullwidth Full Stop . (U+FF0E) -> Full Stop . (U+002E)
- Fullwidth Yen Sign ¥ (U+FFE5) -> Yen Sign ¥ (U+00A5)
- Fullwidth Backslash \ (U+FF3C) -> Backslash \ (U+005C)
- Fullwidth Left Corner Bracket (U+300C) -> Halfwidth Left Corner Bracket (U+FF62)
- Fullwidth Right Corner Bracket (U+300D) -> Halfwidth Right Corner Bracket (U+FF63)
- Fullwidth Middle Dot · (U+30FB) -> Halfwidth Katakana Middle Dot · (U+FF65)

- HNZNKANA and ZNHNKANA focus on the conversion of Katakana

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

  - Fullwidth Ideographic Comma (U+3001) <-> Halfwidth Ideographic Comma (U+FF64)
  - Fullwidth Ideographic Full Stop (U+3002) <-> Halfwidth Ideographic Full Stop

(U+FF61)

  - Fullwidth Left Corner Bracket (U+300C) <-> Halfwidth Left Corner Bracket (U+FF62)
  - Fullwidth Right Corner Bracket (U+300D) <-> Halfwidth Right Corner Bracket (U+FF63)
  - Fullwidth Middle Dot · (U+30FB) <-> Halfwidth Katakana Middle Dot · (U+FF65)
  - Fullwidth Prolonged Sound (U+30FC) <-> Halfwidth Prolonged Sound (U+FF70)

- JPTRANS can be nested for multiple conversions.

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

  For バンゴウ＃１２３, the result is バンゴウ#123

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

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

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

KKFCUT: Truncating a String

If your configuration uses a DBCS code page, you can use the KKFCUT function to truncate a string.

Truncate a String

```
KKFCUT(length, source_string, output)
```

where:

length

Integer

Is the length of the source string in *bytes*, or a field that contains the length. The string can have a mixture of DBCS and SBCS characters. Therefore, the number of bytes represents the maximum number of characters possible in the source string.

source_string

Alphanumeric

Is the string that will be truncated enclosed in single quotation marks ('), or the field containing the string.

output

Alphanumeric

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

The string will be truncated to the number of bytes in the output field.

Truncating a String

In the following, KKFCUT truncates the COUNTRY field (up to 10 bytes long) to A4 format:

```
COUNTRY_CUT/A4 = KKFCUT(10, COUNTRY, 'A4');
```

The output in ASCII environments is shown in the following image:

```
国名        COUNTRY_CUT
----        -----------
イギリス    イギ
日本        日本
イタリア    イタ
ドイツ      ドイ
フランス    フラ
```

The output in EBCDIC environments is shown in the following image:

国名	COUNTRY_CUT
イギリス	イ
日本	日
イタリア	イ
ドイツ	ド
フランス	フ

SFTDEL: Deleting the Shift Code From DBCS Data

If your configuration uses a DBCS code page, you can use the SFTDEL function to delete the shift code from DBCS data.

Delete the Shift Code From DBCS Data

```
SFTDEL(source_string, length, output)
```

where:

source_string

Alphanumeric

Is the string from which the shift code will be deleted enclosed in single quotation marks ('), or the field containing the string.

length

Integer

Is the length of the source string in *bytes*, or a field that contains the length. The string can have a mixture of DBCS and SBCS characters. Therefore, the number of bytes represents the maximum number of characters possible in the source string.

output

Alphanumeric

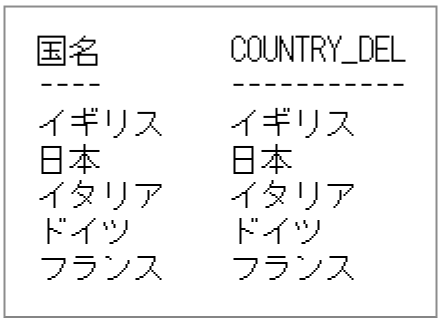
Is the field to which the result is returned, or the format of the output value enclosed in single quotation marks (').

Deleting the Shift Code From a String

In the following, SFTDEL deleted the shift code from the COUNTRY field (up to 10 bytes long):

```
COUNTRY_DEL/A10 = SFTDEL(COUNTRY, 10, 'A10');
```

The output in ASCII environments is shown in the following image:



```
国名        COUNTRY_DEL
----        -----------
イギリス    イギリス
日本        日本
イタリア    イタリア
ドイツ      ドイツ
フランス    フランス
```

The output in EBCDIC environments is shown in the following image:

```
 国名       COUNTRY_DEL
------      -----------
 イギリス   ｢b｢A｢ﾒ｢ﾇ
 日本       、ｲ、ｶ
 イタリア   ｢b｢j｢ﾒ｢a
 ドイツ     ｢｢b｢]
 フランス   ｢ﾎ｢[｢]｢ﾇ
```

SFTINS: Inserting the Shift Code Into DBCS Data

If your configuration uses a DBCS code page, you can use the SFTINS function to insert the shift code into DBCS data.

Insert the Shift Code Into DBCS Data

```
SFTINS(source_string, length, output)
```

where:

source_string

Alphanumeric

Is the string into which the shift code will be inserted enclosed in single quotation marks ('), or the field containing the string.

length

Integer

Is the length of the source string in *bytes*, or a field that contains the length. The string can have a mixture of DBCS and SBCS characters. Therefore, the number of bytes represents the maximum number of characters possible in the source string.

output

Alphanumeric

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

SFTINS: Inserting the Shift Code Into a String

In the following example, SFTINS inserts the shift code into the COUNTRY_DEL field (which is the COUNTRY field with the shift code deleted):

```
COUNTRY_INS/A10 = SFTINS(COUNTRY_DEL, 10, 'A10');
```

The output displays the original COUNTRY field, the COUNTRY_DEL field with the shift code deleted, and the COUNTRY_INS field with the shift code re-inserted.

The output in ASCII environments, is shown in the following image:

```
国名          COUNTRY_DEL   COUNTRY_INS
----          -----------   -----------
イギリス      イギリス      イギリス
日本          日本          日本
イタリア      イタリア      イタリア
ドイツ        ドイツ        ドイツ
フランス      フランス      フランス
```

The output in EBCDIC environments is shown in the following image:

```
  国名        COUNTRY_DEL   COUNTRY_INS
 ------       -----------   -----------
 イギリス     ｢b｢A｢ﾒ｢ｽ      イギリス
 日本         ､ｲ､ｶ           日本
 イタリア     ｢b｢j｢ﾒ｢a       イタリア
 ドイツ       ｢｢b｢l          ドイツ
 フランス     ｢ﾎ｢[｢]｢ｽ       フランス
```

Data Source and Decoding Functions

Data source and decoding functions search for data source records, retrieve data source records or values, and assign values based on the value of an input field.

The result of a data source function must be stored in a field. The result cannot be stored in a Dialogue Manager variable.

For many functions, the *output* argument can be supplied either as a field name or as a format enclosed in single quotation marks (‘). However, if a function is called from a Dialogue Manager command, this argument must always be supplied as a format. For detailed information about calling a function and supplying arguments, see Accessing and Calling a Function.

CHECKMD5: Computing an MD5 Hash Check Value - Computes an MD5 hash check value

CHECKSUM: Computing a Hash Sum - Computes a hash sum

COALESCE: Returning the First Non-Missing Value - Returns the first non-missing value

DB_EXPR: Inserting an SQL Expression Into a Request - Inserts an SQL expression into a request

DB_INFILE: Testing Values Against a File or an SQL Subquery - Tests values against a file or an SQL subquery

DB_LOOKUP: Retrieving Data Source Values - Retrieves data source values

DECODE: Decoding Values - Decodes value

FIND: Verifying the Existence of a Value in a Data Source - Verifies the existence of a value in a data source

IMPUTE: Replacing Missing Values With Aggregated Values - Replaces missing values with aggregated values

LAST: Retrieving the Preceding Value - Retrieves the preceding value

LOOKUP: Retrieving a Value From a Cross-referenced Data Source - Retrieves a value from a cross-referenced data source

NULLIF: Returning a Null Value When Parameters Are Equal - returns a null value when parameters are equal

FOCUSPAGES: Finding the size of a FOCUS or XFOCUS file - Finds the size of a FOCUS or a XFOCUS file

CHECKMD5: Computing an MD5 Hash Check Value

CHECKMD5 takes an alphanumeric input value and returns a 128-bit value in a fixed-length alphanumeric string, using the MD5 hash function. A hash function is any function that can be used to map data of arbitrary size to data of fixed size. The values returned by a hash function are called hash values. They can be used for assuring the integrity of transmitted data.

Compute an MD5 Hash Check Value

```
CHECKMD5(buffer)
```

where:

buffer

Is a data buffer whose hash value is to be calculated. It can be a set of data of different types presented as a single field, or a group field in one of the following data type formats: An, AnV, or TXn.

Calculating an MD5 Hash Check Value

The following request calculates an MD5 hash check value and converts it to an alphanumeric hexadecimal value for display.

```
DEFINE FILE WFLITE
MD5/A32 = HEXTYPE(CHECKMD5(PRODUCT_CATEGORY));
END
TABLE FILE WFLITE
SUM MD5
BY PRODUCT_CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT,FONT=COURIER,$
```

```
 ENDSTYLE
END
```

The output is shown in the following image. The monospaced font shows that although the input values have varying length, the output has a fixed length.

Product Category	MD5
Accessories	98EDB85B00D9527AD5ACEBE451B3FAE6
Camcorder	612A923BDD05C2231F81991B8D12A3A1
Computers	45888A4DA062F16A099A7F7C6CC15EE0
Media Player	D34BEA29F24AF9FDE2E10B3E1D857CF9
Stereo Systems	3AA9FFE9806E269A7EB066A84092F0A3
Televisions	A3B5BC99DD2B42627EF64A4FCAAAB0B2
Video Production	60913E95848330A2C4A5D921E7C8BB14

CHECKSUM: Computing a Hash Sum

CHECKSUM computes a hash sum, called the checksum, of its input parameter, as a whole number in format I11. This can be used for equality search of the fields. A checksum is a hash sum used to ensure the integrity of a file after it has been transmitted from one storage device to another.

Compute a CHECKSUM Hash Value

```
CHECKSUM(buffer)
```

where:

buffer

Is a data buffer whose hash index is to be calculated. It can be a set of data of different types presented as a single field, in one of the following data type formats: An, AnV, or TXn.

Calculating a CHECKSUM Hash Value

The following request computes a checksum hash value.

```
DEFINE FILE WFLITE
CHKSUM/I11 = (CHECKSUM(PRODUCT_CATEGORY));
END
TABLE FILE WFLITE
PRINT CHKSUM
BY PRODUCT_CATEGORY
WHERE PRODUCT_CATEGORY NE LAST PRODUCT_CATEGORY
ON TABLE SET PAGE NOLEAD
END
```

The output is shown in the following image.

Product Category	CHKSUM
Accessories	-830549649
Camcorder	-912058982
Computers	-469201037
Media Player	-1760917009
Stereo Systems	-1853215244
Televisions	810407163
Video Production	275494446

COALESCE: Returning the First Non-Missing Value

Given a list of arguments, COALESCE returns the value of the first argument that is not missing. If all argument values are missing, it returns a missing value if MISSING is ON. Otherwise it returns a default value (zero or blank).

Return the First Non-Missing Value

```
COALESCE(arg1, arg2, ...)
```

where:

arg1, arg2, ...

Any field, expression, or constant. The arguments should all be either numeric or alphanumeric.

Are the input parameters that are tested for missing values.

The output data type is the same as the input data types.

Returning the First Non-Missing Value

This example uses the SALES data source with missing values added. The missing values are added by the following procedure named SALEMISS:

```
MODIFY FILE SALES
 FIXFORM STORE/4 DATE/5 PROD/4
 FIXFORM UNIT/3 RETAIL/5 DELIVER/3
 FIXFORM OPEN/3 RETURNS/C2 DAMAGED/C2
 MATCH STORE
    ON NOMATCH REJECT
    ON MATCH CONTINUE
 MATCH DATE
    ON NOMATCH REJECT
    ON MATCH CONTINUE
 MATCH PROD_CODE
    ON NOMATCH INCLUDE
    ON MATCH REJECT
DATA
14Z 1017 C13 15 1.99 35 30   6
14Z 1017 C14 18 2.05 30 25 4
14Z 1017 E2  33 0.99 45 40
END
```

The following request uses COALESCE to return the first non-missing value:

```
TABLE FILE SALES
PRINT DAMAGED RETURNS RETAIL_PRICE
COMPUTE
COAL1/D12.2 MISSING ON = COALESCE(DAMAGED, RETURNS, RETAIL_PRICE);
BY STORE_CODE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
```

```
ENDSTYLE
END
```

The output is shown in the following image. The value of DAMAGED is returned, if it is not missing. If DAMAGED is missing, the value of RETURNS is returned, if it is not missing. If they are both missing, the value of RETAIL_PRICE is returned.

STORE_CODE	DAMAGED	RETURNS	RETAIL_PRICE	COAL1
14B	6	10	$.95	6.00
	3	3	$1.29	3.00
	1	2	$1.89	1.00
	0	3	$1.99	.00
	4	5	$2.39	4.00
	0	0	$2.19	.00
	4	9	$.99	4.00
	9	8	$1.09	9.00
14Z	3	2	$.85	3.00
	1	2	$1.89	1.00
	1	0	$1.99	1.00
	6	.	$1.99	6.00
	.	4	$2.05	4.00
	0	0	$2.09	.00
	2	3	$2.09	2.00
	7	4	$.89	7.00
	.	.	$.99	.99
	2	4	$1.09	2.00
77F	1	1	$2.09	1.00
	0	0	$2.49	.00
K1	0	1	$1.49	.00
	1	1	$.99	1.00

DB_EXPR: Inserting an SQL Expression Into a Request

The DB_EXPR function inserts a native SQL expression exactly as entered into the native SQL generated for a FOCUS or SQL language request.

The DB_EXPR function can be used in a DEFINE command, a DEFINE in a Master File, a WHERE clause, a FILTER FILE command, a filter in a Master File, or in an SQL statement. It can be used in a COMPUTE command if the request is an aggregate request (uses the SUM, WRITE, or ADD command) and has a single display command. The expression must return a single value.

Insert an SQL Expression Into a Request With DB_EXPR

```
DB_EXPR(native_SQL_expression)
```

where:

native_SQL_expression

Is a partial native SQL string that is valid to insert into the SQL generated by the request. The SQL string must have double quotation marks (") around each field reference, unless the function is used in a DEFINE with a WITH phrase.

Usage Notes for the DB_EXPR Function

- The expression must return a single value.
- Any request that includes one or more DB_EXPR functions must be for a synonym that has a relational SUFFIX.
- Field references in the native SQL expression must be within the current synonym context.
- The native SQL expression must be coded inline. SQL read from a file is not supported.

Inserting the DB2 BIGINT and CHAR Functions Into a TABLE Request

The following TABLE request against the WF_RETAIL data source uses the DB_EXPR function in the COMPUTE command to call two DB2 functions. It calls the BIGINT function to convert the squared revenue to a BIGINT data type, and then uses the CHAR function to convert that value to alphanumeric.

```
TABLE FILE WFLITE
SUM REVENUE NOPRINT
```

```
AND COMPUTE BIGREV/A31 = DB_EXPR(CHAR(BIGINT("REVENUE" * "REVENUE") ) )
; AS 'Alpha Square Revenue'
BY REGION
ON TABLE SET PAGE NOPAGE
END
```

The trace shows that the expression from the DB_EXPR function was inserted into the DB2 SELECT statement:

```
  SELECT
  T11."REGION",
   SUM(T1."Revenue"),
   ((CHAR(BIGINT( SUM(T1."Revenue") *  SUM(T1."Revenue")) ) ))
   FROM
  wrd_fact_sales T1,
  wrd_dim_customer T5,
  wrd_dim_geography T11
   WHERE
  (T5."ID_CUSTOMER" = T1."ID_CUSTOMER") AND
  (T11."ID_GEOGRAPHY" = T5."ID_GEOGRAPHY")
   GROUP BY
  T11."REGION  "
   ORDER BY
  T11."REGION  "
   FOR FETCH ONLY;
END
```

The output is:

Region	Alpha Square Revenue
Central	459024717717929
MidEast	61720506151994
NorthEast	247772056471221
NorthWest	42335175855351
SouthEast	205820846242532
SouthWest	9449541537794
West	164356565757257

DB_INFILE: Testing Values Against a File or an SQL Subquery

The DB_INFILE function compares one or more field values in a source file to values in a target file. The comparison can be based on one or more field values. DB_INFILE returns the value 1 (TRUE) if the set of source fields matches a set of values from the target file. Otherwise, the function returns 0 (zero, FALSE). DB_INFILE can be used where a function is valid in a FOCUS request, such as in a DEFINE or a WHERE phrase.

The target file can be any data source that FOCUS can read. Depending on the data sources accessed and the components in the request, either FOCUS or an RDBMS will process the comparison of values.

If FOCUS processes the comparison, it reads the target data source and dynamically creates a sequential file containing the target data values, along with a synonym describing the data file. It then builds IF or WHERE structures in memory with all combinations of source and target values. If the target data contains characters that FOCUS considers wildcard characters, it will treat them as wildcard characters unless the command SET EQTEST = EXACT is in effect.

The following situations exist when a relational data source is the source file:

- **The target values are in a relational data source from the same RDBMS and connection.** In this case, the target file referenced by DB_INFILE can be:
    - An SQL file containing a subquery that retrieves the target values. A synonym must exist that describes the target SQL file. The Access File must specify the CONNECTION and DATASET for the target file.

      If the subquery results in a SELECT statement supported by the RDBMS, the relational adapter inserts the subquery into the WHERE predicate of the generated SQL.

      If the subquery does not result in a valid SELECT statement for the RDBMS, the relational adapter retrieves the target values. It then generates a WHERE predicate, with a list of all combinations of source and target field values.

      You can create an SQL file containing a subquery and a corresponding synonym using the HOLD FORMAT SQL_SCRIPT command. For more information, see the *ibi™ FOCUS® Creating Reports* manual.

  - A relational data source. A synonym must exist that describes the target data source.

    If the data source contains only those fields referenced by DB_INFILE as target fields, the relational adapter creates a subquery that retrieves the target values. If the subquery results in a SELECT statement supported by the RDBMS, the relational adapter inserts the subquery into the WHERE predicate of the generated SQL.

    If the subquery does not result in a valid SELECT statement for the RDBMS, the relational adapter retrieves a unique list of the target values. It then generates a WHERE predicate with a list of all combinations of source and target field values.

- **The target values are in a non-relational data source or a relational data source from a different RDBMS or connection.** In this case, the target values are retrieved and passed to FOCUS for processing.

Compare Source and Target Field Values With DB_INFILE

```
DB_INFILE(target_file, s1, t1, ... sn, tn)
```

where:

target_file

Is the synonym for the target file.

s1, ..., sn

Are fields from the source file.

t1, ..., tn

Are fields from the target file.

The function returns the value 1 if a set of target values matches the set of source values. Otherwise, the function returns a zero (0).

Usage Notes for DB_INFILE

- If both the source and target data sources have MISSING=ON for a comparison field, then a missing value in both files is considered an equality. If MISSING=OFF in one or both files, a missing value in one or both files results in an inequality.

- Values are not padded or truncated when compared, except when comparing date and date-time values.
    - If the source field is a date field and the target field is a date-time field, the time component is removed before comparison.
    - If the source field is a date-time field and the target field is a date field, a zero time component is added to the target value before comparison.
- If an alphanumeric field is compared to a numeric field, an attempt will be made to convert the alphanumeric value to a number before comparison.
- If FOCUS processes the comparison, and the target data contains characters that FOCUS considers wildcard characters, it will treat them as wildcard characters unless the command SET EQTEST = EXACT is in effect.

Comparing Source and Target Values Using an SQL Subquery File

This example uses the WF_RETAIL DB2 data source.

The SQL file named retail_subquery.sql contains the following subquery that retrieves specified state codes in the Central and NorthEast regions:

```
SELECT  MAX(T11.REGION), MAX(T11.STATECODE)  FROM wrd_dim_geography T11
WHERE (T11.STATECODE IN('AR', 'IA', 'KS', 'KY', 'WY', 'CT', 'MA', 'NJ',
'NY', 'RI')) AND (T11.REGION IN('Central', 'NorthEast'))  GROUP BY
T11.REGION, T11.STATECODE
```

The retail_subquery.mas Master File follows:

```
FILENAME=RETAIL_SUBQUERY, SUFFIX=DB2     , $
  SEGMENT=RETAIL_SUBQUERY, SEGTYPE=S0, $
    FIELDNAME=REGION, ALIAS=E01, USAGE=A15V, ACTUAL=A15V,
      MISSING=ON, $
    FIELDNAME=STATECODE, ALIAS=E02, USAGE=A2, ACTUAL=A2,
      MISSING=ON, $
```

The retail_subquery.acx Access File follows:

```
SEGNAME=RETAIL_SUBQUERY, CONNECTION=CON1, DATASET=RETAIL_SUBQUERY.SQL, $
```

Note: You can create an SQL subquery file, along with a corresponding synonym, using the HOLD FORMAT SQL_SCRIPT command. For more information, see the *ibi™ FOCUS® Creating Reports* manual.

The following request uses the DB_INFILE function to compare region names and state codes against the names retrieved by the subquery:

```
TABLE FILE WFLITE
SUM REVENUE
BY REGION
BY STATECODE
WHERE DB_INFILE(RETAIL_SUBQUERY, REGION, REGION, STATECODE, STATECODE)
ON TABLE SET PAGE NOPAGE
END
```

The trace shows that the subquery was inserted into the WHERE predicate in the generated SQL:

```
 SELECT
  T11."REGION",
  T11."STATECODE",
   SUM(T1."Revenue")
   FROM
  wrd_fact_sales T1,
  wrd_dim_customer T5,
  wrd_dim_geography T11
   WHERE
  (T5."ID_CUSTOMER" = T1."ID_CUSTOMER") AND
  (T11."ID_GEOGRAPHY" = T5."ID_GEOGRAPHY") AND
  ((T11."REGION", T11."STATECODE") IN (SELECT  MAX(T11.REGION),
  MAX(T11.STATECODE)  FROM wrd_dim_geography T11 WHERE
  (T11.STATECODE IN('AR', 'IA', 'KS', 'KY', 'WY', 'CT', 'MA',
  'NJ', 'NY', 'RI')) AND (T11.REGION IN('Central', 'NorthEast'))
  GROUP BY T11.REGION, T11.STATECODE))
   GROUP BY
  T11."REGION",
  T11."STATECODE  "
   ORDER BY
  T11."REGION",
  T11."STATECODE  "
   FOR FETCH ONLY;
END
```

The output is:

Region	State Code	Revenue
Central	AR	839,075.22
	IA	1,197,171.09
	KS	1,014,388.99
	KY	1,014,825.22
	WY	182,808.08
NorthEast	CT	1,146,626.05
	MA	2,070,919.74
	NJ	2,148,955.56
	NY	6,360,267.52
	RI	342,972.30

Comparing Source and Target Values Using a Sequential File

The empvalues.ftm sequential file contains the last and first names of employees in the MIS department:

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

The empvalues.mas Master File describes the data in the empvalues.ftm file

```
FILENAME=EMPVALUES, SUFFIX=FIX     , IOTYPE=BINARY, $
  SEGMENT=EMPVALUE, SEGTYPE=S0, $
    FIELDNAME=LN, ALIAS=E01, USAGE=A15, ACTUAL=A16, $
    FIELDNAME=FN, ALIAS=E02, USAGE=A10, ACTUAL=A12, $
```

Note: You can create a sequential file, along with a corresponding synonym, using the HOLD FORMAT SQL_SCRIPT command. For more information, see the *ibi™ FOCUS® Creating Reports* manual.

The following request against the FOCUS EMPLOYEE data source uses the DB_INFILE function to compare employee names against the names stored in the empvalues.ftm file:

```
FILEDEF EMPVALUES DISK baseapp/empvalues.ftm
TABLE FILE EMPLOYEE
SUM CURR_SAL
BY LAST_NAME BY FIRST_NAME
WHERE DB_INFILE(EMPVALUES, LAST_NAME, LN, FIRST_NAME, FN)
ON TABLE SET PAGE NOPAGE
END
```

The output is:

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

Control DB_INFILE Optimization

To control whether to prevent optimization of the DB_INFILE expression, issue the following command:

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

In a TABLE request, issue the following command:

```
ON TABLE SET DB_INFILE  {DEFAULT|EXPAND_ALWAYS|EXPAND_NEVER}
```

where:

DEFAULT

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

EXPAND_ALWAYS

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

EXPAND_NEVER

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

DB_LOOKUP: Retrieving Data Source Values

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

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

DB_LOOKUP can be called in a DEFINE command, TABLE COMPUTE command, MODIFY COMPUTE command, or ibi Data Migrator flow.

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

Retrieve a Value From a Lookup Data Source

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

where:

look_mf

Is the lookup Master File.

srcfld1, srcfld2 ...

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

lookfld1, lookfld2 ...

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

returnfld

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

Usage Notes for DB_LOOKUP

- The maximum number of pairs that can be used to match records is 63.
- If the lookup file is a fixed format sequential file, it must be sorted and retrieved in the same order as the source file, unless the ENGINE INT SET CACHE ON command is in effect. Having this setting in effect may also improve performance if the values will be looked up more than once. The key field of the sequential file must be the first lookup field specified in the DB_LOOKUP request. If it is not, no records will match.

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

the lookup file.

- Only real fields in the lookup Master File are valid as lookup and return fields.
- If there are multiple rows in the lookup table where the source field is equal to the lookup field, the first value of the return field is returned.

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

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

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

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

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

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

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

```
DEFINE FILE GGPRODS CLEAR
END
```

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

Product Code	Product	PCAT
-------	-------	----
B141	Hazelnut	
B142	French Roast	
B144	Kona	
F101	Scone	Food
F102	Biscotti	Food
F103	Croissant	Food
G100	Mug	Gifts

G104	Thermos	Gifts
G110	Coffee Grinder	Gifts
G121	Coffee Pot	Gifts

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

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

DECODE: Decoding Values

The DECODE function assigns values based on the coded value of an input field. DECODE is useful for giving a more meaningful value to a coded value in a field. For example, the field GENDER may have the code F for female employees and M for male employees for efficient storage (for example, one character instead of six for *female*). DECODE expands (decodes) these values to ensure correct interpretation on a report.

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

Supply Values in the Function

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

where:

fieldname

Alphanumeric or Numeric

Is the name of the input field.

code

Alphanumeric or Numeric

Is the coded value that DECODE compares with the current value of *fieldname*. If the value has embedded blanks, commas, or other special characters, it must be enclosed in single quotation marks. When DECODE finds the specified value, it returns the corresponding result. When the code is compared to the value of the field name, the code and field name must be in the same format.

result

Alphanumeric or Numeric

Is the returned value that corresponds to the code. If the result has embedded blanks or commas, or contains a negative number, it must be enclosed in single quotation marks. Do not use double quotation marks (").

If the result is presented in alphanumeric format, it must be a non-null, non-blank string. The format of the result must correspond to the data type of the expression.

default

Alphanumeric or Numeric

Is the value returned as a result for non-matching codes. The format must be the same as the format of *result*. If you omit a default value, DECODE assigns a blank or zero to non-matching codes.

filename

Alphanumeric

Is the ddname that points to the file in which code/result pairs are stored. Every record in the file must contain a pair.

You can use up to 40 lines to define the code and result pairs for any given DECODE function, or 39 lines if you also use an ELSE phrase. Use either a comma or blank to separate the code from the result, or one pair from another.

Note: DECODE has no *output* argument.

Supplying Values Using the DECODE Function

EDIT extracts the first character of the CURR_JOBCODE field, then DECODE returns either ADMINISTRATIVE or DATA PROCESSING depending on the value extracted.

```
TABLE FILE EMPLOYEE
PRINT CURR_JOBCODE AND COMPUTE
DEPX_CODE/A1 = EDIT(CURR_JOBCODE, '9$$'); NOPRINT AND COMPUTE
JOB_CATEGORY/A15 = DECODE DEPX_CODE(A 'ADMINISTRATIVE'
              B 'DATA PROCESSING');
BY LAST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME    CURR_JOBCODE    JOB_CATEGORY
---------    ------------    ------------
BLACKWOOD    B04             DATA PROCESSING
CROSS        A17             ADMINISTRATIVE
GREENSPAN    A07             ADMINISTRATIVE
JONES        B03             DATA PROCESSING
MCCOY        B02             DATA PROCESSING
SMITH        B14             DATA PROCESSING
```

Guidelines for Reading Values From a File

- Each record in the file is expected to contain pairs of elements separated by a comma or blank.
- If each record in the file consists of only one element, this element is interpreted as

the code, and the result becomes either a blank or zero, as needed.

This makes it possible to use the file to hold screening literals referenced in the screening condition:

```
IF field IS (filename)
```

and as a file of literals for an IF criteria specified in a computational expression. For example:

```
TAKE = DECODE SELECT (filename ELSE 1);
VALUE = IF TAKE IS 0 THEN... ELSE...;
```

TAKE is 0 for SELECT values found in the literal file and 1 in all other cases. The VALUE computation is carried out as if the expression had been:

```
IF SELECT (filename) THEN... ELSE...;
```

- The file can contain up to 32,767 characters in the file.
- All data is interpreted in ASCII format on UNIX, or in EBCDIC format on z/OS, and converted to the USAGE format of the DECODE pairs.
- Leading and trailing blanks are ignored.
- The remainder of each record is ignored and can be used for comments or other data. This convention applies in all cases, except when the file name is HOLD. In that case, the file is presumed to have been created by the HOLD command, which writes fields in the internal format, and the DECODE pairs are interpreted accordingly. In this case, extraneous data in the record is ignored.

Reading DECODE Values From a File

The following example has two parts. The first part creates a file with a list of IDs and reads the EDUCFILE data source. The second part reads the EMPLOYEE data source and assigns 0 to those employees who have taken classes and 1 to those employees who have not. The HOLD file contains only one column of values. Therefore, DECODE assigns the value 0 to an employee whose EMP_ID appears in the file and 1 when EMP_ID does not appear in the file.

```
TABLE FILE EDUCFILE
PRINT EMP_ID
ON TABLE HOLD
END
```

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND LAST_NAME AND FIRST_NAME AND COMPUTE
NOT_IN_LIST/I1 = DECODE EMP_ID(HOLD ELSE 1);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
EMP_ID     LAST_NAME        FIRST_NAME  NOT_IN_LIST
------     ---------        ----------  -----------
112847612  SMITH            MARY                  0
117593129  JONES            DIANE                 0
219984371  MCCOY            JOHN                  1
326179357  BLACKWOOD        ROSEMARIE             0
543729165  GREENSPAN        MARY                  1
818692173  CROSS            BARBARA               0
```

FIND: Verifying the Existence of a Value in a Data Source

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

Note: For MODIFY only, the FIND function verifies the existence of an incoming data value in an indexed FOCUS data source field.

You can also use FIND in a VALIDATE command to determine if a transaction field value exists in another FOCUS data source. If the field value is not in that data source, the function returns a value of 0, causing the validation test to fail and the request to reject the transaction.

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

Limit: FIND does not work on files with different DBA passwords.

The opposite of FIND is NOT FIND. The NOT FIND function sets a temporary field to **1** if the incoming value is not in the data source and to **0** if the incoming value is in the data source.

Verify the Existence of a Value in a Data Source

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

where:

fieldname

Is the name of the field that contains the incoming data value.

AS dbfield

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

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

file

Is the name of the FOCUS data source.

For MODIFY - the IN field must be indexed.

Note:

- FIND does not use an *output* argument.
- Do not include a space between FIND and the left parenthesis.

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

FIND determines if a supplied value in the EMP_ID field is in the EDUCFILE data source. The procedure then displays a message indicating the result of the search.

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID
COMPUTE
   EDTEST = FIND(EMP_ID IN EDUCFILE);
   MSG/A40 = IF EDTEST NE 0 THEN
      'STUDENT LISTED IN EDUCATION FILE' ELSE
      'STUDENT NOT LISTED IN EDUCATION FILE';
MATCH EMP_ID
      ON NOMATCH TYPE "<MSG"
      ON MATCH TYPE "<MSG"
DATA
```

A sample execution is:

```
>
 EMPLOYEE ON 12/04/2001 AT 12.09.03
 DATA FOR TRANSACTION                  1

 EMP_ID              =
112847612
 STUDENT LISTED IN EDUCATION FILE
 DATA FOR TRANSACTION                  2

 EMP_ID              =
219984371
 STUDENT NOT LISTED IN EDUCATION FILE
 DATA FOR TRANSACTION                  3
```

The procedure processes as follows:

1. The procedure prompts you for an employee ID. You enter 112847612.
2. The procedure searches the EDUCFILE data source for the employee ID 112847612. It finds the ID so it prints STUDENT LISTED IN EDUCATION FILE.
3. The procedure prompts you for an employee ID. You enter 219984371.
4. The procedure searches the EDUCFILE data source for the employee ID 219984371. It does not find the ID so it prints STUDENT NOT LISTED IN EDUCATION FILE.

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

The following updates the number of hours an employee spent in class. The VALIDATE command rejects a transaction for an employee whose ID is not found in the EDUCFILE data source, which records class attendance.

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID ED_HRS
VALIDATE
   EDTEST = FIND(EMP_ID IN EDUCFILE);
MATCH EMP_ID
   ON NOMATCH REJECT
   ON MATCH UPDATE ED_HRS
DATA
```

A sample execution is:

```
>
 EMPLOYEE ON 12/04/2001 AT 12/26/08
 DATA FOR TRANSACTION        1

 EMP_ID       =
112847612
 ED_HRS     =
7
 DATA FOR TRANSACTION        2

 EMP_ID      =
219984371
 ED_HRS     =
0
 (FOC421) TRANS 2 REJECTED INVALID EDTEST
 219984371, 0, $
 DATA FOR TRANSACTION        3
```

The procedure processes as follows:

1. The procedure prompts you for an employee ID and the number of hours the employee spent in class. You enter the following data:

   ```
   EMP_ID:  112847612
   ED_HRS:  7
   ```

2. The procedure updates the number of hours for the ID 112847612.

3. The procedure prompts you for an employee ID and the number of hours the employee spent in class. You enter the following data:

   ```
   EMP_ID:  219984371
   ```

```
ED_HRS:  0
```

4. The procedure rejects the record for the ID 219984371 because it does not exist in the EDUCFILE data source, and an error message is returned.

IMPUTE: Replacing Missing Values With Aggregated Values

IMPUTE calculates a value to replace missing numeric data on report output, within a partition.

In place of eliminating data records with missing values from analysis, IMPUTE enables you to substitute a variety of estimates for the missing values, including the mean, the median, the mode, or a numeric constant, all calculated within the data partition specified by the reset key. This function is designed to be used with detail level reports (PRINT or LIST commands), and with calculated values (fields created with the COMPUTE command).

Replace Missing Values With Aggregated Values

```
IMPUTE(field, reset_key, replacement)
```

where:

field

Is the name of the numeric input field that is defined with MISSING ON.

reset_key

Defines the partition for the calculation. Valid values are:

- A sort field name.
- PRESET, which uses the break defined by the SET PARTITION_ON command.
- TABLE, which performs the calculation on the entire table.

replacement

Is a numeric constant or one of the following:

- MEAN
- MEDIAN
- MODE

Replacing Missing Values With Aggregated Values

To run this example, the FOCUS data source SALEMISS must be created. SALEMISS is the SALES data source with some missing values added in the RETURNS and DAMAGED fields. The following is the SALEMISS Master File, which should be added to the IBISAMP application.

```
FILENAME=KSALES, SUFFIX=FOC, REMARKS='Legacy Metadata Sample: sales',$

SEGNAME=STOR_SEG, SEGTYPE=S1,
   FIELDNAME=STORE_CODE,  ALIAS=SNO,  FORMAT=A3,   $
   FIELDNAME=CITY,        ALIAS=CTY,  FORMAT=A15,  $
   FIELDNAME=AREA,        ALIAS=LOC,  FORMAT=A1,   $

SEGNAME=DATE_SEG, PARENT=STOR_SEG, SEGTYPE=SH1,
   FIELDNAME=DATE,        ALIAS=DTE,  FORMAT=A4MD, $

SEGNAME=PRODUCT, PARENT=DATE_SEG, SEGTYPE=S1,
   FIELDNAME=PROD_CODE,     ALIAS=PCODE,   FORMAT=A3,    FIELDTYPE=I, $
   FIELDNAME=UNIT_SOLD,     ALIAS=SOLD,    FORMAT=I5,    $
   FIELDNAME=RETAIL_PRICE,  ALIAS=RP,      FORMAT=D5.2M, $
   FIELDNAME=DELIVER_AMT,   ALIAS=SHIP,    FORMAT=I5,    $
   FIELDNAME=OPENING_AMT,   ALIAS=INV,     FORMAT=I5,    $
   FIELDNAME=RETURNS,       ALIAS=RTN,     FORMAT=I3,    MISSING=ON, $
   FIELDNAME=DAMAGED,       ALIAS=BAD,     FORMAT=I3,    MISSING=ON, $
```

The following procedure creates the SALEMISS data source and then adds the missing values to the RETURNS and DAMAGED fields:

```
CREATE FILE ibisamp/SALEMISS
MODIFY FILE ibisamp/SALEMISS
FIXFORM STORE_CODE/3 CITY/15 AREA/1 DATE/4 PROD_CODE/3
FIXFORM UNIT_SOLD/5 RETAIL_PRICE/5 DELIVER_AMT/5
FIXFORM OPENING_AMT/5 RETURNS/3 DAMAGED/3
MATCH STORE_CODE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH DATE
```

```
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH PROD_CODE
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
14BSTAMFORD       S1212B10   60  .95   80   65 10  6
14BSTAMFORD       S1212B12   40 1.29   20   50  3  3
14BSTAMFORD       S1212B17   29 1.89   30   30  2  1
14BSTAMFORD       S1212C13   25 1.99   30   40  3  0
14BSTAMFORD       S1212C7    45 2.39   50   49  5  4
14BSTAMFORD       S1212D12   27 2.19   40   35  0  0
14BSTAMFORD       S1212E2    80  .99  100  100  9  4
14BSTAMFORD       S1212E3    70 1.09   80   90  8  9
14ZNEW YORK       U1017B10   30  .85   30   10  2  3
14ZNEW YORK       U1017B17   20 1.89   40   25  2  1
14ZNEW YORK       U1017B20   15 1.99   30    5  0  1
14ZNEW YORK       U1017C17   12 2.09   10   15  0  0
14ZNEW YORK       U1017D12   20 2.09   30   10  3  2
14ZNEW YORK       U1017E1    30  .89   25   45  4  7
14ZNEW YORK       U1017E3    35 1.09   25   45  4  2
77FUNIONDALE      R1018B20   25 2.09   40   25  1  1
77FUNIONDALE      R1018C7    40 2.49   40   40  0  0
K1 NEWARK         U1019B12   29 1.49   30   30  1  0
K1 NEWARK         U1018B10   13  .99   30   15  1  1
END
-RUN
```

```
MODIFY FILE ibisamp/SALEMISS
FIXFORM STORE_CODE/3 DATE/5 PROD_CODE/4
FIXFORM UNIT/3 RETAIL/5 DELIVER/3
FIXFORM OPEN/3 RETURNS/C3 DAMAGED/C3
MATCH STORE_CODE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH DATE
ON NOMATCH INCLUDE
ON MATCH CONTINUE
MATCH PROD_CODE
ON NOMATCH INCLUDE
ON MATCH REJECT
DATA
14Z1017 C13 15 1.99 35 30    6
14Z1017 C14 18 2.05 30 25 4
14Z1017 E2  33 0.99 45 40
```

```
END
-RUN
```

The following request against the SALEMISS data source generates replacement values for the missing values in the RETURNS field, using only the values within the same store.

```
SET PARTITION_ON=FIRST
TABLE FILE SALEMISS
PRINT RETURNS
COMPUTE MEDIAN1 = IMPUTE(RETURNS, PRESET, MEDIAN);
COMPUTE MEAN1 = IMPUTE(RETURNS, PRESET, MEAN);
COMPUTE MODE1 = IMPUTE(RETURNS, PRESET, MODE);
BY STORE_CODE
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
TYPE=REPORT, GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. The missing values occur in store 14Z, and the replacement values are calculated using only the RETURNS values from that store because PARTITION_ON is set to FIRST.

STORE_CODE	RETURNS	MEDIAN1	MEAN1	MODE1
14B	10	10.00	10.00	10.00
	3	3.00	3.00	3.00
	2	2.00	2.00	2.00
	3	3.00	3.00	3.00
	5	5.00	5.00	5.00
	0	.00	.00	.00
	9	9.00	9.00	9.00
	8	8.00	8.00	8.00
14Z	2	2.00	2.00	2.00
	2	2.00	2.00	2.00
	0	.00	.00	.00
	.	2.00	2.00	4.00
	4	4.00	4.00	4.00
	0	.00	.00	.00
	3	3.00	3.00	3.00
	4	4.00	4.00	4.00
	.	2.00	2.00	4.00
	4	4.00	4.00	4.00
77F	1	1.00	1.00	1.00
	0	.00	.00	.00
K1	1	1.00	1.00	1.00
	1	1.00	1.00	1.00

Changing the PARTITION_ON setting to TABLE produces the following output, in which the replacement values are calculated using all of the rows in the table.

STORE_CODE	RETURNS	MEDIAN1	MEAN1	MODE1
14B	10	10.00	10.00	10.00
	3	3.00	3.00	3.00
	2	2.00	2.00	2.00
	3	3.00	3.00	3.00
	5	5.00	5.00	5.00
	0	.00	.00	.00
	9	9.00	9.00	9.00
	8	8.00	8.00	8.00
14Z	2	2.00	2.00	2.00
	2	2.00	2.00	2.00
	0	.00	.00	.00
	.	2.00	3.00	.00
	4	4.00	4.00	4.00
	0	.00	.00	.00
	3	3.00	3.00	3.00
	4	4.00	4.00	4.00
	.	2.00	3.00	.00
	4	4.00	4.00	4.00
77F	1	1.00	1.00	1.00
	0	.00	.00	.00
K1	1	1.00	1.00	1.00
	1	1.00	1.00	1.00

LAST: Retrieving the Preceding Value

The LAST function retrieves the preceding value for a field.

The effect of LAST depends on whether it appears in a DEFINE or COMPUTE command:

- In a DEFINE command, the LAST value applies to the previous record retrieved from the data source before sorting takes place.
- In a COMPUTE command, the LAST value applies to the record in the previous line of the internal matrix.

Do not use LAST with the -SET command in Dialogue Manager.

Retrieve the Preceding Value

```
LAST fieldname
```

where:

fieldname

Alphanumeric or Numeric

Is the field name.

Note: LAST does not use an *output* argument.

Retrieving the Preceding Value

LAST retrieves the previous value of the DEPARTMENT field to determine whether to restart the running total of salaries by department. If the previous value equals the current value, CURR_SAL is added to RUN_TOT to generate a running total of salaries within each department.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME CURR_SAL AND COMPUTE
RUN_TOT/D12.2M = IF DEPARTMENT EQ LAST DEPARTMENT THEN
                (RUN_TOT + CURR_SAL) ELSE CURR_SAL ;
AS 'RUNNING,TOTAL,SALARY'
BY DEPARTMENT SKIP-LINE
END
```

The output is:

```
                                                 RUNNING
                                                   TOTAL
DEPARTMENT  LAST_NAME             CURR_SAL        SALARY
----------  ---------             --------       -------
MIS         SMITH               $13,200.00    $13,200.00
            JONES               $18,480.00    $31,680.00
            MCCOY               $18,480.00    $50,160.00
            BLACKWOOD           $21,780.00    $71,940.00
            GREENSPAN            $9,000.00    $80,940.00
            CROSS               $27,062.00   $108,002.00
PRODUCTION  STEVENS             $11,000.00    $11,000.00
```

```
SMITH              $9,500.00      $20,500.00
BANNING           $29,700.00      $50,200.00
IRVING            $26,862.00      $77,062.00
ROMANS            $21,120.00      $98,182.00
MCKNIGHT          $16,100.00     $114,282.00
```

LOOKUP: Retrieving a Value From a Cross-referenced Data Source

The LOOKUP function retrieves a data value from a cross-referenced FOCUS data source in a MODIFY request. You can retrieve data from a data source cross-referenced statically in a Master File or a data source joined dynamically to another by the JOIN command. LOOKUP retrieves a value, but does not activate the field. LOOKUP is required because a MODIFY request, unlike a TABLE request, cannot read cross-referenced data sources freely.

LOOKUP allows a request to use the retrieved data in a computation or message, but it does not allow you to modify a cross-referenced data source.

To modify more than one data source in one request, use the COMBINE command.

LOOKUP can read a cross-referenced segment that is linked directly to a segment in the host data source (the host segment). This means that the cross-referenced segment must have a segment type of KU, KM, DKU, or DKM (but not KL or KLU) or must contain the cross-referenced field specified by the JOIN command. Because LOOKUP retrieves a single cross-referenced value, it is best used with unique cross-referenced segments.

The cross-referenced segment contains two fields used by LOOKUP:

- The field containing the retrieved value. Alternatively, you can retrieve all the fields in a segment at one time. The field, or your decision to retrieve all the fields, is specified in LOOKUP.

  For example, LOOKUP retrieves all the fields from the segment

  ```
  RTN = LOOKUP(SEG.DATE_ATTEND);
  ```

- The cross-referenced field. This field shares values with a field in the host segment called the host field. These two fields link the host segment to the cross-referenced segment. LOOKUP uses the cross-referenced field, which is indexed, to locate a specific segment instance.

When using LOOKUP, the MODIFY request reads a transaction value for the host field. It then searches the cross-referenced segment for an instance containing this value in the cross-referenced field:

- If there are no instances of the value, the function sets a return variable to 0. If you use the field specified by LOOKUP in the request, the field assumes a value of blank if alphanumeric and 0 if numeric.
- If there are instances of the value, the function sets the return variable to 1 and retrieves the value of the specified field from the first instance it finds. There can be more than one if the cross-referenced segment type is KM or DKM, or if you specified the ALL keyword in the JOIN command.

Retrieve a Value From a Cross-referenced Data Source

```
LOOKUP(field);
```

where:

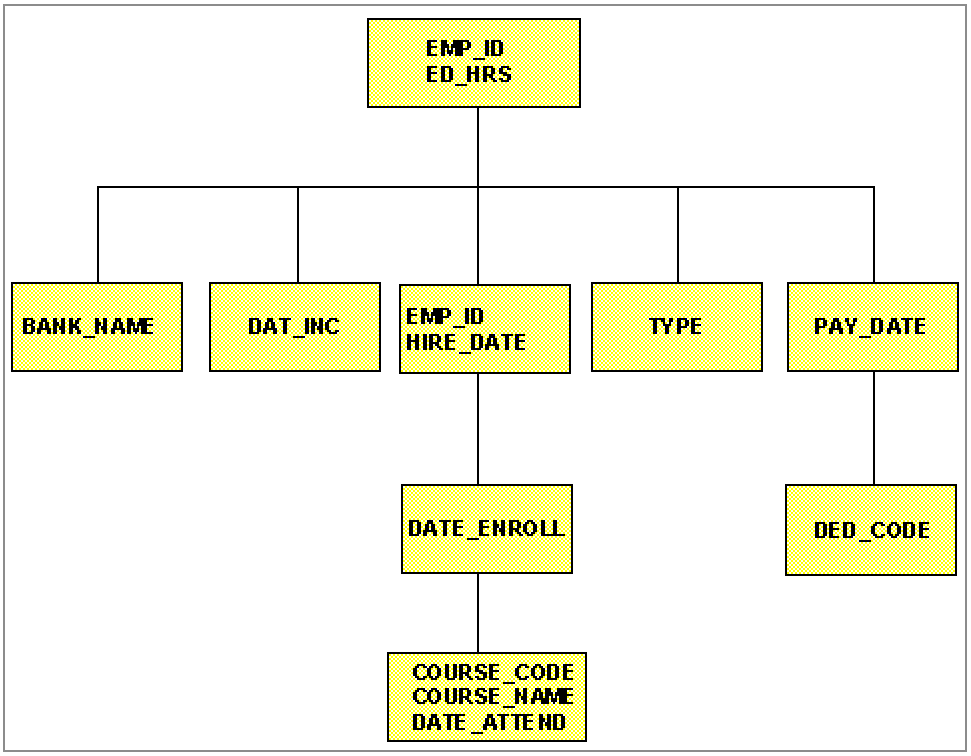
field

Is the name of the field to retrieve in the cross-referenced file. If the field name also exists in the host data source, you must qualify it here. Do not include a space between LOOKUP and the left parenthesis.

Note: LOOKUP does not use an *output* argument.

Reading a Value From a Cross-referenced Data Source

You may need to determine if employees were hired before or after a specific date, for example, January 1, 1982. The employee IDs (EMP_ID) and hire date (HIRE_DATE) are located in the host segment. The following diagram shows the file structure:



The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID ED_HRS
COMPUTE  EDTEST = LOOKUP(HIRE_DATE);
   COMPUTE ED_HRS = IF DATE_ENROLL GE 820101 THEN ED_HRS * 1.1
         ELSE ED_HRS;
MATCH EMP_ID
   ON MATCH UPDATE ED_HRS
   ON NOMATCH REJECT
DATA
```

A sample execution is:

1. The request prompts you for the employee ID and number of class hours. Enter the ID 117593129 and 10 class hours.
2. LOOKUP locates the first instance in the cross-referenced segment containing the employee ID 117593129. Since the instance exists, the function returns a 1 to the EDTEST variable. This instance lists the enroll date as 821028 (October 28, 1982).
3. LOOKUP retrieves the value 821028 for the DATE_ENROLL field.

4. The COMPUTE command tests the value of DATE_ENROLL. Since October 28, 1982 is after January 1, 1982, the ED_HRS are increased from 10 to 11.
5. The request updates the classroom hours for employee 117593129 with the new value.

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

You can use a field value in a host segment instance to search a cross-referenced segment. Do the following:

- In the MATCH command that selects the host segment instance, activate the host field with the ACTIVATE command.
- In the same MATCH command, code LOOKUP after the ACTIVATE command.

This request displays the employee ID, date of salary increase, employee name, and the employee position after the raise was granted:

- The employee ID and name (EMP_ID) are in the root segment.
- The date of increase (DAT_INC) is in the descendant host segment.
- The job position is in the cross-referenced segment.
- The shared field is JOBCODE. You never enter a job code; the values are stored in the data source.

The request is:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID DAT_INC
MATCH EMP_ID
   ON NOMATCH REJECT
   ON MATCH CONTINUE
MATCH DAT_INC
   ON NOMATCH REJECT
   ON MATCH ACTIVATE JOBCODE
   ON MATCH COMPUTE
      RTN = LOOKUP(JOB_DESC);
   ON MATCH TYPE
      "EMPLOYEE ID:         <EMP_ID"
      "DATE INCREASE:       <DAT_INC"
      "NAME:                <D.FIRST_NAME <D.LAST_NAME"
      "POSITION:            <JOB_DESC"
DATA
```

A sample execution is:

1. The request prompts you for the employee ID and date of pay increase. Enter the employee ID 071382660 and the date 820101 (January 1, 1982).
2. The request locates the instance containing the ID 071382660, then locates the child instance containing the date of increase 820101.
3. This child instance contains the job code A07. The ACTIVATE command makes this value available to LOOKUP.
4. LOOKUP locates the job code A07 in the cross-referenced segment. It returns a 1 the RTN variable and retrieves the corresponding job description SECRETARY.
5. The TYPE command displays the values:

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

Fields retrieved by LOOKUP do not require the D. prefix. FOCUS treats the field values as transaction values.

You may also need to activate the host field if you are using LOOKUP within a NEXT command. This request displays the latest position held by an employee:

```
MODIFY FILE EMPLOYEE
PROMPT EMP_ID
MATCH EMP_ID
   ON NOMATCH REJECT
   ON MATCH CONTINUE
NEXT DAT_INC
   ON NONEXT REJECT
   ON NEXT ACTIVATE JOBCODE
   ON NEXT COMPUTE
      RTN = LOOKUP(JOB_DESC);
   ON MATCH TYPE
      "EMPLOYEE ID:      <EMP_ID"
      "DATE OF POSITION: <DAT_INC"
      "NAME:             <D.FIRST_NAME <D.LAST_NAME"
      "POSITION:         <JOB_DESC"
DATA
```

Using the LOOKUP Function With a VALIDATE Command

When you use LOOKUP, reject transactions containing values for which there is no corresponding instance in the cross-reference segment. To do this, place the function in a VALIDATE command. If the function cannot locate the instance in the cross-referenced segment, it sets the value of the return variable to 0, causing the request to reject the transaction.

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

The request is as follows:

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

If an employee record is not found in the cross-referenced segment, that employee never enrolled in a class. The transaction is rejected as an error.

Using the Extended LOOKUP Function

If the LOOKUP function cannot locate a value of the host field in the cross-referenced segment, use extended syntax to locate the next highest or lowest cross-referenced field value in the cross-referenced segment.

To use this feature, create the index with the INDEX parameter set to NEW (the binary tree scheme). To determine the type of index used by a data source, enter the FDT command.

Use the Extended LOOKUP Function

```
COMPUTE
LOOKUP(field action);
```

where:

field

Is the name of the field in the cross-referenced data source, used in a MODIFY computation. If the field name also exists in the host data source, you must qualify it here.

action

Specifies the action the request takes. Valid values are:

`EQ` causes LOOKUP to take no further action if an exact match is not found. If a match is found, the value of rcode is set to 1; otherwise, it is set to 0. This is the default.

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

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

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

The following table shows the value of *rcode,* depending on which instance LOOKUP locates:

Value	Action
1	Exact cross-referenced value located.
2	Next highest cross-referenced value located.
-2	Next lowest cross-referenced value located.
0	Cross-referenced value not located.

NULLIF: Returning a Null Value When Parameters Are Equal

NULLIF returns a null (missing) value when its parameters are equal. If they are not equal, it returns the first value. The field to which the value is returned should have MISSING ON.

Return a Null Value for Equal Parameters

```
NULLIF(arg1,arg2)
```

where:

arg1,arg2

Any type of field, constant, or expression.

Are the input parameters that are tested for equality. They must either both be numeric or both be alphanumeric.

The output data type is the same as the input data types.

Testing for Equal Parameters

The following request uses NULLIF to test the DAMAGED and RETURNS field values for equality.

```
DEFINE FILE SALES
NULL1/I4 MISSING ON = NULLIF(DAMAGED, RETURNS);
END
TABLE FILE SALES
PRINT DAMAGED RETURNS NULL1
BY STORE_CODE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
END
```

The output is shown in the following image.

STORE_CODE	DAMAGED	RETURNS	NULL1
14B	6	10	6
	3	3	.
	1	2	1
	0	3	0
	4	5	4
	0	0	.
	4	9	4
	9	8	9
14Z	3	2	3
	1	2	1
	1	0	1
	0	0	.
	2	3	2
	7	4	7
	2	4	2
77F	1	1	.
	0	0	.
K1	0	1	0
	1	1	.

FOCUSPAGES: Finding the size of a FOCUS or XFOCUS file

`FOCUSPAGES` is a function that you can use to find the size of a `FOCUS` or `XFOCUS` file in pages. It returns the same value that is returned by `? FILE (filename)`. You can store the value returned by this function in an amper variable and use it to determine whether a `FOCUS` or `XFOCUS` file has data.

Compute the Number of Pages in a FOCUS or XFOCUS File

```
FOCUSPAGES(filename)
```

where:

filename

Is the name of the FOCUS or XFOCUS file whose size you want to find.

Get the size of a FOCUS or XFOCUS file

The following request gets the size of a FOCUS or XFOCUS file in pages.

```
-SET &S=FOCUSPAGES(GGSALES);
-TYPE GGSALES file has TOTAL &S PAGES
-RUN
? FDT GGSALES
? FILE GGSALES
```

The output is as follows.

```
 GGSALES file has TOTAL 205 PAGES
0    DIRECTORY:/port/edaport/R729999D/C/conf/apps/ibisamp/ggsales.foc ON
02/21/2024 AT 13.46.20
     DATE/TIME OF LAST CHANGE:   02/20/2024   21.53.12
0    SEGNAME   LENGTH   PARENT   START     END    PAGES  LINKS  TYPE
1    SALES01       25                1     204     111       1
     CATEGORY                        2     205      21            NEW
     PCD                             3     202      16            NEW
     REGION                          4     200      30            NEW
     ST                              5     203      12            NEW
     STCD                            6     191      15            NEW
0 STATUS OF FOCUS FILE:
/port/edaport/R729999D/C/conf/apps/ibisamp/ggsales.foc ON 02/21/2024 AT
13.46.20
0                  ACTIVE  DELETED    DATE OF    TIME OF    LAST TRANS
  SEGNAME          COUNT   COUNT      LAST CHG   LAST CHG     NUMBER
  SALES01          4317              02/20/2024   21.53.12     4317
  *INDEXES*
  CATEGORY                           02/20/2024   21.53.12     4317
  PCD                                02/20/2024   21.53.12     4317
  REGION                             02/20/2024   21.53.12     4317
  ST                                 02/20/2024   21.53.12     4317
  STCD                               02/20/2024   21.53.12     4317
0 TOTAL SEGS      4317
  TOTAL CHAR    414432
  TOTAL PAGES      205
  LAST CHANGE                        02/20/2024   21.53.12     4317
```

The value returned by the FOCUSPAGES function is stored in the amper variable &S. The output also shows the total size of the file GGSALES as 205 pages.

Simplified Date and Date-Time Functions

Simplified date and date-time functions have streamlined parameter lists, similar to those used by SQL functions. In some cases, these simplified functions provide slightly different functionality than previous versions of similar functions.

The simplified functions do not have an output argument. Each function returns a value that has a specific data type.

When used in a request against a relational data source, these functions are optimized (passed to the RDBMS for processing).

Standard date and date-time formats refer to YYMD and HYYMD syntax (dates that are not stored in alphanumeric or numeric fields). Dates not in these formats must be converted before they can be used in the simplified functions. Input date and date-time parameters must provide full component dates. Literal date-time values can be used with the DT function.

All arguments can be either literals, field names, or amper variables.

DAYNAME: Returning the Name of the Day From a Date Expression

DAYNAME returns a character string that contains the data-source-specific name of the day for the day part of a date expression.

Return the Name of the Day From a Date Expression

```
DAYNAME(date_exp)
```

where:

date_exp

Is a date or date-time expression.

Returning the Name of the Day From a Date Expression

The following request returns the name of the day from the TIME_DATE field.

```
TABLE FILE WF_RETAIL_TIME
PRINT TIME_DATE
COMPUTE DAYNAME1/A12 = DAYNAME(TIME_DATE);
WHERE RECORDLIMIT EQ 5
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	DAYNAME1
2009/01/01	Thursday
2009/01/02	Friday
2009/01/03	Saturday
2009/01/04	Sunday
2009/01/05	Monday

DT_CURRENT_DATE: Returning the Current Date

The DT_CURRENT_DATE function returns the current date-time provided by the running operating environment in date-time format. The time portion of the date-time is set to zero.

Return the Current Date

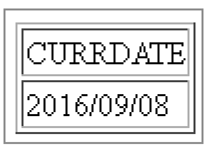
```
DT_CURRENT_DATE()
```

Returning the Current Date

The following request returns the current date.

```
DEFINE FILE WFLITE
CURRDATE/YYMD WITH COUNTRY_NAME = DT_CURRENT_DATE();
END
TABLE FILE WFLITE
SUM CURRDATE
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.



CURRDATE
2016/09/08

DT_CURRENT_DATETIME: Returning the Current Date and Time

DT_CURRENT_DATETIME returns the current date and time provided by the running operating environment in date-time format, with a specified time precision.

Return the Current Date and Time

```
DT_CURRENT_DATETIME(component)
```

where:

component

Is one of the following time precisions.

- SECOND.
- MILLISECOND.
- MICROSECOND.

> **Note:** The field to which the value is returned must have a format that supports the time precision requested.

Returning the Current Date and Time

The following request returns the current date and time, with the time specified in microseconds.

```
DEFINE FILE WFLITE
CURRDATE/HYYMDm WITH COUNTRY_NAME = DT_CURRENT_DATETIME(MICROSECOND);
END
TABLE FILE WFLITE
SUM CURRDATE
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.

CURRDATE
2016/09/08 17:10:31.605718

DT_CURRENT_TIME: Returning the Current Time

The DT_CURRENT_TIME function returns the current time provided by the running operating environment in date-time format, with a specified time precision. The date portion of the returned date-time value is set to zero.

Return the Current Time

```
DT_CURRENT_TIME(component)
```

where:

component

Is one of the following time precisions.

- SECOND.
- MILLISECOND.
- MICROSECOND.

> **Note:** The field to which the value is returned must have a format that supports the time precision requested.

Returning the Current Time

The following request returns the current time, with the time precision set to milliseconds.

```
DEFINE FILE WFLITE
CURRTIME/HHISs WITH COUNTRY_NAME = DT_CURRENT_TIME(MILLISECOND);
END
TABLE FILE WFLITE
SUM CURRTIME
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.

CURRTIME
17:23:13.098

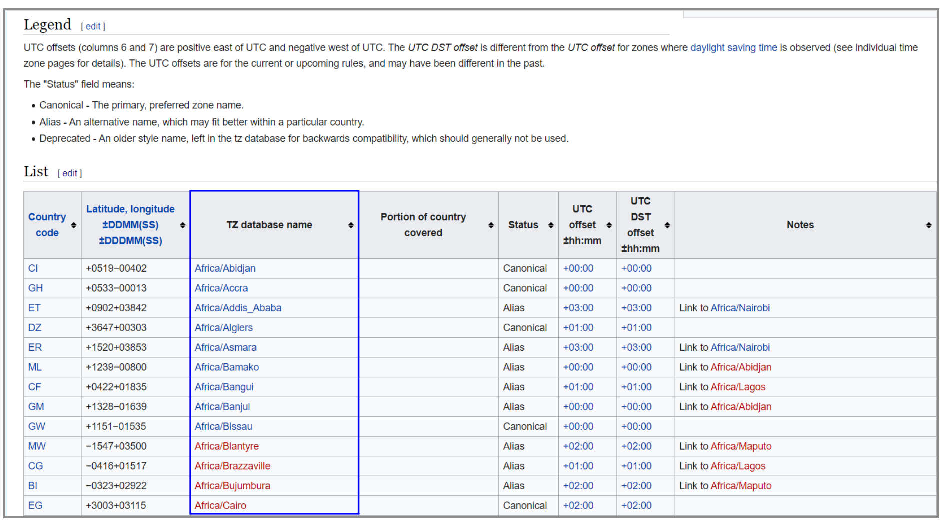
DT_TOLOCAL: Converting Universal Coordinated Time to Local Time

Coordinated Universal Time (UTC) is the time standard commonly used around the world. To convert UTC time to a local time, a certain number of hours must be added to or subtracted from the UTC time, depending on the number of time zones between the locality and Greenwich, England (GMT).

DT_TOLOCAL converts UTC time to local time.

Converting timestamp values from different localities to a common standard time enables you to sort events into the actual event sequence.

This function requires an IANA (Internet Assigned Numbers Authority) time zone database names (expressed as 'Area/Location') as a parameter. You can find a table of IANA TZ database names on Wikipedia at https://en.wikipedia.org/wiki/List_of_tz_database_time_zones, as shown in the following image.



Legend [edit]

UTC offsets (columns 6 and 7) are positive east of UTC and negative west of UTC. The *UTC DST offset* is different from the *UTC offset* for zones where daylight saving time is observed (see individual time zone pages for details). The UTC offsets are for the current or upcoming rules, and may have been different in the past.

The "Status" field means:

- Canonical - The primary, preferred zone name.
- Alias - An alternative name, which may fit better within a particular country.
- Deprecated - An older style name, left in the tz database for backwards compatibility, which should generally not be used.

List [edit]

Country code	Latitude, longitude ±DDMM(SS) ±DDDMM(SS)	TZ database name	Portion of country covered	Status	UTC offset ±hh:mm	UTC DST offset ±hh:mm	Notes
CI	+0519−00402	Africa/Abidjan		Canonical	+00:00	+00:00	
GH	+0533−00013	Africa/Accra		Canonical	+00:00	+00:00	
ET	+0902+03842	Africa/Addis_Ababa		Alias	+03:00	+03:00	Link to Africa/Nairobi
DZ	+3647+00303	Africa/Algiers		Canonical	+01:00	+01:00	
ER	+1520+03853	Africa/Asmara		Alias	+03:00	+03:00	Link to Africa/Nairobi
ML	+1239−00800	Africa/Bamako		Alias	+00:00	+00:00	Link to Africa/Abidjan
CF	+0422+01835	Africa/Bangui		Alias	+01:00	+01:00	Link to Africa/Lagos
GM	+1328−01639	Africa/Banjul		Alias	+00:00	+00:00	Link to Africa/Abidjan
GW	+1151−01535	Africa/Bissau		Canonical	+00:00	+00:00	
MW	−1547+03500	Africa/Blantyre		Alias	+02:00	+02:00	Link to Africa/Maputo
CG	−0416+01517	Africa/Brazzaville		Alias	+01:00	+01:00	Link to Africa/Lagos
BI	−0323+02922	Africa/Bujumbura		Alias	+02:00	+02:00	Link to Africa/Maputo
EG	+3003+03115	Africa/Cairo		Canonical	+02:00	+02:00	

If you do not know what Area and Location corresponds to your time zone, but you do know your offset from GMT, or your legacy time zone name (such as EST), scroll down in the table. There are TZ database names that correspond to these time zone identifiers, as shown in the following image.

		EST		Deprecated	−05:00	−05:00	Choose a zone that currently *observes* EST without daylight saving time, such as America/Cancun.
		EST5EDT		Deprecated	−05:00	−04:00	Choose a zone that *observes* EST with United States daylight saving time rules, such as America/New_York.
		Etc/GMT		Canonical	+00:00	+00:00	
		Etc/GMT+0		Alias	+00:00	+00:00	Link to Etc/GMT
		Etc/GMT+1		Canonical	−01:00	−01:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+10		Canonical	−10:00	−10:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+11		Canonical	−11:00	−11:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+12		Canonical	−12:00	−12:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+2		Canonical	−02:00	−02:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+3		Canonical	−03:00	−03:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+4		Canonical	−04:00	−04:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+5		Canonical	−05:00	−05:00	Sign is intentionally inverted. See the Etc area description.

Note: If you use a standard IANA time zone database name in the form 'Area/Location' (for example, 'America/New_York'), automatic adjustments are made for Daylight Savings Time. If you use a name that corresponds to an offset from GMT or to a legacy time zone name, it is your responsibility to account for Daylight Savings Time.

Convert UTC Time to Local Time

```
DT_TOLOCAL(datetime, timezone)
```

where:

datetime

Date-time

Is a date-time expression representing UTC time, containing date and time components.

timezone

Alphanumeric

Is a character expression containing the IANA time zone name of the local time, in the form 'Area/Location' (for example, 'America/New_York').

Converting UTC Time to Local Time

The following request converts the current date-time value from UTC time to local time for the time zone 'America/New_York'.

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT
COMPUTE UTC1/HYYMDS  = DT_CURRENT_DATETIME(SECOND);
COMPUTE LOCAL1/HYYMDS = DT_TOLOCAL(UTC1, 'America/New_York');
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

UTC1	LOCAL1
2020/09/04 15:00:26	2020/09/04 11:00:26

DT_TOUTC: Converting Local Time to Universal Coordinated Time

Coordinated Universal Time (UTC) is the time standard commonly used around the world. To convert UTC time to a local time, a certain number of hours must be added to or subtracted from the UTC time, depending on the number of time zones between the locality and Greenwich, England (GMT).

DT_TOUTC converts local time to UTC time.

Converting timestamp values from different localities to a common standard time enables you to sort events into the actual event sequence.

This function requires an IANA (Internet Assigned Numbers Authority) time zone database names (expressed as 'Area/Location') as a parameter. You can find a table of IANA TZ database names on Wikipedia at https://en.wikipedia.org/wiki/List_of_tz_database_time_zones, as shown in the following image.

Legend [edit]

UTC offsets (columns 6 and 7) are positive east of UTC and negative west of UTC. The *UTC DST offset* is different from the *UTC offset* for zones where daylight saving time is observed (see individual time zone pages for details). The UTC offsets are for the current or upcoming rules, and may have been different in the past.

The "Status" field means:

- Canonical - The primary, preferred zone name.
- Alias - An alternative name, which may fit better within a particular country.
- Deprecated - An older style name, left in the tz database for backwards compatibility, which should generally not be used.

List [edit]

Country code	Latitude, longitude ±DDMM(SS) ±DDDMM(SS)	TZ database name	Portion of country covered	Status	UTC offset ±hh:mm	UTC DST offset ±hh:mm	Notes
CI	+0519−00402	Africa/Abidjan		Canonical	+00:00	+00:00	
GH	+0533−00013	Africa/Accra		Canonical	+00:00	+00:00	
ET	+0902+03842	Africa/Addis_Ababa		Alias	+03:00	+03:00	Link to Africa/Nairobi
DZ	+3647+00303	Africa/Algiers		Canonical	+01:00	+01:00	
ER	+1520+03853	Africa/Asmara		Alias	+03:00	+03:00	Link to Africa/Nairobi
ML	+1239−00800	Africa/Bamako		Alias	+00:00	+00:00	Link to Africa/Abidjan
CF	+0422+01835	Africa/Bangui		Alias	+01:00	+01:00	Link to Africa/Lagos
GM	+1328−01639	Africa/Banjul		Alias	+00:00	+00:00	Link to Africa/Abidjan
GW	+1151−01535	Africa/Bissau		Canonical	+00:00	+00:00	
MW	−1547+03500	Africa/Blantyre		Alias	+02:00	+02:00	Link to Africa/Maputo
CG	−0416+01517	Africa/Brazzaville		Alias	+01:00	+01:00	Link to Africa/Lagos
BI	−0323+02922	Africa/Bujumbura		Alias	+02:00	+02:00	Link to Africa/Maputo
EG	+3003+03115	Africa/Cairo		Canonical	+02:00	+02:00	

If you do not know what Area and Location corresponds to your time zone, but you do know your offset from GMT, or your legacy time zone name (such as EST), scroll down in the table. There are TZ database names that correspond to these time zone identifiers, as shown in the following image.

		EST		Deprecated	−05:00	−05:00	Choose a zone that currently *observes* EST without daylight saving time, such as America/Cancun.
		EST5EDT		Deprecated	−05:00	−04:00	Choose a zone that *observes* EST with United States daylight saving time rules, such as America/New_York.
		Etc/GMT		Canonical	+00:00	+00:00	
		Etc/GMT+0		Alias	+00:00	+00:00	Link to Etc/GMT
		Etc/GMT+1		Canonical	−01:00	−01:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+10		Canonical	−10:00	−10:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+11		Canonical	−11:00	−11:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+12		Canonical	−12:00	−12:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+2		Canonical	−02:00	−02:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+3		Canonical	−03:00	−03:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+4		Canonical	−04:00	−04:00	Sign is intentionally inverted. See the Etc area description.
		Etc/GMT+5		Canonical	−05:00	−05:00	Sign is intentionally inverted. See the Etc area description.

Note: If you use a standard IANA time zone database name in the form 'Area/Location' (for example, 'America/New_York'), automatic adjustments are made for Daylight Savings Time. If you use a name that corresponds to an offset from GMT or to a legacy time zone name, it is your responsibility to account for Daylight Savings Time.

Convert Local Time to UTC Time

```
DT_TOUTC(datetime, timezone)
```

where:

datetime

Date-time

Is a date-time expression representing local time, containing date and time components.

timezone

Alphanumeric

Is a character expression containing the IANA time zone name of the local time, in the form 'Area/Location' (for example, 'America/New_York').

Converting Local Time to UTC Time

The following request converts the current local date-time value for time zone America/New_York to UTC time.

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT
COMPUTE LOCAL1/HYYMDS  = DT_CURRENT_DATETIME(SECOND);
COMPUTE UTC1/HYYMDS = DT_TOUTC(LOCAL1, 'America/New_York');
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

LOCAL1	UTC1
2020/09/04 14:49:41	2020/09/04 18:49:41

Sorting by UTC Time

The following request retrieves the current date and time into the field LOCALT1 and sets the field TIMEZONE to IANA time zone database names. It then uses DT_TOUTC to convert the same local time, with different time zones, to the UTC time that corresponds to the given time zone, and sorts the output by the generated UTC time.

```
DEFINE FILE GGSALES
LOCALT1/HYYMDS=DT_CURRENT_DATETIME(SECOND);
TIMEZONE/A30=IF LAST TIMEZONE EQ ' ' THEN 'AMERICA/NEW_YORK'
 ELSE IF LAST TIMEZONE EQ 'AMERICA/NEW_YORK' THEN 'AMERICA/CHICAGO'
 ELSE IF LAST TIMEZONE EQ 'AMERICA/CHICAGO' THEN  'AMERICA/DENVER'
 ELSE IF LAST TIMEZONE EQ 'AMERICA/DENVER' THEN 'ASIA/TOKYO'
 ELSE IF LAST TIMEZONE EQ 'ASIA/TOKYO' THEN 'EUROPE/LONDON'
 ELSE IF LAST TIMEZONE EQ 'EUROPE/LONDON' THEN 'AMERICA/NEW_YORK';
UTCTIME/HYYMDS=DT_TOUTC(LOCALT1,TIMEZONE);
END
TABLE FILE GGSALES
PRINT TIMEZONE LOCALT1 DOLLARS NOPRINT
BY UTCTIME
WHERE PRODUCT EQ 'Thermos'
IF RECORDLIMIT EQ 20
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

UTCTIME	TIMEZONE	LOCALT1
2020/10/02 06:45:59	ASIA/TOKYO	2020/10/02 15:45:59
	ASIA/TOKYO	2020/10/02 15:45:59
	ASIA/TOKYO	2020/10/02 15:45:59
	ASIA/TOKYO	2020/10/02 15:45:59
2020/10/02 14:45:59	EUROPE/LONDON	2020/10/02 15:45:59
	EUROPE/LONDON	2020/10/02 15:45:59
	EUROPE/LONDON	2020/10/02 15:45:59
	EUROPE/LONDON	2020/10/02 15:45:59
2020/10/02 19:45:59	AMERICA/NEW_YORK	2020/10/02 15:45:59
	AMERICA/NEW_YORK	2020/10/02 15:45:59
	AMERICA/NEW_YORK	2020/10/02 15:45:59
	AMERICA/NEW_YORK	2020/10/02 15:45:59
2020/10/02 20:45:59	AMERICA/CHICAGO	2020/10/02 15:45:59
	AMERICA/CHICAGO	2020/10/02 15:45:59
	AMERICA/CHICAGO	2020/10/02 15:45:59
	AMERICA/CHICAGO	2020/10/02 15:45:59
2020/10/02 21:45:59	AMERICA/DENVER	2020/10/02 15:45:59
	AMERICA/DENVER	2020/10/02 15:45:59
	AMERICA/DENVER	2020/10/02 15:45:59
	AMERICA/DENVER	2020/10/02 15:45:59

DTADD: Incrementing a Date or Date-Time Component

Given a date in standard date or date-time format, DTADD returns a new date after adding the specified number of a supported component. The returned date format is the same as the input date format.

Increment a Date or Date-Time Component

```
DTADD(date, component, increment)
```

where:

date

Date or date-time

Is the date or date-time value to be incremented, which must provide a full component date.

component

Keyword

Is the component to be incremented. Valid components (and acceptable values) are:

- YEAR (1-9999).
- QUARTER (1-4).
- MONTH (1-12).
- WEEK (1-53). This is affected by the WEEKFIRST setting.
- DAY (of the Month, 1-31).
- HOUR (0-23).
- MINUTE (0-59).
- SECOND (0-59).

increment

Integer

Is the value (positive or negative) to add to the component.

Incrementing the DAY Component of a Date

The following request against the WF_RETAIL data source adds three days to the employee date of birth:

```
DEFINE FILE WFLITE
NEWDATE/YYMD = DTADD(DATE_OF_BIRTH, DAY, 3);
MGR/A3 = DIGITS(ID_MANAGER, 3);
END
TABLE FILE WFLITE
SUM MGR NOPRINT DATE_OF_BIRTH NEWDATE
BY MGR
ON TABLE SET PAGE NOPAGE
END
```

The output is:

MGR	Date of Birth	NEWDATE
001	1985/01/29	1985/02/01
101	1982/04/01	1982/04/04
201	1976/11/14	1976/11/17
301	1980/05/15	1980/05/18
401	1975/10/19	1975/10/22
501	1985/04/11	1985/04/14
601	1967/02/03	1967/02/06
701	1977/10/16	1977/10/19
801	1970/04/18	1970/04/21
901	1972/03/29	1972/04/01
999	1976/10/21	1976/10/24

Usage Notes for DTADD

- Each element must be manipulated separately. Therefore, if you want to add 1 year and 1 day to a date, you need to call the function twice, once for YEAR (you need to take care of leap years) and once for DAY. The simplified functions can be nested in a single expression, or created and applied in separate DEFINE or COMPUTE expressions.
- With respect to parameter validation, DTADD will not allow anything but a standard date or a date-time value to be used in the first parameter.
- The increment is not checked, and the user should be aware that decimal numbers are not supported and will be truncated. Any combination of values that increases the YEAR beyond 9999 returns the input date as the value, with no message. If the user receives the input date when expecting something else, it is possible there was an error.

DTDIFF: Returning the Number of Component Boundaries Between Date or Date-Time Values

Given two dates in standard date or date-time formats, DTIFF returns the number of given component boundaries between the two dates. The returned value has integer format for calendar components or double precision floating point format for time components.

Return the Number of Component Boundaries

```
DTDIFF(end_date, start_date, component)
```

where:

end_date

Date or date-time

Is the ending full-component date in either standard date or date-time format. If this date is given in standard date format, all time components are assumed to be zero.

start_date

Date or date-time

Is the starting full-component date in either standard date or date-time format. If this date is given in standard date format, all time components are assumed to be zero.

component

Keyword

Is the component on which the number of boundaries is to be calculated. For example, QUARTER finds the difference in quarters between two dates. Valid components (and acceptable values) are:

- YEAR (1-9999).
- QUARTER (1-4).
- MONTH (1-12).

- WEEK (1-53). This is affected by the WEEKFIRST setting.
- DAY (of the Month, 1-31).
- HOUR (0-23).
- MINUTE (0-59).
- SECOND (0-59).

Returning the Number of Years Between Two Dates

The following request against the WF_RETAIL data source calculates employee age when hired:

```
DEFINE FILE WFLITE
YEARS/I9 = DTDIFF(START_DATE, DATE_OF_BIRTH, YEAR);
END
TABLE FILE WFLITE
PRINT START_DATE DATE_OF_BIRTH YEARS AS 'Hire,Age'
BY  EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AA'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

Employee Number	Start Date	Date of Birth	Hire Age
AA100	2008/11/14	1991/06/04	17
AA12	2008/11/19	1985/07/13	23
AA137	2013/01/15	1988/12/24	25
AA174	2013/01/15	1980/08/30	33
AA195	2013/01/15	1977/12/11	36
AA427	2008/12/23	1969/08/08	39
AA820	2013/10/29	1983/11/27	30
AA892	2013/10/27	1981/04/24	32

DTIME: Extracting Time Components From a Date-Time Value

Given a date-time value and time component keyword as input, DTIME returns the value of all of the time components up to and including the requested component. The remaining time components in the value are set to zero. The field to which the time component is returned must have a time format that supports the component being returned.

Extract a Time Component From a Date-Time Value

```
DTIME(datetime, component)
```

where:

datetime

Date-time

Is the date-time value from which to extract the time component. It can be a field name or a date-time literal. It must provide a full component date.

component

Keyword

Valid values are:

- TIME. The complete time portion is returned. Its smallest component depends on the input date-time format. Nanoseconds are not supported or returned.
- HOUR. The time component up to and including the hour component is extracted.
- MINUTE. The time component up to and including the minute component is extracted.
- SECOND. The time component up to and including the second component is extracted.

- MILLISECOND. The time component up to and including the millisecond component is extracted.
- MICROSECOND. The time component up to and including the microsecond component is extracted.

Extracting Time Components

The following request defines two date-time fields:

- TRANSTIME contains the extracted time components from TRANSDATE down to the minute.
- TRANSTIME2 extracts all of the time components from the literal date-time value 2018/01/17 05:45:22.777888.

```
DEFINE FILE VIDEOTR2
TRANSTIME/HHISsm = DTIME(TRANSDATE, MINUTE);
TRANSTIME2/HHISsm = DTIME(DT(2018/01/17 05:45:22.777888), TIME);
END
TABLE FILE VIDEOTR2
SUM  TRANSTIME TRANSTIME2
BY MOVIECODE
BY  TRANSDATE
WHERE MOVIECODE CONTAINS 'MGM'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

MOVIECODE	TRANSDATE	TRANSTIME	TRANSTIME2
145MGM	1999/11/06 02:12	02:12:00.000000	05:45:22.777888
243MGM	1991/06/19 04:11	04:11:00.000000	05:45:22.777888
259MGM	1991/06/19 07:18	07:18:00.000000	05:45:22.777888
284MGM	1999/06/18 03:30	03:30:00.000000	05:45:22.777888
505MGM	1996/06/21 01:16	01:16:00.000000	05:45:22.777888
518MGM	1991/06/24 04:43	04:43:00.000000	05:45:22.777888
	1998/10/03 02:41	02:41:00.000000	05:45:22.777888
	1999/11/18 10:27	10:27:00.000000	05:45:22.777888
688MGM	1998/03/19 07:23	07:23:00.000000	05:45:22.777888
	1999/04/22 06:19	06:19:00.000000	05:45:22.777888
	1999/10/22 06:25	06:25:00.000000	05:45:22.777888
	1999/10/30 06:29	06:29:00.000000	05:45:22.777888
	1999/11/19 10:26	10:26:00.000000	05:45:22.777888

DTPART: Returning a Date or Date-Time Component in Integer Format

Given a date in a standard date or date-time format and a component, DTPART returns the component value in integer format.

Return a Date or Date-Time Component in Integer Format

```
DTPART(date, component)
```

where:

date

Date or date-time

Is the full-component date in standard date or date-time format.

component

Keyword

Is the component to extract in integer format. Valid components (and values) are:

- YEAR (1-9999).
- QUARTER (1-4).
- MONTH (1-12).
- WEEK (of the year, 1-53). This is affected by the WEEKFIRST setting.
- DAY (of the Month, 1-31).
- DAY_OF_YEAR (1-366).
- WEEKDAY (day of the week, 1-7). This is affected by the WEEKFIRST setting.
- HOUR (0-23).
- MINUTE (0-59).
- SECOND (0-59).
- MILLISECOND (0-999).
- MICROSECOND (0-999999).

Extracting the Quarter Component as an Integer

The following request against the WF_RETAIL data source extracts the QUARTER component from the employee start date:

```
DEFINE FILE WFLITE
QTR/I2 = DTPART(START_DATE, QUARTER);
END
TABLE FILE WFLITE
PRINT START_DATE QTR AS Quarter
BY  EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AH'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

Employee Number	Start Date	Quarter
AH118	2013/01/15	1
AH288	2013/11/11	4
AH42	2008/11/13	4
AH928	2009/04/11	2

DTRUNC: Returning the Start of a Date Period for a Given Date

Given a date or timestamp and a component, DTRUNC returns the first date within the period specified by that component.

Return the First or Last Date of a Date Period

```
DTRUNC(date_or_timestamp, date_period, extend)
```

where:

date_or_timestamp

Date or date-time

Is the date or timestamp of interest, which must provide a full component date.

date_period

Is the period whose starting or ending date you want to find. Can be one of the following:

- DAY, returns the date that represents the input date (truncates the time portion, if there is one).
- YEAR, returns the date of the first day of the year.

- MONTH, returns the date of the first day of the month.
- QUARTER, returns the date of the first day in the quarter.
- WEEK, returns the date that represents the first date of the given week.

  By default, the first day of the week will be Sunday, but this can be changed using the WEEKFIRST parameter.
- YEAR_END, returns the last date of the year.
- QUARTER_END, returns the last date of the quarter.
- MONTH_END, returns the last date of the month.
- WEEK_END, returns the last date of the week.

extend

Optional. Is a number that indicates how many of the specified date components to include in the resulting date period.

Since all intervals have to be the same size, the extend argument is limited to the following values for the date period:

- YEAR. No limitations.
- QUARTER. 1 and 2 only.
- MONTH. 1, 2, 3, 4, and 6 only.
- HOUR. 1, 2, 3, 4, 6, and 12 only.
- MINUTE. 1, 2, 3, 4, 5, 6, 10, 15, 20, and 30 only.
- SECOND. 1, 2, 3, 4, 5, 6, 10, 15, 20, and 30 only.

Returning the First Date in a Date Period

In the following request, DTRUNC returns the first date of the quarter given the start date of the employee:

```
DEFINE FILE WFLITE
QTRSTART/YYMD = DTRUNC(START_DATE, QUARTER);
END
TABLE FILE WFLITE
```

```
PRINT START_DATE QTRSTART AS 'Start,of Quarter'
BY EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AH'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

Employee Number	Start Date	Start of Quarter
AH118	2013/01/15	2013/01/01
AH288	2013/11/11	2013/10/01
AH42	2008/11/13	2008/10/01
AH928	2009/04/11	2009/04/01

Using the Start of Week Parameter for DTRUNC

The following request returns the date that is the start of the week for the start date of certain employees:

```
DEFINE FILE WFLITE
DAY1/WT = DTRUNC(START_DATE, DAY);
WKSTART/YYMD = DTRUNC(START_DATE, WEEK);
DAY2/WT = DTRUNC(WKSTART, DAY);
END
TABLE FILE WFLITE
PRINT START_DATE
DAY1 AS 'DOW 1'
WKSTART AS 'Start,of Week'
DAY2 AS 'DOW 2'
BY EMPLOYEE_NUMBER
WHERE START_DATE GT '20130101'
WHERE EMPLOYEE_NUMBER CONTAINS 'AH'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
Employee      Start                 Start
Number        Date         DOW 1    of Week     DOW 2
--------      -----        -----    -------     -----
AH118         2013/01/15   TUE      2013/01/13  SUN
AH2272        2013/01/17   THU      2013/01/13  SUN
AH288         2013/11/11   MON      2013/11/10  SUN
AH3520        2013/09/23   MON      2013/09/22  SUN
AH3591        2013/09/22   SUN      2013/09/22  SUN
AH5177        2013/07/21   SUN      2013/07/21  SUN
```

Returning the Date of the First and Last Days of a Week

The following request returns the dates that correspond to the first day of the week and the last day of the week for the given date.

```
DEFINE FILE WFLITE
WEEKSTART/YYMD = DTRUNC(START_DATE, WEEK);
WEEKEND/YYMD = DTRUNC(START_DATE, WEEK_END);
END
TABLE FILE WFLITE
PRINT START_DATE WEEKSTART AS 'Start,of Week'
WEEKEND AS 'End,of Week'
BY EMPLOYEE_NUMBER
WHERE EMPLOYEE_NUMBER CONTAINS 'AH1'
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.

Employee Number	Start Date	Start of Week	End of Week
AH118	2013/01/15	2013/01/13	2013/01/19
AH1348	2009/11/19	2009/11/15	2009/11/21
AH1398	2009/11/11	2009/11/08	2009/11/14
AH1994	2006/01/01	2006/01/01	2006/01/07

Returning a Date Using the Extend Argument

In the following request, given the date of birth for each employee, the DTRUNC function uses the extend argument to return the start date of the decade in which they were born.

```
DEFINE FILE WFLITE
BIRTH_DECADE/YYMD = DTRUNC(DATE_OF_BIRTH, YEAR, 10);
END
TABLE FILE WFLITE
PRINT DATE_OF_BIRTH BIRTH_DECADE AS 'Start,of Decade'
BY EMPLOYEE_NUMBER
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.

Employee Number	Date of Birth	Start of Decade
AD1804	1975/04/21	1970/01/01
AG5105	1971/02/28	1970/01/01
AT1871	1983/08/04	1980/01/01
BD3005	1975/08/22	1970/01/01
BM1802	1988/12/04	1980/01/01
DW5139	1979/04/02	1970/01/01
HV3086	1977/02/08	1970/01/01
IA1888	1989/08/15	1980/01/01
JF99999	1975/07/03	1970/01/01
JH5164	1970/08/01	1970/01/01
KV5101	1976/12/23	1970/01/01
LE3001	1982/11/05	1980/01/01
MS5102	1986/03/24	1980/01/01
PM5104	1979/05/02	1970/01/01
RA1801	1974/11/14	1970/01/01
RB3033	1977/02/22	1970/01/01
SV3002	1988/09/14	1980/01/01
YS3004	1976/09/13	1970/01/01
ZC1870	1974/05/10	1970/01/01

MONTHNAME: Returning the Name of the Month From a Date Expression

MONTHNAME returns a character string that contains the data-source-specific name of the month for the month part of a date expression.

Return the Name of the Month From a Date Expression

```
MONTHNAME(date_exp)
```

where:

date_exp

Is a date or date-time expression.

Returning the Name of the Month From a Date Expression

The following request returns the name of the month from the TRANSDATE field.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE
COMPUTE TRANSDATE/YYMD= HIRE_DATE; NOPRINT
COMPUTE MONTHNAME1/A12 = MONTHNAME(TRANSDATE);
BY  TRANSDATE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

TRANSDATE	HIRE_DATE	MONTHNAME1
1980/06/02	80/06/02	June
1981/07/01	81/07/01	July
	81/07/01	July
1981/11/02	81/11/02	November
1982/01/04	82/01/04	January
	82/01/04	January
1982/02/02	82/02/02	February
1982/04/01	82/04/01	April
	82/04/01	April
1982/05/01	82/05/01	May
1982/07/01	82/07/01	July
1982/08/01	82/08/01	August

Date Functions

Date functions manipulate date values. There are two types of date functions:

- Standard date functions for use with non-legacy dates.
- Legacy date functions for use with legacy dates.

  If a date is in an alphanumeric or numeric field that contains date display options (for example, I6YMD), you must use the legacy date functions.

Overview of Date Functions

The following explains the difference between the types of date functions:

- **Standard date** functions are for use with standard date formats, or just date formats. A date format refers to internally stored data that is capable of holding date components, such as century, year, quarter, month, and day. It does not include time components. A synonym does not specify an internal data type or length for a date format. Instead, it specifies display date components, such as D (day), M (month), Q (quarter), Y (2-digit year), or YY (4-digit year). For example, format MDYY is a date format that has three date components; it can be used in the USAGE attribute of a synonym. A real date value, such as March 9, 2004, described by this format is displayed as 03/09/2004, by default. Date formats can be full component and non-full component. Full component formats include all three letters, for example, D, M, and Y. JUL for Julian can also be included. All other date formats are non-full component. Some date functions require full component arguments for date fields, while others will accept full or non-full components. A date format was formerly called a smart date.
- **Legacy date** functions are for use with legacy dates only. A legacy date refers to formats with date edit options, such as I6YMD, A6MDY, I8YYMD, or A8MDYY. For example, A6MDY is a 6-byte alphanumeric string. The suffix MDY indicates the order in which the date components are stored in the field, and the prefix I or A indicates a numeric or alphanumeric form of representation. For example, a value '030599' can be assigned to a field with format A6MDY, which will be displayed as 03/05/99.

Date formats have an internal representation matching either numeric or alphanumeric format. For example, A6MDY matches alphanumeric format, YYMD and I6DMY match numeric format. When the function output is a date in specified by *output*, it can be used either for assignment to another date field of this format, or it can be used for further data manipulation in the expression with data of matching formats. Assignment to another field of a different date format, will yield a random result.

For many functions, the output argument can be supplied either as a field name or as a format enclosed in single quotation marks. However, if a function is called from a Dialogue Manager command, this argument must always be supplied as a format. For detailed information about calling a function and supplying arguments, see Accessing and Calling a Function.

Using Standard Date Functions

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

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

- Defining which days of the week are work days and which are not. Then, when you use a date function involving work days, dates that are not work days are ignored. For details, see Using Standard Date Functions.
- Determining whether to display leading zeros when a date function in Dialogue Manager returns a date. For details, see Using Standard Date Functions.

For detailed information on each standard date function, see:

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

DATECVT: Converting the Format of a Date

DATEDIF: Finding the Difference Between Two Dates

DATEMOV: Moving a Date to a Significant Point

DATETRAN: Formatting Dates in International Formats

DPART: Extracting a Component From a Date

FIYR: Obtaining the Financial Year

FIQTR: Obtaining the Financial Quarter

FIYYQ: Converting a Calendar Date to a Financial Date

TODAY: Returning the Current Date

Specifying Work Days

You can determine which days are work days and which are not. Work days affect the DATEADD, DATEDIF, and DATEMOV functions. You identify work days as business days or holidays.

Specifying Business Days

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

Set Business Days

```
SET BUSDAYS = smtwtfs
```

where:

smtwtfs

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

- To identify a day of the week as a business day, enter the first letter of that day in that day's position.
- To identify a non-business day, enter an underscore (_) in that day's position.

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

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
```

View the Current Setting of Business Days

```
? SET BUSDAYS
```

Specifying Holidays

You can specify a list of dates that are designated as holidays in your company. These dates are excluded when using functions that perform calculations based on working days. For example, if Thursday in a given week is designated as a holiday, the next working day after Wednesday is Friday.

To define a list of holidays, you must:

1. Create a holiday file using a standard text editor.
2. Select the holiday file by issuing the SET command with the HDAY parameter.

Rules for Creating a Holiday File

- Dates must be in YYMD format.
- Dates must be in ascending order.
- Each date must be on its own line.
- Each year for which data exists must be included or the holiday file is considered invalid. Calling a date function with a date value outside the range of the holiday file returns a zero for business day requests.

If you are subtracting two dates in 2005, and the latest date in the holiday file is 20041231, the subtraction will not be performed. One way to avoid invalidating the holiday file is to put a date very far in the future in any holiday file you create (for example, 29991231), and then it will always be considered valid.

- You may include an optional description of the holiday, separated from the date by a space.

By default, the holiday file is a member named HDAY**xxxx** of a PDS allocated to DDNAME ERRORS. In your procedure or request, you must issue the SET HDAY=**xxxx** command to identify the file or member name. Alternatively, you can allocate the holiday file as a sequential file of any name or as member HDAY**xxxx** of any PDS. For information about using non-default holiday file names, see Using Standard Date Functions.

Create a Holiday File

Procedure

1. In a text editor, create a list of dates designated as holidays using the Using Standard Date Functions.
2. Save the file.

   If you are not using the default naming convention, see Using Standard Date Functions. If you are using the default naming convention, use the following instructions:

   The file must be a member of ERRORS named HDAY*xxxx*.

   where:

   xxxx

   Is a string of text four characters long.

Select a Holiday File

```
SET HDAY = xxxx
```

where:

xxxx

Is the part of the name of the holiday file after HDAY. This string must be four characters long.

Creating and Selecting a Holiday File

The following is the HDAYTEST file, which establishes holidays:

```
19910325 TEST HOLIDAY
19911225 CHRISTMAS
```

The following sets HDAYTEST as the holiday file:

```
SET BUSDAYS = SMTWTFS
SET HDAY = TEST
```

This request uses HDAYTEST in its calculations:

```
TABLE FILE MOVIES
PRINT TITLE RELDATE
COMPUTE NEXTDATE/YMD = DATEADD(RELDATE, 'BD', 1);
WHERE RELDATE GE '19910101';
END
```

The output is:

```
TITLE                                    RELDATE   NEXTDATE
-----                                    -------   --------
TOTAL RECALL                             91/03/24  91/03/26
```

DYNAM the Holiday File

On z/OS, use the following to allocate a sequential holiday file.

```
DYNAM ALLOC {DD|FILE} HDAYxxxx DA qualif.filename.suffix SHR REU
```

On z/OS, use the following to allocate a holiday file that is a member of a PDS.

```
DYNAM ALLOC {DD|FILE} HDAYxxxx DA qualif.filename.suffix(HDAYxxx) SHR
REU
```

where:

HDAYxxxx

Is the DDNAME for the holiday file. Your FOCEXEC or request must set the HDAY parameter to **xxxx**, where **xxxx** is any four characters you choose. If your holiday file is a member of a PDS, HDAY**xxxx** must also be the member name.

qualif.filename.suffix

Is the fully-qualified name of the sequential file that contains the list of holidays or the PDS with member HDAY**xxxx** that contains the list of holidays.

Allocating the Holiday File to a Sequential File

The following sequential file, named USER1.HOLIDAY.DATA, defines November 3, 2011 and December 24, 2011 as holidays:

```
20111103
20111224
```

The following request against the MOVIES data source uses the DYNAM command to allocate this file as the holiday file. The DDNAME in the DYNAM command is HDAYMMMM, and the procedure issues the SET HDAY=MMMM command. It then defines the date November 2, 2011 and calculates the next business day:

```
DYNAM ALLOC DD HDAYMMMM DA USER1.HOLIDAY.DATA SHR REU
SET HDAY = MMMM
SET BUSDAYS = _MTWTF_
DEFINE FILE MOVIES
NEWDATE/YYMD = '20111102';
NEXTDATE/YYMD = DATEADD(NEWDATE, 'BD', 1);
END
TABLE FILE MOVIES
SUM COPIES NEWDATE NEXTDATE
ON TABLE SET PAGE NOPAGE
END
```

The output shows that the next business day after November 2 is November 4 because November 3 is a holiday:

Allocating the Holiday File to a PDS Member

The following holiday file, member HDAYMMMM in a PDS named USER1.HOLIDAY.DATA, defines November 3, 2011 and December 24, 2011 as holidays:

```
20111103
20111224
```

The following request against the MOVIES data source uses the DYNAM command to allocate this file as the holiday file. The DDNAME in the DYNAM command is HDAYMMMM, the member name is also HDAYMMMM, and the procedure issues the SET HDAY=MMMM command. It then defines the date November 2, 2011 and calculates the next business day:

```
DYNAM ALLOC DD HDAYMMMM DA USER1.HOLIDAY.DATA(HDAYMMMM) SHR REU
SET HDAY = MMMM
SET BUSDAYS = _MTWTF_
DEFINE FILE MOVIES
NEWDATE/YYMD = '20111102';
NEXTDATE/YYMD = DATEADD(NEWDATE, 'BD', 1);
END
TABLE FILE MOVIES
SUM COPIES NEWDATE NEXTDATE
ON TABLE SET PAGE NOPAGE
END
```

The output shows that the next business day after November 2 is November 4 because November 3 is a holiday:

```
COPIES  NEWDATE     NEXTDATE
------  -------     --------
   117  2011/11/02  2011/11/04
```

Enabling Leading Zeros For Date and Time Functions in Dialogue Manager

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

LEADZERO only supports an expression that makes a direct call to a function. An expression that has nesting or another mathematical function always truncates leading zeros. For example,

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

truncates leading zeros regardless of the LEADZERO parameter setting.

Set the Display of Leading Zeros

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

where:

ON

Displays leading zeros if present.

OFF

Truncates leading zeros. OFF is the default value.

Displaying Leading Zeros

The AYM function adds one month to the input date of December 1999:

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

Using the default LEADZERO setting, this yields:

```
1
```

This represents the date January 2000 incorrectly. Setting the LEADZERO parameter in the request as follows:

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

results in the following:

```
0001
```

This correctly indicates January 2000.

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

The DATEADD function adds a unit to or subtracts a unit from a full component date format. A unit is one of the following:

- **Year.**
- **Month.** If the calculation using the month unit creates an invalid date, DATEADD corrects it to the last day of the month. For example, adding one month to October 31 yields November 30, not November 31, since November has 30 days.
- **Day.**
- **Weekday.** When using the weekday unit, DATEADD does not count Saturday or Sunday. For example, if you add one day to Friday, first DATEADD moves to the next

weekday, Monday, then it adds a day. The result is Tuesday.

- **Business day.** When using the business day unit, DATEADD uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. If Monday is not a working day, then one business day past Sunday is Tuesday.

Note that when the DATEADD function calculates the next or previous business day or work day, it always starts from a business day or work day. So if the actual day is Saturday or Sunday, and the request wants to calculate the next business day, the function will use Monday as the starting day, not Saturday or Sunday, and will return Tuesday as the next business day. Similarly, when calculating the previous business day, it will use the starting day Friday, and will return Thursday as the previous business day. You can use the DATEMOV function to move the date to the correct type of day before using DATEADD.

DATEADD requires a date to be in date format. Since Dialogue Manager interprets a date as alphanumeric or numeric, and DATEADD requires a standard date stored as an offset from the base date, do not use DATEADD with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date.

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

DATEADD works only with full component dates.

Add or Subtract a Date Unit to or From a Date

```
DATEADD(date, 'component', increment)
```

where:

date

Date

Is a full component date.

component

Alphanumeric

Is one of the following enclosed in single quotation marks:

`Y` indicates a year component.

`M` indicates a month component.

`D` indicates a day component.

`WD` indicates a weekday component.

`BD` indicates a business day component.

increment

Integer

Is the number of date units added to or subtracted from *date*. If this number is not a whole unit, it is rounded down to the next largest integer.

Note: DATEADD does not use an **output** argument. It uses the format of the **date** argument for the result. As long as the result is a full component date, it can be assigned only to a full component date field or to integer field.

Truncation With DATEADD

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

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

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

Using the Weekday Unit

If you use the weekday unit and a Saturday or Sunday is the input date, DATEADD changes the input date to Monday. The function

```
DATEADD('910623', 'WD', 1)
```

in which DATE is either Saturday or Sunday yields Tuesday; Saturday and Sunday are not weekdays, so DATEADD begins with Monday and adds one.

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

Adding Weekdays to a Date (Reporting)

DATEADD adds three weekdays to NEW_DATE. In some cases, it adds more than three days because HIRE_DATE_PLUS_THREE would otherwise be on a weekend.

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

The output is:

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

Determining If a Date Is a Work Day (Reporting)

DATEADD determines which values in the TRANSDATE field do not represent work days by adding zero days to TRANSDATE using the business day unit. If TRANSDATE does not represent a business day, DATEADD returns the next business day to DATEX. TRANSDATE is then compared to DATEX, and the day of the week is printed for all dates that do not match between the two fields, resulting in a list of all non-work days.

```
DEFINE FILE VIDEOTRK
DATEX/YMD  = DATEADD(TRANSDATE, 'BD', 0);
DATEINT/I8YYMD = DATECVT(TRANSDATE, 'YMD','I8YYMD');
END
```

```
TABLE FILE VIDEOTRK
SUM TRANSDATE NOPRINT
COMPUTE DAYNAME/A8 = DOWKL(DATEINT, DAYNAME); AS 'Day of Week'
BY TRANSDATE AS 'Date'
WHERE TRANSDATE NE DATEX
END
```

The output is:

```
Date      Day of Week
----      -----------
91/06/22  SATURDAY
91/06/23  SUNDAY
91/06/30  SUNDAY
```

DATECVT: Converting the Format of a Date

The DATECVT function converts the field value of any standard date format or legacy date format into a date format (offset from the base date), in the desired standard date format or legacy date format. If you supply an invalid format, DATECVT returns a zero or a blank.

DATECVT turns off optimization and compilation.

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

Convert a Date Format

```
DATECVT(date, 'in_format', output)
```

where:

date

Date

Is the date to be converted. If you supply an invalid date, DATECVT returns zero. When the conversion is performed, a legacy date obeys any DEFCENT and YRTHRESH parameter settings supplied for that field.

in_format

Alphanumeric

Is the format of the date enclosed in single quotation marks. It is one of the following:

- A non-legacy date format (for example, YYMD, YQ, M, DMY, JUL).
- A legacy date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in *in_format* functions as an offset from the base date of a YYMD field (12/31/1900).

output

Alphanumeric

Is the output format enclosed in single quotation marks or a field containing the format. It is one of the following:

- A non-legacy date format (for example, YYMD, YQ, M, DMY, JUL).
- A legacy date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). This format type causes DATECVT to convert the date into a full component date and return it as a whole number in the format provided.

Converting a YYMD Date to DMY

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

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

Or

```
ONV_FIELD/DMY = DATECVT('19991231', 'A8YYMD', 'DMY');
```

Converting a Legacy Date to Date Format (Reporting)

DATECVT converts HIRE_DATE from I6YMD legacy date format to YYMD date format:

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

The output is:

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

DATEDIF: Finding the Difference Between Two Dates

The DATEDIF function returns the difference between two full component standard dates in units of a specified component. A component is one of the following:

- **Year.** Using the year unit with DATEDIF yields the inverse of DATEADD. If subtracting one year from date X creates date Y, then the count of years between X and Y is one. Subtracting one year from February 29 produces the date February 28.
- **Month.** Using the month component with DATEDIF yields the inverse of DATEADD. If subtracting one month from date X creates date Y, then the count of months between X and Y is one. If the to-date is the end-of-month, then the month difference may be rounded up (in absolute terms) to guarantee the inverse rule.

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

- **Day.**
- **Weekday.** With the weekday unit, DATEDIF does not count Saturday or Sunday when calculating days. This means that the difference between Friday and Monday is one day.
- **Business day.** With the business day unit, DATEDIF uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. This means that if Monday is not a working day, the difference between Friday and Tuesday is one day.

DATEDIF returns a whole number. If the difference between two dates is not a whole number, DATEDIF truncates the value to the next largest integer. For example, the number of years between March 2, 2001, and March 1, 2002, is zero. If the end date is before the start date, DATEDIF returns a negative number.

You can find the difference between non-day based dates (for example YM or YQ) directly without using DATEDIF.

Since Dialogue Manager interprets a date as alphanumeric or numeric, and DATEDIF requires a standard date stored as an offset from the base date, do not use DATEDIF with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date.

DATEDIF works only with full component dates.

Find the Difference Between Two Dates

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

where:

from_date

Date

Is the start date from which to calculate the difference. Is a full component date.

to_date

Date

Is the end date from which to calculate the difference.

component

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.

Note: DATEDIF does not use an **output** argument because for the result it uses the format 'I8'.

Truncation With DATEDIF

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

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

Using Month Calculations

The following expressions return a result of minus one month:

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

Additional examples:

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

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

`DATEDIF( 'March 31 2001', 'April 30 2001', 'M')` yields 1.

Finding the Number of Weekdays Between Two Dates (Reporting)

DATECVT converts the legacy dates in HIRE_DATE and DAT_INC to the date format YYMD. DATEDIF then uses those date formats to determine the number of weekdays between NEW_HIRE_DATE and NEW_DAT_INC:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND
COMPUTE NEW_HIRE_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AND
COMPUTE NEW_DAT_INC/YYMD = DATECVT(DAT_INC, 'I6YMD', 'YYMD'); AND
COMPUTE WDAYS_HIRED/I8 = DATEDIF(NEW_HIRE_DATE, NEW_DAT_INC, 'WD');
BY LAST_NAME
IF WDAYS_HIRED NE 0
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME      FIRST_NAME  NEW_HIRE_DATE  NEW_DAT_INC  WDAYS_HIRED
---------      ----------  -------------  -----------  -----------
IRVING         JOAN        1982/01/04     1982/05/14   94
MCKNIGHT       ROGER       1982/02/02     1982/05/14   73
SMITH          RICHARD     1982/01/04     1982/05/14   94
STEVENS        ALFRED      1980/06/02     1982/01/01   414
               ALFRED      1980/06/02     1981/01/01   153
```

DATEMOV: Moving a Date to a Significant Point

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

Note: Using the beginning of week point (BOW) will always return Monday, and using the end of week point (EOW) will always return Friday. Also, if the date used with the DATEMOV function falls on Saturday or Sunday, the actual date used by the function will be the moved forward to the next Monday. If you do not want to do the calculation by moving the date from Saturday or Sunday to Monday, or if you want the BOW to be Sunday and the EOW to be Saturday, you can use the DTRUNC function.

Since Dialogue Manager interprets a date as alphanumeric or numeric, and DATEMOV requires a standard date stored as an offset from the base date, do not use DATEMOV with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date. For example, the following converts the integer legacy date 20050131 to a smart date, adds one month, and converts the result to an alphanumeric legacy date:

```
-SET &STRT=DATECVT(20050131,'I8YYMD', 'YYMD');
-SET &NMT=DATEADD(&STRT,'M',1);
-SET &NMTA=DATECVT(&NMT,'YYMD','A8MTDYY');
-TYPE A MONTH FROM 20050131 IS &NMTA
```

The output shows that the DATEADD function added the actual number of days in the month of February to get to the end of the month from the end of January:

```
A MONTH FROM 20050131 IS 02282005
```

DATEMOV works only with full component dates.

Move a Date to a Significant Point

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

where:

date

Date

Is the date to be moved. It must be a full component format date (for example, MDYY or YYJUL).

move-point

Alphanumeric

Is the significant point the date is moved to be enclosed in single quotation marks ('). An invalid point results in a return code of zero. Valid values are:

- **EOM,** which is the end of month.
- **BOM,** which is the beginning of month.
- **EOQ,** which is the end of quarter.
- **BOQ,** which is the beginning of quarter.
- **EOY,** which is the end of year.
- **BOY,** which is the beginning of year.
- **EOW,** which is the end of week.
- **BOW,** which is the beginning of week.
- **NWD,** which is the next weekday.
- **NBD,** which is the next business day.
- **PWD,** which is the prior weekday.
- **PBD,** which is the prior business day.
- **WD-,** which is a weekday or earlier.
- **BD-,** which is a business day or earlier.
- **WD+,** which is a weekday or later.
- **BD+,** which is a business day or later.

A business day calculation is affected by the BUSDAYS and HDAY parameter settings.

Note that when the DATEADD function calculates the next or previous business day or work day, it always starts from a business day or work day. So if the actual day is Saturday or Sunday, and the request wants to calculate the next business day, the function will use Monday as the starting day, not Saturday or Sunday, and will return Tuesday as the next business day. Similarly, when calculating the previous business day, it will use the starting day Friday, and will return Thursday as the previous business day.

To avoid skipping a business day or work day, use DATEMOV. To return the next business or work day, use BD- or WD- to first move to the previous business or work day (if it is already a business day or work day, it will not be moved). Then use DATEADD to move to the next business or work day. If you want to return the previous business or work day, first use BD+ or WD+ to move to the next business or work day (if it is already the correct type of day, it will not be moved). Then use DATEADD to return the previous business or work day.

Note: DATEMOV does not use an **output** argument. It uses the format of the **date** argument for the result. As long as the result is a full component date, it can be assigned only to a full component date field or to an integer field.

Returning the Next Business Day

This example shows why you may need to use DATEMOV to get the correct result.

The following request against the GGSALES data source uses the BD (Business Day) move point against the DATE field. First DATE is converted to a smart date, then DATEADD is called with the BD move-point:

```
DEFINE FILE GGSALES
DT1/WMDYY=DATE;
DT2/WMDYY = DATEADD(DT1 ,'BD',1);

DAY/Dt = DT1;
 END


TABLE FILE GGSALES

SUM  DT1
DT2
BY DT1 NOPRINT
WHERE RECORDLIMIT EQ 10
 END
```

When the date is on a Saturday or Sunday on the output, the next business day is returned as a Tuesday. This is because before doing the calculation, the original date was moved to a business day:

```
  DT1              DT2
  ---              ---
```

```
  SUN, 09/01/1996  TUE, 09/03/1996
  FRI, 11/01/1996  MON, 11/04/1996
  SUN, 12/01/1996  TUE, 12/03/1996
  SAT, 03/01/1997  TUE, 03/04/1997
  TUE, 04/01/1997  WED, 04/02/1997
  THU, 05/01/1997  FRI, 05/02/1997
  SUN, 06/01/1997  TUE, 06/03/1997
  MON, 09/01/1997  TUE, 09/02/1997
  WED, 10/01/1997  THU, 10/02/1997
```

In the following version of the request, DATEMOV is called to make sure the starting day is a business day. The move point specified in the first call is BD- which only moves the date to the prior business day if it is not already a business day. The call to DATEADD then uses the BD move point to return the next business day:

```
DEFINE FILE GGSALES
DT1/WMDYY=DATE;
DT1A/WMDYY=DATEMOV(DT1, 'BD-');

DT2/WMDYY = DATEADD(DT1A,'BD',1);

DAY/Dt = DT1;
 END


TABLE FILE GGSALES

SUM  DT1 DT1A DT2
BY DT1 NOPRINT
WHERE RECORDLIMIT EQ 10
 END
```

On the output, the next business day after a Saturday or Sunday is now returned as Monday:

```
DT1               DT1A             DT2
  ---               ----             ---
  SUN, 09/01/1996  FRI, 08/30/1996  MON, 09/02/1996
  FRI, 11/01/1996  FRI, 11/01/1996  MON, 11/04/1996
  SUN, 12/01/1996  FRI, 11/29/1996  MON, 12/02/1996
  SAT, 03/01/1997  FRI, 02/28/1997  MON, 03/03/1997
  TUE, 04/01/1997  TUE, 04/01/1997  WED, 04/02/1997
  THU, 05/01/1997  THU, 05/01/1997  FRI, 05/02/1997
  SUN, 06/01/1997  FRI, 05/30/1997  MON, 06/02/1997
```

```
  MON, 09/01/1997  MON, 09/01/1997  TUE, 09/02/1997
  WED, 10/01/1997  WED, 10/01/1997  THU, 10/02/1997
```

Using a DEFINE FUNCTION to Move a Date to the Beginning of the Week

The following DEFINE FUNCTION named BOWK takes a date and the name of the day you want to consider the beginning of the week and returns a date that corresponds to the beginning of the week:

```
DEFINE FUNCTION BOWK(THEDATE/MDYY,WEEKSTART/A10)
DAYOFWEEK/W=THEDATE;
DAYNO/I1=IF DAYOFWEEK EQ 7 THEN 0 ELSE DAYOFWEEK;
FIRSTOFWK/I1=DECODE WEEKSTART('SUNDAY' 0 'MONDAY' 1 'TUESDAY' 2
'WEDNESDAY' 3 'THURSDAY' 4 'FRIDAY' 5 'SATURDAY' 6
'SUN' 0 'MON' 1 'TUE' 2 'WED' 3 'THU' 4 'FRI' 5 'SAT' 6);
BOWK/MDYY=IF DAYNO GE FIRSTOFWK THEN THEDATE-DAYNO+FIRSTOFWK
ELSE THEDATE-7-DAYNO+FIRSTOFWK;
END
```

The following request uses the BOWK function to use return a date (DT2) that corresponds to the beginning of the week for each value of the DT1 field:

```
DEFINE FILE GGSALES
DT1/WMDYY=DATE;
DT2/WMDYY = BOWK(DT1 ,'SUN');
 END


TABLE FILE GGSALES

SUM  DT1
DT2
BY DT1 NOPRINT
WHERE RECORDLIMIT EQ 10
ON TABLE SET PAGE NOLEAD
END
```

The output is shown in the following image:

DT1	DT2
SUN, 09/01/1996	SUN, 09/01/1996
FRI, 11/01/1996	SUN, 10/27/1996
SUN, 12/01/1996	SUN, 12/01/1996
SAT, 03/01/1997	SUN, 02/23/1997
TUE, 04/01/1997	SUN, 03/30/1997
THU, 05/01/1997	SUN, 04/27/1997
SUN, 06/01/1997	SUN, 06/01/1997
MON, 09/01/1997	SUN, 08/31/1997
WED, 10/01/1997	SUN, 09/28/1997

Determining Significant Points for a Date (Reporting)

The BUSDAYS parameter sets the business days to Monday, Tuesday, Wednesday, and Thursday. DATECVT converts the legacy date HIRE_DATE to the date format YYMD and provides date display options. DATEMOV then determines significant points for HIRE_DATE.

```
SET BUSDAY = _MTWT__
TABLE FILE EMPLOYEE
PRINT
COMPUTE NEW_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD'); AND
COMPUTE NEW_DATE/WT = DATECVT(HIRE_DATE, 'I6YMD', 'WT'); AS 'DOW' AND
COMPUTE NWD/WT = DATEMOV(NEW_DATE, 'NWD'); AND
COMPUTE PWD/WT = DATEMOV(NEW_DATE, 'PWD'); AND
COMPUTE WDP/WT = DATEMOV(NEW_DATE, 'WD+'); AS 'WD+' AND
COMPUTE WDM/WT = DATEMOV(NEW_DATE, 'WD-'); AS 'WD-' AND
COMPUTE NBD/WT = DATEMOV(NEW_DATE, 'NBD'); AND
COMPUTE PBD/WT = DATEMOV(NEW_DATE, 'PBD'); AND
COMPUTE WBP/WT = DATEMOV(NEW_DATE, 'BD+'); AS 'BD+' AND
COMPUTE WBM/WT = DATEMOV(NEW_DATE, 'BD-'); AS 'BD-' BY LAST_NAME NOPRINT
HEADING
"Examples of DATEMOV"
"Business days are Monday, Tuesday, Wednesday, + Thursday "
" "
"START DATE.. | MOVE POINTS.........................."
```

```
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
Examples of DATEMOV
Business days are Monday, Tuesday, Wednesday, + Thursday
START DATE.. | MOVE POINTS..........................
NEW_DATE    DOW  NWD  PWD  WD+  WD-  NBD  PBD  BD+  BD-
--------    ---  ---  ---  ---  ---  ---  ---  ---  ---
1982/04/01  THU  FRI  WED  THU  THU  MON  WED  THU  THU
1981/11/02  MON  TUE  FRI  MON  MON  TUE  THU  MON  MON
1982/04/01  THU  FRI  WED  THU  THU  MON  WED  THU  THU
1982/05/01  SAT  TUE  THU  MON  FRI  TUE  WED  MON  THU
1981/07/01  WED  THU  TUE  WED  WED  THU  TUE  WED  WED
1981/07/01  WED  THU  TUE  WED  WED  THU  TUE  WED  WED
```

Determining the End of the Week (Reporting)

DATEMOV determines the end of the week for each date in NEW_DATE and stores the result in EOW:

```
TABLE FILE EMPLOYEE
PRINT FIRST_NAME AND
COMPUTE NEW_DATE/YYMDWT = DATECVT(HIRE_DATE, 'I6YMD', 'YYMDWT'); AND
COMPUTE EOW/YYMDWT = DATEMOV(NEW_DATE, 'EOW');
BY LAST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME      FIRST_NAME  NEW_DATE         EOW
---------      ----------  --------         ---
BANNING        JOHN        1982 AUG  1, SUN  1982 AUG  6, FRI
IRVING         JOAN        1982 JAN  4, MON  1982 JAN  8, FRI
MCKNIGHT       ROGER       1982 FEB  2, TUE  1982 FEB  5, FRI
ROMANS         ANTHONY     1982 JUL  1, THU  1982 JUL  2, FRI
SMITH          RICHARD     1982 JAN  4, MON  1982 JAN  8, FRI
STEVENS        ALFRED      1980 JUN  2, MON  1980 JUN  6, FRI
```

DATETRAN: Formatting Dates in International Formats

The DATETRAN function formats dates in international formats.

Format Dates in International Formats

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

where:

indate

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

intype

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

The following table shows the single component input types:

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

The following table shows the two-component input types:

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

The following table shows the three-component input types:

Three-Component Input Type	Description
`'(YYMD)'`	Four-digit year followed by month followed by day.
`'(YMD)'`	Two-digit year followed by month followed by day.
`'(DMYY)'`	Day component followed by month followed by four-digit year.
`'(DMY)'`	Day component followed by month followed by two-digit year.
`'(MDYY)'`	Month component followed by day followed by four-digit year.

Three-Component Input Type	Description
`'(MDY)'`	Month component followed by day followed by two-digit year.
`'(MD)'`	Month component followed by day (derived from three-component date by ignoring the year component).
`'(DM)'`	Day component followed by month (derived from three-component date by ignoring the year component).

formatops

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

- Options for suppressing initial zeros in month or day numbers.

  Note: Zero suppression replaces initial zeros with blanks spaces.

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

Valid options for suppressing initial zeros in month or day numbers are listed in the following table. Note that the initial zero is replaced by a blank space:

Format Option	Description
`m`	Zero-suppresses months (displays numeric months before October as 1 through 9 rather than 01 through 09).

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

The following table shows valid month and day name translation options:

Format Option	Description
`T`	Displays month as an abbreviated name, with no punctuation, all uppercase.
`TR`	Displays month as a full name, all uppercase.
`Tp`	Displays month as an abbreviated name, followed by a period, all uppercase.
`t`	Displays month as an abbreviated name with no punctuation. The name is all lowercase or initial uppercase, depending on language code.
`tr`	Displays month as a full name. The name is all lowercase or initial uppercase, depending on language code.

Format Option	Description
`tp`	Displays month as an abbreviated name, followed by a period. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`W`	Includes an abbreviated day-of-the-week name at the start of the displayed date, all uppercase with no punctuation.
`WR`	Includes a full day-of-the-week name at the start of the displayed date, all uppercase.
`Wp`	Includes an abbreviated day-of-the-week name at the start of the displayed date, all uppercase, followed by a period.
`w`	Includes an abbreviated day-of-the-week name at the start of the displayed date with no punctuation. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`wr`	Includes a full day-of-the-week name at the start of the displayed date. The name displays in the default case of the specified language (for example, all lowercase for French and Spanish, initial uppercase for English and German).
`wp`	Includes an abbreviated day-of-the-week name at the start of the displayed date followed by a period. The name displays in

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

Format Option	Description
	Spanish, initial uppercase for English and German).

The following table shows valid date delimiter options:

Format Option	Description
B	Uses a blank as the component delimiter. This is the default if the month or day of week is translated or if comma is used.
.	Uses a period (.) as the component delimiter.
–	Uses a minus sign (-) as the component delimiter. This is the default when the conditions for a blank default delimiter are not satisfied.
/	Uses a slash (/) as the component delimiter.
|	Omits component delimiters.
K	Uses appropriate Asian characters as component delimiters.
c	Places a comma (,) after the month name (following T, Tp, TR, t, tp, or tr). Places a comma and blank after the day name (following W, Wp, WR, w, wp, or wr). Places a comma and blank before the day

Format Option	Description
	name (following X, XR, x, or xr).
e	Displays the Spanish or Portuguese word de or DE between the day and month, and between the month and year. The case of the word de is determined by the case of the month name. If the month is displayed in uppercase, DE is displayed. Otherwise, de is displayed. Useful for formats DMY, DMYY, MY, and MYY.
D	Inserts a comma (,) after the day number and before the general delimiter character specified.
Y	Inserts a comma (,) after the year and before the general delimiter character specified.

lang

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

- 'AR' Arabic
- 'CS' Czech
- 'DA' Danish
- 'DE' German
- 'EN' English
- 'ES' Spanish
- 'FI' Finnish
- 'FR' French
- 'EL' Greek
- 'IW' Hebrew
- 'IT' Italian

- 'JA' Japanese
- 'KO' Korean
- 'LT' Lithuanian
- 'NL' Dutch
- 'NO' Norwegian
- 'PO' Polish
- 'PT' Portuguese
- 'RU' Russian
- 'SV' Swedish
- 'TH' Thai
- 'TR' Turkish
- 'TW' Chinese (Traditional)
- 'ZH' Chinese (Simplified)

outlen

Numeric

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

output

Alphanumeric

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

Usage Notes for the DATETRAN Function

- The output field, though it must be type A, and not A*n*V, may in fact contain variable length information, since the lengths of month names and day names can vary, and also month and day numbers may be either one or two bytes long if a zero-suppression option is selected. Unused bytes are filled with blanks.

- All invalid and inconsistent inputs result in all blank output strings. Missing data also results in blank output.
- The base dates (1900-12-31 and 1900-12 or 1901-01) are treated as though the DATEDISPLAY setting were ON (that is, not automatically shown as blanks). To suppress the printing of base dates, which have an internal integer value of 0, test for 0 before calling DATETRAN. For example:

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

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

Using the DATETRAN Function

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

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

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

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

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

The output is:

FORMAT wr

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

Tuesday	Monday
martes	lunes
mardi	lundi
Dienstag	Montag

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

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

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

```
OUT2A/A15=DATETRAN(DATEYYMD,  '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
OUT2B/A15=DATETRAN(DATEYYMD2, '(MDYY)', '(Btdo)', 'EN', 15, 'A15') ;
END
```

```
TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdo"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with suffix (do)"
"English"
""
SUM OUT2A AS '' OUT2B AS '' TRANSDATE NOPRINTON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:

FORMAT Btdo

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

Jan 4th 2005	Mar 2nd 2005

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

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

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

```
OUT3A/A12=DATETRAN(DATEYYMD,  '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
OUT3B/A12=DATETRAN(DATEYYMD2, '(DMYY)', '(Btdp)', 'DE', 12, 'A12');
END
```

```
TABLE FILE VIDEOTRK
HEADING
"FORMAT Btdp"
""
"Blank-delimited (B)"
"Abbreviated month name, default case (t)"
"Zero-suppress day number, end with period (dp)"
"German"
""
SUM OUT3A AS '' OUT3B AS '' TRANSDATE NOPRINTON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:

FORMAT Btdp

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

4. Jan 2005	2. Mär 2005

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

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

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

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

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

The output is:

FORMAT Bwrtr

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

mardi 04 janvier 2005	mercredi 02 mars 2005

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

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

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

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

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

The output is:

FORMAT Bwrctrde Blank-delimited (B) Full day of week name at beginning of date, default case (wr) Comma after day name (c) Full month name, default case (tr) Zero-suppress day number (d) de between day and month and between month and year (e) Spanish	
martes, 4 de enero de 2005	miércoles, 2 de marzo de 2005

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

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

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

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

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

The output is:

FORMAT Ktrd

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

2005年1月4日	2005年3月2日

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

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

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

```
OUT7A/A30=DATETRAN(DATEYYMD , '(DMYY)', '(Bwrctr)', 'GR', 30, 'A30');
OUT7B/A30=DATETRAN(DATEYYMD2, '(DMYY)', '(Bwrctr)', 'GR', 30, 'A30');
END
```

```
TABLE FILE VIDEOTRK
HEADING
"FORMAT Bwrctrde"
""
"Blank-delimited (B)"
"Full day of week name at beginning of date, default case (wr)"
"Comma after day name (c)"
"Full month name, default case (tr)"
"Greek"
""
SUM OUT7A AS '' OUT7B AS '' TRANSDATE NOPRINTON TABLE HOLD FORMAT HTML
ON TABLE SET PAGE-NUM OFF
END
```

The output is:

FORMAT Bwrctr

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

Τρίτη, 04 Ιανουάριος 2005	Τετάρτη, 02 Μάρτιος 2005

DPART: Extracting a Component From a Date

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

Since Dialogue Manager interprets a date as alphanumeric or numeric, and DPART requires a standard date stored as an offset from the base date, do not use DPART with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date.

Extract a Date Component and Return It in Integer Format

```
DPART (datevalue, 'component', output)
```

where:

datevalue

Date

Is a full component date.

component

Alphanumeric

Is the name of the component to be retrieved enclosed in single quotation marks. Valid values are:

For year: YEAR, YY

For month: MONTH, MM

For day: DAY, For day of month: DAY-OF-MONTH, DD.

For weekday: WEEKDAY, WW.

For quarter: QUARTER, QQ

output

Integer

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

Extracting Date Components in Integer Format

The following request against the VIDEOTRK data source uses the DPART function to extract the year, month, and day component from the TRANSDATE field:

```
DEFINE FILE
 VIDEOTRK
 YEAR/I4 = DPART(TRANSDATE, 'YEAR', 'I11');
 MONTH/I4 = DPART(TRANSDATE, 'MM', 'I11');
 DAY/I4 = DPART(TRANSDATE, 'DAY', 'I11');
END

TABLE FILE VIDEOTRK
PRINT TRANSDATE YEAR MONTH DAY
BY LASTNAME BY FIRSTNAME
WHERE LASTNAME LT 'DIAZ'
END
```

The output is:

```
LASTNAME         FIRSTNAME   TRANSDATE  YEAR  MONTH   DAY
--------         ---------   ---------  ----  -----   ---
ANDREWS          NATALIA     91/06/19   1991      6    19
                             91/06/18   1991      6    18
BAKER            MARIE       91/06/19   1991      6    19
                             91/06/17   1991      6    17
BERTAL           MARCIA      91/06/23   1991      6    23
                             91/06/18   1991      6    18
CHANG            ROBERT      91/06/28   1991      6    28
                             91/06/27   1991      6    27
                             91/06/26   1991      6    26
COLE             ALLISON     91/06/24   1991      6    24
                             91/06/23   1991      6    23
CRUZ             IVY         91/06/27   1991      6    27
DAVIS            JASON       91/06/24   1991      6    24
```

FIQTR: Obtaining the Financial Quarter

The FIQTR function returns the financial quarter corresponding to a given calendar date based on the financial year starting date and the financial year numbering convention.

Since Dialogue Manager interprets a date as alphanumeric or numeric, and FIQTR requires a standard date stored as an offset from the base date, do not use FIQTR with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date.

Obtain the Financial Quarter

```
FIQTR(inputdate, lowcomponent, startmonth, startday, yrnumbering,
output)
```

where:

inputdate

Date

Is the date for which the financial year is returned. The date must be a standard date stored as an offset from the base date.

If the financial year does not begin on the first day of a month, the date must have Y(Y), M, and D components, or Y(Y) and JUL components (note that JUL is equivalent to YJUL). Otherwise, the date only needs Y(Y) and M components or Y(Y) and Q components.

lowcomponent

Alphanumeric

Is one of the following:

- `D` if the date contains a D or JUL component.
- `M` if the date contains an M component, but no D component.
- `Q` if the date contains a Q component.

startmonth

Numeric

1 through 12 are used to represent the starting month of the financial year, where 1 represents January and 12 represents December. If the low component is Q, the start month must be 1, 4, 7, or 10.

startday

Numeric

Is the starting day of the starting month, usually 1. If the low component is M or Q, 1 is required.

yrnumbering

Alphanumeric

Valid values are:

FYE to specify the **Financial Year Ending** convention. The financial year number is the calendar year of the ending date of the financial year. For example, when the financial year starts on October 1, 2008, the date 2008 November 1 is in FY 2009 Q1 because that date is in the financial year that ends on 2009 September 30.

FYS to specify the **Financial Year Starting** convention. The financial year number is the calendar year of the starting date of the financial year. For example, when the financial year starts on April 6, 2008, the date 2008 July 6 is in FY 2008 Q2 because that date is in the financial year that starts on 2008 April 6.

output

I or Q

The result will be in integer format, or Q. This function will return a value of 1 through 4. In case of an error, zero is returned.

Note: February 29 cannot be used as a start day for a financial year.

Obtaining the Financial Quarter

The following request against the CENTHR data source obtains the financial quarter corresponding to an employee starting date (field START_DATE, format YYMD) and returns the values in each of the supported formats: Q and I1.

```
DEFINE FILE CENTHR
FISCALQ/Q=FIQTR(START_DATE,'D',10,1,'FYE',FISCALQ);
FISCALI/I1=FIQTR(START_DATE,'D',10,1,'FYE',FISCALI);
END
TABLE FILE CENTHR
PRINT START_DATE FISCALQ FISCALI
BY LNAME BY FNAME
WHERE LNAME LIKE 'C%'
END
```

On the output, note that the date November 12, 1998 (1998/11/12) is in fiscal quarter Q1 because the starting month is October (10):

```
Last             First          Starting
Name             Name           Date         FISCALQ  FISCALI
----             -----          --------     -------  -------
CHARNEY          ROSS           1998/09/12   Q4             4
CHIEN            CHRISTINE      1997/10/01   Q1             1
CLEVELAND        PHILIP         1996/07/30   Q4             4
CLINE            STEPHEN        1998/11/12   Q1             1
COHEN            DANIEL         1997/10/05   Q1             1
CORRIVEAU        RAYMOND        1997/12/05   Q1             1
COSSMAN          MARK           1996/12/19   Q1             1
CRONIN           CHRIS          1996/12/03   Q1             1
```

```
CROWDER         WESLEY        1996/09/17  Q4             4
CULLEN          DENNIS        1995/09/05  Q4             4
CUMMINGS        JAMES         1993/07/11  Q4             4
CUTLIP          GREGG         1997/03/26  Q2             2
```

FIYR: Obtaining the Financial Year

The FIYR function returns the financial year, also known as the fiscal year, corresponding to a given calendar date based on the financial year starting date and the financial year numbering convention.

Since Dialogue Manager interprets a date as alphanumeric or numeric, and FIYR requires a standard date stored as an offset from the base date, do not use FIYR with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date.

Obtain the Financial Year

```
FIYR(inputdate, lowcomponent, startmonth, startday, yrnumbering, output)
```

where:

inputdate

Date

Is the date for which the financial year is returned. The date must be a standard date stored as an offset from the base date.

If the financial year does not begin on the first day of a month, the date must have Y(Y), M, and D components, or Y(Y) and JUL components (note that JUL is equivalent to YJUL). Otherwise, the date only needs Y(Y) and M components or Y(Y) and Q components.

lowcomponent

Alphanumeric

Is one of the following:

- `D` if the date contains a D or JUL component.
- `M` if the date contains an M component, but no D component.
- `Q` if the date contains a Q component.

startmonth

Numeric

1 through 12 are used to represent the starting month of the financial year, where 1 represents January and 12 represents December. If the low component is Q, the start month must be 1, 4, 7, or 10.

startday

Numeric

Is the starting day of the starting month, usually 1. If the low component is M or Q, 1 is required.

yrnumbering

Alphanumeric

Valid values are:

FYE to specify the **Financial Year Ending** convention. The financial year number is the calendar year of the ending date of the financial year. For example, when the financial year starts on October 1, 2008, the date 2008 November 1 is in FY 2009 Q1 because that date is in the financial year that ends on 2009 September 30.

FYS to specify the **Financial Year Starting** convention. The financial year number is the calendar year of the starting date of the financial year. For example, when the financial year starts on April 6, 2008, the date 2008 July 6 is in FY 2008 Q2 because that date is in the financial year that starts on 2008 April 6.

output

I, Y, or YY

The result will be in integer format, or Y or YY. This function returns a year value. In case of an error, zero is returned.

Note: February 29 cannot be used as a start day for a financial year.

Obtaining the Financial Year

The following request against the CENTSTMT data source obtains the financial year corresponding to an account period (field PERIOD, format YYM) and returns the values in each of the supported formats: Y, YY, and I4.

```
DEFINE FILE CENTSTMT
FISCALYY/YY=FIYR(PERIOD,'M', 4,1,'FYE',FISCALYY);
FISCALY/Y=FIYR(PERIOD,'M', 4,1,'FYE',FISCALY);
FISCALI/I4=FIYR(PERIOD,'M', 4,1,'FYE',FISCALI);
END
TABLE FILE CENTSTMT
PRINT PERIOD FISCALYY FISCALY FISCALI
BY GL_ACCOUNT
WHERE GL_ACCOUNT LT '2100'
END
```

On the output, note that the period April 2002 (2002/04) is in fiscal year 2003 because the starting month is April (4), and the FYE numbering convention is used:

```
Ledger
Account  PERIOD   FISCALYY  FISCALY  FISCALI
-------  ------   --------  -------  -------
1000     2002/01  2002      02          2002
         2002/02  2002      02          2002
         2002/03  2002      02          2002
         2002/04  2003      03          2003
         2002/05  2003      03          2003
         2002/06  2003      03          2003
2000     2002/01  2002      02          2002
         2002/02  2002      02          2002
         2002/03  2002      02          2002
         2002/04  2003      03          2003
         2002/05  2003      03          2003
         2002/06  2003      03          2003
```

FIYYQ: Converting a Calendar Date to a Financial Date

The FIYYQ function returns a financial date containing both the financial year and quarter that corresponds to a given calendar date. The returned financial date is based on the financial year starting date and the financial year numbering convention.

Since Dialogue Manager interprets a date as alphanumeric or numeric, and FIYYQ requires a standard date stored as an offset from the base date, do not use FIYYQ with Dialogue Manager unless you first convert the variable used as the input date to an offset from the base date.

Convert a Calendar Date to a Financial Date

```
FIYYQ(inputdate, lowcomponent, startmonth, startday, yrnumbering,
output)
```

where:

inputdate

Date

Is the date for which the financial year is returned. The date must be a standard date stored as an offset from the base date.

If the financial year does not begin on the first day of a month, the date must have Y(Y), M, and D components, or Y(Y) and JUL components (note that JUL is equivalent to YJUL). Otherwise, the date only needs Y(Y) and M components or Y(Y) and Q components.

lowcomponent

Alphanumeric

Is one of the following:

- `D` if the date contains a D or JUL component.
- `M` if the date contains an M component, but no D component.
- `Q` if the date contains a Q component.

startmonth

Numeric

1 through 12 are used to represent the starting month of the financial year, where 1 represents January and 12 represents December. If the low component is Q, the start month must be 1, 4, 7, or 10.

startday

Numeric

Is the starting day of the starting month, usually 1. If the low component is M or Q, 1 is required.

yrnumbering

Alphanumeric

Valid values are:

FYE to specify the **Financial Year Ending** convention. The financial year number is the calendar year of the ending date of the financial year. For example, when the financial year starts on October 1, 2008, the date 2008 November 1 is in FY 2009 Q1 because that date is in the financial year that ends on 2009 September 30.

FYS to specify the **Financial Year Starting** convention. The financial year number is the calendar year of the starting date of the financial year. For example, when the financial year starts on April 6, 2008, the date 2008 July 6 is in FY 2008 Q2 because that date is in the financial year that starts on 2008 April 6.

output

Y[Y]Q or QY[Y]

In case of an error, zero is returned.

Note: February 29 cannot be used as a start day for a financial year.

Converting a Calendar Date to a Financial Date

The following request against the CENTHR data source converts each employee starting date (field START_DATE, format YYMD) to a financial date containing year and quarter components in all the supported formats: YQ, YYQ, QY, and QYY.

```
DEFINE FILE CENTHR
FISYQ/YQ=FIYYQ(START_DATE,'D',10,1,'FYE',FISYQ);
FISYYQ/YYQ=FIYYQ(START_DATE,'D',10,1,'FYE',FISYYQ);
FISQY/QY=FIYYQ(START_DATE,'D',10,1,'FYE',FISQY);
FISQYY/QYY=FIYYQ(START_DATE,'D',10,1,'FYE',FISQYY);
END
TABLE FILE CENTHR
PRINT START_DATE FISYQ FISYYQ FISQY FISQYY
BY LNAME BY FNAME
WHERE LNAME LIKE 'C%'
END
```

On the output, note that the date November 12, 1998 (1998/11/12) is converted to Q1 1999 because the starting month is October (10), and the FYE numbering convention is used:

```
Last            First          Starting
Name            Name           Date        FISYQ  FISYYQ   FISQY  FISQYY
----            -----          --------    -----  ------   -----  ------
CHARNEY         ROSS           1998/09/12  98 Q4  1998 Q4  Q4 98  Q4 1998
CHIEN           CHRISTINE      1997/10/01  98 Q1  1998 Q1  Q1 98  Q1 1998
CLEVELAND       PHILIP         1996/07/30  96 Q4  1996 Q4  Q4 96  Q4 1996
CLINE           STEPHEN        1998/11/12  99 Q1  1999 Q1  Q1 99  Q1 1999
COHEN           DANIEL         1997/10/05  98 Q1  1998 Q1  Q1 98  Q1 1998
CORRIVEAU       RAYMOND        1997/12/05  98 Q1  1998 Q1  Q1 98  Q1 1998
COSSMAN         MARK           1996/12/19  97 Q1  1997 Q1  Q1 97  Q1 1997
CRONIN          CHRIS          1996/12/03  97 Q1  1997 Q1  Q1 97  Q1 1997
CROWDER         WESLEY         1996/09/17  96 Q4  1996 Q4  Q4 96  Q4 1996
CULLEN          DENNIS         1995/09/05  95 Q4  1995 Q4  Q4 95  Q4 1995
CUMMINGS        JAMES          1993/07/11  93 Q4  1993 Q4  Q4 93  Q4 1993
CUTLIP          GREGG          1997/03/26  97 Q2  1997 Q2  Q2 97  Q2 1997
```

TODAY: Returning the Current Date

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

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

A compiled MODIFY procedure must use TODAY to obtain the date. It cannot use the system variables.

Retrieve the Current Date

```
TODAY(output)
```

where:

output

Alphanumeric, at least A8

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

The following apply:

- If the format is A8 or A9, TODAY returns the 2-digit year.
- If the format is A10 or greater, TODAY returns the 4-digit year.

Retrieving the Current Date

TODAY retrieves the current date and stores it in the DATE field. The request then displays the date in the page heading.

```
DEFINE FILE EMPLOYEE
DATE/A10 WITH EMP_ID = TODAY(DATE);
END
```

```
TABLE FILE EMPLOYEE
SUM CURR_SAL BY DEPARTMENT
HEADING
"PAGE <TABPAGENO  "
"SALARY REPORT RUN ON <DATE  "
END
```

The output is:

```
SALARY REPORT RUN ON 12/13/2006
DEPARTMENT       CURR_SAL
----------      --------
MIS            $108,002.00
PRODUCTION     $114,282.00
```

Using Legacy Date Functions

The legacy date functions were created for use with dates in integer, packed decimal, or alphanumeric format.

For detailed information on each legacy date function, see:

AYM: Adding or Subtracting Months

AYMD: Adding or Subtracting Days

CHGDAT: Changing How a Date String Displays

DA Functions: Converting a Legacy Date to an Integer

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

DOWK and DOWKL: Finding the Day of the Week

DT Functions: Converting an Integer to a Date

GREGDT: Converting From Julian to Gregorian Format

JULDAT: Converting From Gregorian to Julian Format

YM: Calculating Elapsed Months

Using Old Versions of Legacy Date Functions

The functions described in this section are legacy date functions. They were created for use with dates in integer or alphanumeric format. They are no longer recommended for date manipulation. Standard date and date-time functions are preferred.

All legacy date functions support dates for the year 2000 and later.

Using Dates With Two- and Four-Digit Years

Legacy date functions accept dates with two- or four-digit years. Four-digit years that display the century, such as 2000 or 1900, can be used if their formats are specified as I8YYMD, P8YYMD, D8YYMD, F8YYMD, or A8YYMD. Two-digit years can use the DEFCENT and YRTHRESH parameters to assign century values if the field has a length of six (for example, I6YMD). For information on these parameters, see *Customizing Your Environment* in the *ibi™ FOCUS® Developing Applications* manual.

Using Four-Digit Years

The EDIT function creates dates with four-digit years. The functions JULDAT and GREGDT then convert these dates to Julian and Gregorian formats.

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

The output is:

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

Using Two-Digit Years

The AYMD function returns an eight-digit date when the input argument has a six-digit legacy date format. Since DEFCENT is 19 and YRTHRESH is 83, year values from 83 through 99 are interpreted as 1983 through 1999, and year values from 00 through 82 are interpreted as 2000 through 2082.

```
SET DEFCENT=19, YRTHRESH=83

DEFINE FILE EMPLOYEE
NEW_DATE/I8YYMD = AYMD(EFFECT_DATE, 30, 'I8');
END

TABLE FILE EMPLOYEE
PRINT EFFECT_DATE NEW_DATE BY EMP_ID
END
```

The output is:

```
EMP_ID     EFFECT_DATE    NEW_DATE
------     -----------    --------
071382660
112847612
117593129     82/11/01  2082/12/01
119265415
119329144     83/01/01  1983/01/31
123764317     83/03/01  1983/03/31
126724188
219984371
326179357     82/12/01  2082/12/31
451123478     84/09/01  1984/10/01
543729165
818692173     83/05/01  1983/05/31
```

AYM: Adding or Subtracting Months

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

Add or Subtract Months to or From a Date

```
AYM(indate, months, output)
```

where:

indate

I4, I4YM, I6, or I6YYM

Is the legacy date in year-month format, the name of a field that contains the date, or an expression that returns the date. If the date is not valid, the function returns the value 0 (zero).

months

Integer

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

output

I4YM or I6YYM

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

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

Adding Months to a Date

The COMPUTE command converts the dates in HIRE_DATE from year-month-day to year-month format and stores the result in HIRE_MONTH. AYM then adds six months to HIRE_MONTH and stores the result in AFTER6MONTHS:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100 ;
AFTER6MONTHS/I4YM = AYM(HIRE_MONTH, 6, AFTER6MONTHS);
```

```
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

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

AYMD: Adding or Subtracting Days

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

Add or Subtract Days to or From a Date

```
AYMD(indate, days, output)
```

where:

indate

I6, I6YMD, I8, I8YYMD

Is the legacy date in year-month-day format. If the date is not valid, the function returns the value 0 (zero).

days

Integer

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

output

I6, I6YMD, I8, or I8YYMD

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. If *indate* is a field, the *output* must have the same format.

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

Adding Days to a Date

AYMD adds 35 days to each value in the HIRE_DATE field, and stores the result in AFTER35DAYS:

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

The output is:

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

CHGDAT: Changing How a Date String Displays

The CHGDAT function rearranges the year, month, and day portions of an input character string representing a date. It may also convert the input string from long to short or short to long date representation. Long representation contains all three date components: year, month, and day; short representation omits one or two of the date components, such as year, month, or day. The input and output date strings are described by display options that specify both the order of date components (year, month, day) in the date string and whether two or four digits are used for the year (for example, 04 or 2004). CHGDAT reads an input date character string and creates an output date character string that represents the same date in a different way.

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

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

Character	Description
`D`	Day of the month (01 through 31).
`M`	Month of the year (01 through 12).
`Y[Y]`	Year. `Y` indicates a two-digit year (such as 94); `YY` indicates a four-digit year (such as 1994).

To spell out the month rather than use a number in the resulting string, append one of the following characters to the display options for the resulting string:

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

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

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

Note: Display options are *not* date formats.

Short to Long Conversion

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

Portion of Date Missing	Portion Supplied by Function
Day (for example, from YM to YMD)	Last day of the month.
Month (for example, from Y to YM)	Last month of the year (December).
Year (for example, from MD to YMD)	The year 99.
Converting the year from two-digit to four-digit (for example, from YMD to YYMD)	The century will be determined by the 100-year window defined by DEFCENT and YRTHRESH. See *Working With Cross-Century Dates* in the *ibi™ FOCUS® Developing Applications* manual for details on DEFCENT and YRTHRESH.

Change the Date Display String

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

where:

'in_display_options'

A1 to A5

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

'out_display_options'

A1 to A5

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

date_string

A2 to A8

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

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

output

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

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

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

Converting the Date Display From YMD to MDYYX

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

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

The output is:

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

DA Functions: Converting a Legacy Date to an Integer

The DA functions convert a legacy date to the number of days between it and a base date (December 31, 1899). By converting a date to the number of days, you can add and subtract dates and calculate the intervals between them, or you can add to or subtract numbers from the dates to get new dates.

There are six DA functions; each one accepts a date in a different format.

Convert a Date to an Integer

```
function(indate, output)
```

where:

function

Is one of the following:

`DADMY` converts a date in day-month-year format.

`DADYM` converts a date in day-year-month format.

`DAMDY` converts a date in month-day-year format.

`DAMYD` converts a date in month-year-day format.

`DAYDM` converts a date in year-day-month format.

`DAYMD` converts a date in year-month-day format.

indate

I6xxx or P6xxx, where xxx corresponds to the function DAxxx you are using.

Is the legacy date to be converted, or the name of a field that contains the date. The date is truncated to an integer before conversion. If **indate** is a numeric literal, enter only the last two digits of the year; the function assumes the century component. If the date is invalid, the function returns a `0`.

output

Integer

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format of the date returned depends on the function.

Converting Dates and Calculating the Difference Between Them

DAYMD converts the DAT_INC and HIRE_DATE fields to the number of days since December 31, 1899, and the smaller number is then subtracted from the larger number:

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
DAYS_HIRED/I8 = DAYMD(DAT_INC, 'I8') - DAYMD(HIRE_DATE, 'I8');
```

```
BY LAST_NAME BY FIRST_NAME
IF DAYS_HIRED NE 0
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

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

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

The DMY, MDY, and YMD functions calculate the difference between two legacy dates in integer, alphanumeric, or packed format.

Calculate the Difference Between Two Dates

```
function(from_date, to_date)
```

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.

from_date

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

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

to_date

I, P, or A format with date display options.I6xxx or I8xxx where xxx corresponds to the specified function (DMY, YMD, or MDY).

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

Calculating the Number of Days Between Two Dates

YMD calculates the number of days between the dates in HIRE_DATE and DAT_INC:

```
TABLE FILE EMPLOYEE
SUM HIRE_DATE FST.DAT_INC AS 'FIRST PAY,INCREASE' AND COMPUTE
DIFF/I4 = YMD(HIRE_DATE, FST.DAT_INC); AS 'DAYS,BETWEEN'
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
                                            FIRST PAY  DAYS
LAST_NAME        FIRST_NAME  HIRE_DATE  INCREASE   BETWEEN
---------        ----------  ---------  ---------  -------
BLACKWOOD        ROSEMARIE    82/04/01   82/04/01        0
CROSS            BARBARA      81/11/02   82/04/09      158
GREENSPAN        MARY         82/04/01   82/06/11       71
JONES            DIANE        82/05/01   82/06/01       31
MCCOY            JOHN         81/07/01   82/01/01      184
SMITH            MARY         81/07/01   82/01/01      184
```

DOWK and DOWKL: Finding the Day of the Week

The DOWK and DOWKL functions find the day of the week that corresponds to a date. DOWK returns the day as a three letter abbreviation; DOWKL displays the full name of the day.

Find the Day of the Week

```
{DOWK|DOWKL}(indate, output)
```

where:

indate

I6YMD or I8YYMD

Is the legacy date in year-month-day format. If the date is not valid, the function returns spaces. If the date specifies a two-digit year and the DEFCENT and YRTHRESH values have not been set, the function assumes the 20th century.

output

DOWK: A4. DOWKL: A12

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

Finding the Day of the Week

DOWK determines the day of the week that corresponds to the value in the HIRE_DATE field and stores the result in DATED:

```
TABLE FILE EMPLOYEE
PRINT EMP_ID AND HIRE_DATE AND COMPUTE
DATED/A4 = DOWK(HIRE_DATE, DATED);
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
EMP_ID     HIRE_DATE  DATED
------     ---------  -----
071382660   80/06/02  MON
119265415   82/01/04  MON
119329144   82/08/01  SUN
123764317   82/01/04  MON
126724188   82/07/01  THU
451123478   82/02/02  TUE
```

DT Functions: Converting an Integer to a Date

The DT functions convert an integer representing the number of days elapsed since December 31, 1899 to the corresponding date. They are useful when you are performing arithmetic on a date converted to the number of days. The DT functions convert the result back to a date.

There are six DT functions; each one converts a number into a date of a different format.

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

Convert an Integer to a Date

```
function(number, output)
```

where:

function

Is one of the following:

`DTDMY` converts a number to a day-month-year date.

`DTDYM` converts a number to a day-year-month date.

`DTMDY` converts a number to a month-day-year date.

DTMYD converts a number to a month-year-day date.

DTYDM converts a number to a year-day-month date.

DTYMD converts a number to a year-month-day date.

number

Integer

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

output

I8xxx, where xxx corresponds to the function DTxxx in the above list.

Is the name of the field containing the result or the format of the output value enclosed in single quotation marks. The output format depends on the function being used.

Converting an Integer to a Date

DTMDY converts the NEWF field (which was converted to the number of days by DAYMD) to the corresponding date and stores the result in NEW_HIRE_DATE:

```
-* THIS PROCEDURE CONVERTS HIRE_DATE, WHICH IS IN I6YMD FORMAT,
-* TO A DATE IN I8MDYY FORMAT.
-* FIRST IT USES THE DAYMD FUNCTION TO CONVERT HIRE_DATE
-* TO A NUMBER OF DAYS.
-* THEN IT USES THE DTMDY FUNCTION TO CONVERT THIS NUMBER OF
-* DAYS TO I8MDYY FORMAT
-*
DEFINE FILE EMPLOYEE
NEWF/I8 WITH EMP_ID = DAYMD(HIRE_DATE, NEWF);
NEW_HIRE_DATE/I8MDYY WITH EMP_ID = DTMDY(NEWF, NEW_HIRE_DATE);
END
TABLE FILE EMPLOYEE
PRINT HIRE_DATE NEW_HIRE_DATE
BY FN BY LN
WHERE DEPARTMENT EQ 'MIS'
END
```

The output is:

```
FIRST_NAME  LAST_NAME        HIRE_DATE  NEW_HIRE_DATE
----------  ---------        ---------  -------------
```

```
BARBARA      CROSS               81/11/02     11/02/1981
DIANE        JONES               82/05/01     05/01/1982
JOHN         MCCOY               81/07/01     07/01/1981
MARY         GREENSPAN           82/04/01     04/01/1982
             SMITH               81/07/01     07/01/1981
ROSEMARIE    BLACKWOOD           82/04/01     04/01/1982
```

GREGDT: Converting From Julian to Gregorian Format

The GREGDT function converts a date in Julian format (year-day) to Gregorian format (year-month-day).

A date in Julian format is a five- or seven-digit number. The first two or four digits are the year; the last three digits are the number of the day, counting from January 1. For example, January 1, 1999 in Julian format is either 99001 or 1999001; June21, 2004 in Julian format is 2004173.

Format Options for GREGDT

GREGDT converts a Julian date to either YMD or YYMD format using the DEFCENT and YRTHRESH parameter settings to determine the century, if required. GREGDT returns a date as follows:

- If the format is I6 or I7, GREGDT returns the date in YMD format.
- If the format is I8 or greater, GREGDT returns the date in YYMD format.

Convert From Julian to Gregorian Format

```
GREGDT(indate, output)
```

where:

indate

I5 or I7

Is the Julian date, which is truncated to an integer before conversion. Each value must be a five- or seven-digit number after truncation. If the date is invalid, the function returns a 0 (zero).

output

I6, I8, I6YMD, or I8YYMD

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

Converting From Julian to Gregorian Format

GREGDT converts the JULIAN field to YYMD (Gregorian) format. It determines the century using the default DEFCENT and YRTHRESH parameter settings.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND
COMPUTE JULIAN/I5 = JULDAT(HIRE_DATE, JULIAN); AND
COMPUTE GREG_DATE/I8 = GREGDT(JULIAN, 'I8');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME      FIRST_NAME  HIRE_DATE  JULIAN  GREG_DATE
---------      ----------  ---------  ------  ---------
BANNING        JOHN          82/08/01   82213   19820801
IRVING         JOAN          82/01/04   82004   19820104
MCKNIGHT       ROGER         82/02/02   82033   19820202
ROMANS         ANTHONY       82/07/01   82182   19820701
SMITH          RICHARD       82/01/04   82004   19820104
STEVENS        ALFRED        80/06/02   80154   19800602
```

JULDAT: Converting From Gregorian to Julian Format

The JULDAT function converts a date from Gregorian format (year-month-day) to Julian format (year-day). A date in Julian format is a five- or seven-digit number. The first two or four digits are the year; the last three digits are the number of the day, counting from January 1. For example, January 1, 1999 in Julian format is either 99001 or 1999001.

Format Settings for JULDAT

JULDAT converts a Gregorian date to either YYNNN or YYYYNNN format, using the DEFCENT and YRTHRESH parameter settings to determine if the century is required.

JULDAT returns dates as follows:

- If the format is I6, JULDAT returns the date in YYNNN format.
- If the format is I7 or greater, JULDAT returns the date in YYYYNNN format.

Convert From Gregorian to Julian Format

```
JULDAT(indate, output)
```

where:

indate

I6, I8, I6YMD, I8YYMD

Is the legacy date to convert or the name of the field that contains the date in year-month-day format (YMD or YYMD).

output

I5 or I7

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

Converting From Gregorian to Julian Format

JULDAT converts the HIRE_DATE field to Julian format. It determines the century using the default DEFCENT and YRTHRESH parameter settings.

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
JULIAN/I7 = JULDAT(HIRE_DATE, JULIAN);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME      FIRST_NAME  HIRE_DATE   JULIAN
---------      ----------  ---------   ------
BANNING        JOHN          82/08/01  1982213
IRVING         JOAN          82/01/04  1982004
MCKNIGHT       ROGER         82/02/02  1982033
ROMANS         ANTHONY       82/07/01  1982182
SMITH          RICHARD       82/01/04  1982004
STEVENS        ALFRED        80/06/02  1980154
```

YM: Calculating Elapsed Months

The YM function calculates the number of months between two dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT or EDIT function.

Calculate Elapsed Months

```
YM(fromdate, todate, output)
```

where:

fromdate

I4YM or I6YYM

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

todate

I4YM or I6YYM

Is the end date in year-month format. If the date is not valid, the function returns the value 0 (zero).

output

Integer

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

Tip: If *fromdate* or *todate* is in integer year-month-day format (I6YMD or I8YYMD), simply divide by 100 to convert to year-month format and set the result to an integer. This drops the day portion of the date, which is now after the decimal point.

Calculating Elapsed Months

The COMPUTE commands convert the dates from year-month-day to year-month format; then YM calculates the difference between the values in the HIRE_DATE/100 and DAT_INC/100 fields:

```
TABLE FILE EMPLOYEE
PRINT DAT_INC AS 'RAISE DATE' AND COMPUTE
HIRE_MONTH/I4YM = HIRE_DATE/100; NOPRINT AND COMPUTE
MONTH_INC/I4YM = DAT_INC/100; NOPRINT AND COMPUTE
MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
BY LAST_NAME BY FIRST_NAME BY HIRE_DATE
IF MONTHS_HIRED NE 0
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME     FIRST_NAME  HIRE_DATE  RAISE DATE  MONTHS_HIRED
---------     ----------  ---------  ----------  ------------
CROSS         BARBARA      81/11/02    82/04/09             5
GREENSPAN     MARY         82/04/01    82/06/11             2
JONES         DIANE        82/05/01    82/06/01             1
MCCOY         JOHN         81/07/01    82/01/01             6
SMITH         MARY         81/07/01    82/01/01             6
```

Date-Time Functions

Date-Time functions are for use with timestamps in date-time formats, also known as H formats. A timestamp value refers to internally stored data capable of holding both date and time components with an accuracy of up to a nanosecond.

Using Date-Time Functions

The functions described in this section operate on fields in date-time format (sometimes called H format).

Date-Time Parameters

The DATEFORMAT parameter specifies the order of the date components for certain types of date-time values. The WEEKFIRST parameter specifies the first day of the week. The DTSTRICT parameter determines the extent to which date-time values are checked for validity.

Specifying the Order of Date Components

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

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

```
SET DATEFORMAT = option
```

where:

option

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

Using the DATEFORMAT Parameter

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

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

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

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

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

The WEEKFIRST parameter specifies a day of the week as the start of the week. This is used in week computations by the HADD, HDIFF, HNAME, HPART, and HYYWD functions. It is also used by the DTADD, DTDIFF, DTRUNC, and DTPART functions. The default values are different for these functions, as described in Using Date-Time Functions. The WEEKFIRST parameter does not change the day of the month that corresponds to each day of the week, but only specifies which day is considered the start of the week.

The HPART, DTPART, HYYWD, and HNAME subroutines can extract a week number from a date-time value. To determine a week number, they can use different definitions. For example, ISO 8601 standard week numbering defines the first week of the year as the first week in January with four or more days. Any preceding days in January belong to week 52 or 53 of the preceding year. The ISO standard also establishes Monday as the first day of the week.

You specify which type of week numbering to use by setting the WEEKFIRST parameter, as described in Using Date-Time Functions.

Since the week number returned by HNAME, DTPART, and HPART functions can be in the current year or the year preceding or following, the week number by itself may not be useful. The function HYYWD returns both the year and the week for a given date-time value.

Set a Day as the Start of the Week

```
SET WEEKFIRST = value
```

where:

value

Can be:

- **1 through 7**, representing Sunday through Saturday with non-standard week numbering.

  Week numbering using these values establishes the first week in January with seven days as week number 1. Preceding days in January belong to the last week of the previous year. All weeks have seven days.

- **ISO1 through ISO7**, representing Sunday through Saturday with ISO standard week numbering.

> **Note:** ISO is a synonym for ISO2.

Week numbering using these values establishes the first week in January with at least four days as week number 1. Preceding days in January belong to the last week of the previous year. All weeks have seven days.

- **STD1 through STD7**, in which the digit 1 (Sunday) through 7 (Saturday) indicates the starting day of the week.

  > **Note:** STD without a digit is equivalent to STD1.

  Week numbering using these values is as follows. Week number 1 begins on January 1 and ends on the day preceding the first day of the week. For example, for STD1, the first week ends on the first Saturday of the year. The first and last week may have fewer than seven days.

- **SIMPLE**, which establishes January 1 as the start of week 1, January 8 is the start of week 2, and so on. The first day of the week is, thus, the same as the first day of the year. The last week (week 53) is either one or two days long.
- **0 (zero)**, is the value of the WEEKFIRST setting before the user issues an explicit WEEKFIRST setting. The date-time functions HPART, HNAME, HYYWD, HADD, and HDIFF use Saturday as the start of the week, when the WEEKFIRST setting is 0. The simplified functions DTADD, DTDIFF, DTRUNC, and DTPART, as well as printing of dates truncated to weeks, and recognition of date constant strings that contain week numbers, use Sunday as the default value, when the WEEKFIRST setting is 0. If the user explicitly sets WEEKFIRST to another value, that value is used by all of the functions.

Setting Sunday as the Start of the Week

The following designates Sunday as the start of the week, using non-standard week numbering:

```
SET WEEKFIRST = 1
```

View the Current Setting of WEEKFIRST

```
? SET WEEKFIRST
```

This returns the value that indicates the week numbering algorithm and the first day of the week. For example, the integer 1 represents Sunday with non-standard week numbering.

Controlling Processing of Date-Time Values

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

Enable Strict Processing of Date-Time Values

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

where:

ON

Invokes strict processing. ON is the default value.

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

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

OFF

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

Supplying Arguments for Date-Time Functions

Date-time functions may operate on a component of a date-time value. This topic lists the valid component names and abbreviations for use with these functions.

Arguments for Use With Date and Time Functions

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

Component Name	Abbreviation	Valid Values
`year`	`yy`	0001-9999
`quarter`	`qq`	1-4
`month`	`mm`	1-12 or a month name, depending on the function.
`day-of-year`	`dy`	1-366
`day` or `day-of-month`	`dd`	1-31 (The two component names are equivalent.)
`week`	`wk`	1-53
`weekday`	`dw`	1-7 (Sunday-Saturday)

Component Name	Abbreviation	Valid Values
`hour`	`hh`	0-23
`minute`	`mi`	0-59
`second`	`ss`	0-59
`millisecond`	`ms`	0-999
`microsecond`	`mc`	0-999999
`nanosecond`	`ns`	0-999999999

Note:

- For an argument that specifies a length of eight, ten, or 12 characters, use eight to include milliseconds, ten to include microseconds, and 12 to include nanoseconds in the returned value.
- The last argument is always a USAGE format that indicates the data type returned by the function. The type may be A (alphanumeric), I (integer), D (floating-point double precision), H (date-time), or a date format (for example, YYMD).

Using Date-Time Formats

There are three types of date formats that are valid in date-time values: numeric string format, formatted-string format, and translated-string format. In each format, two-digit years are interpreted using the DEFCENT and YRTHRESH parameters.

Time components are separated by colons and may be followed by A.M., P.M., a.m., or p.m.

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

Numeric String Format

The numeric string format is exactly two, four, six, or eight digits. Four-digit strings are considered to be a year (century must be specified), and the month and day are set to January 1. Six and eight-digit strings contain two or four digits for the year, followed by two for the month, and two for the day. Because the component order is fixed with this format, the DATEFORMAT setting is ignored.

If a numeric-string format longer than eight digits is encountered, it is treated as a combined date-time string in the H*nn* format.

Using Numeric String Format

The following are examples of numeric string date constants:

String	Date
`99`	January 1, 1999
`1999`	January 1, 1999
`19990201`	February 1, 1999

Formatted-string Format

The formatted-string format contains a one or two-digit day, a one or two-digit month, and a two or four-digit year, each component separated by a space, slash, hyphen, or period.

All three components must be present and follow the DATEFORMAT setting. If any of the three fields is four digits, it is interpreted as the year, and the other two fields must follow the order given by the DATEFORMAT setting.

Using Formatted-string Format

The following are examples of formatted-string date constants and specify May 20, 1999:

```
1999/05/20
5 20 1999
99.05.20
1999-05-20
```

Translated-string Format

The translated-string format contains the full or abbreviated month name. The year must also be present in four-digit or two-digit form. If the day is missing, day 1 of the month is assumed; if present, it can have one or two digits. If the string contains both a two-digit year and a two-digit day, they must be in the order given by the DATEFORMAT setting.

Using Translated-string Format

The following date is in translated-string format:

```
January 6 2000
```

Time Format

Time components are separated by colons and may be followed by A.M., P.M., a.m., or p.m.

Seconds can be expressed with a decimal point or be followed by a colon. If there is a colon after seconds, the value following it represents milliseconds. There is no way to express microseconds or nanoseconds using this notation.

A decimal point in the seconds value indicates the decimal fraction of a second. Microseconds can be represented using six decimal digits. Nanoseconds can be represented using nine decimal digits.

Using Time Formats

The following are examples of acceptable time formats:

```
14:30:20:99       (99 milliseconds)
14:30
14:30:20.99       (99/100 seconds)
14:30:20.999999   (999999 microseconds)
02:30:20:500pm
```

Using Universal Date-Time Input Values

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

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

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

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

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

Assigning Date-Time Values

A date-time value is a constant in character format assigned by one of the following:

- A sequential data source.
- An expression that defines WHERE or IF criteria or creates a temporary field using the DEFINE or COMPUTE command.

A date-time constant can have blanks at the beginning or end or immediately preceding an am/pm indicator.

Assign Date-Time Values

In a character file

```
date_string [time_string]
```

or

```
time_string [date_string]
```

In a COMPUTE, DEFINE, or WHERE expression

```
DT(date_string [time_string])
```

or

```
DT(time_string [date_string])
```

In an IF expression

```
'date_string [time_string]'
```

or

```
'time_string [date_string]'
```

where:

time_string

Is a time string in an acceptable format. A time string can have a blank immediately preceding an am/pm indicator.

date_string

Is a date string in numeric string, formatted-string, or translated-string format.

In an IF criteria, if the value does not contain blanks or special characters, the single quotation marks are not necessary.

> **Note:** The date and time strings must be separated by at least one blank space. Blank spaces are also permitted at the beginning and end of the date-time string.

Assigning Date-Time Literals

The DT prefix can be used in a COMPUTE, DEFINE, or WHERE expression to assign a date-time literal to a date-time field. For example:

```
DT2/HYYMDS = DT(20051226 05:45);
DT3/HYYMDS = DT(2005 DEC 26 05:45);
DT4/HYYMDS = DT(December 26 2005 05:45);
```

Assigning a Date-Time Value in a COMPUTE Command

The following uses the DT function in a COMPUTE command to create a new field containing an assigned date-time value.

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME AND COMPUTE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
WHERE CURR_JOBCODE LIKE 'B%'
END
```

The output is:

```
LAST_NAME     FIRST_NAME           NEWSAL  RAISETIME
---------     ----------           ------  ---------
SMITH         MARY             $14,520.00  2000/01/01  9:00AM
JONES         DIANE            $20,328.00  2000/01/01  9:00AM
ROMANS        ANTHONY          $23,232.00  2000/01/01  9:00AM
MCCOY         JOHN             $20,328.00  2000/01/01  9:00AM
BLACKWOOD     ROSEMARIE        $23,958.00  2000/01/01  9:00AM
MCKNIGHT      ROGER            $17,710.00  2000/01/01  9:00AM
```

Assigning a Date-Time Value in WHERE Criteria

The following uses the DT function to create a new field containing an assigned date-time value. This value is then used as a WHERE criteria.

```
DEFINE FILE EMPLOYEE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
END
```

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME NEWSAL RAISETIME
WHERE RAISETIME EQ DT(20000101 09:00AM)
END
```

The output is:

```
LAST_NAME     FIRST_NAME           NEWSAL  RAISETIME
---------     ----------           ------  ---------
STEVENS       ALFRED           $12,100.00  2000/01/01  9:00AM
SMITH         MARY             $14,520.00  2000/01/01  9:00AM
JONES         DIANE            $20,328.00  2000/01/01  9:00AM
```

```
SMITH          RICHARD          $10,450.00  2000/01/01  9:00AM
BANNING        JOHN             $32,670.00  2000/01/01  9:00AM
IRVING         JOAN             $29,548.20  2000/01/01  9:00AM
ROMANS         ANTHONY          $23,232.00  2000/01/01  9:00AM
MCCOY          JOHN             $20,328.00  2000/01/01  9:00AM
BLACKWOOD      ROSEMARIE        $23,958.00  2000/01/01  9:00AM
MCKNIGHT       ROGER            $17,710.00  2000/01/01  9:00AM
GREENSPAN      MARY              $9,900.00  2000/01/01  9:00AM
CROSS          BARBARA          $29,768.20  2000/01/01  9:00AM
```

Assigning a Date-Time Value in IF Criteria

The following uses the DT function to create a new field containing an assigned date-time value. This value is then used in the IF phrase.

```
DEFINE FILE EMPLOYEE
NEWSAL/D12.2M = CURR_SAL + (0.1 * CURR_SAL);
RAISETIME/HYYMDIA = DT(20000101 09:00AM);
END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME NEWSAL RAISETIME
IF RAISETIME EQ '20000101 09:00AM'
END
```

The output is:

```
LAST_NAME      FIRST_NAME           NEWSAL  RAISETIME
---------      ----------           ------  ---------
STEVENS        ALFRED           $12,100.00  2000/01/01  9:00AM
SMITH          MARY             $14,520.00  2000/01/01  9:00AM
JONES          DIANE            $20,328.00  2000/01/01  9:00AM
SMITH          RICHARD          $10,450.00  2000/01/01  9:00AM
BANNING        JOHN             $32,670.00  2000/01/01  9:00AM
IRVING         JOAN             $29,548.20  2000/01/01  9:00AM
ROMANS         ANTHONY          $23,232.00  2000/01/01  9:00AM
MCCOY          JOHN             $20,328.00  2000/01/01  9:00AM
BLACKWOOD      ROSEMARIE        $23,958.00  2000/01/01  9:00AM
MCKNIGHT       ROGER            $17,710.00  2000/01/01  9:00AM
GREENSPAN      MARY              $9,900.00  2000/01/01  9:00AM
CROSS          BARBARA          $29,768.20  2000/01/01  9:00AM
```

HADD: Incrementing a Date-Time Value

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

Increment a Date-Time Value

```
HADD(datetime, 'component', increment, length, output)
```

where:

datetime

Date-time

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

component

Alphanumeric

Is the name of the component to be incremented enclosed in single quotation marks.

> **Note:** WEEKDAY is not a valid component for HADD.

increment

Integer

Is the number of units (positive or negative) by which to increment the component, the name of a numeric field that contains the value, or an expression that returns the value.

length

Integer

Is the number of characters returned. Valid values are:

- **8** indicates a date-time value that includes one to three decimal digits (milliseconds).
- **10** indicates a date-time value that includes four to six decimal digits

(microseconds).

- **12** indicates a date-time value that includes seven to nine decimal digits (nanoseconds).

output

Date-time

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in date-time format (data type H).

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

HADD adds two months to each value in TRANSDATE and stores the result in ADD_MONTH. If necessary, the day is adjusted so that it is valid for the resulting month.

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

The output is:

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

Converting Unix (Epoch) Time to a Date-Time Value

Unix time (also known as Epoch time) defines an instant in time as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, not counting leap seconds.

The following DEFINE FUNCTION takes a number representing epoch time and converts it to a date-time value by using the HADD function to add the number of seconds represented by the input value in epoch time to the epoch base date:

```
DEFINE FUNCTION UNIX2GMT(INPUT/I9)
   UNIX2GMT/HYYMDS = HADD(DT(1970 JAN 1),'SECONDS',INPUT,8,'HYYMDS');
END
```

The following request uses this DEFINE FUNCTION to convert the epoch time 1449068652 to a date-time value:

```
DEFINE FILE GGSALES
INPUT/I9=1449068652;
OUTDATE/HMTDYYSb = UNIX2GMT(INPUT);
END
TABLE FILE GGSALES
PRINT DATE NOPRINT INPUT OUTDATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
END
```

The output is shown in the following image:

INPUT	OUTDATE
1449068652	December 02 2015 3:04:12 pm

HCNVRT: Converting a Date-Time Value to Alphanumeric Format

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

Convert a Date-Time Value to Alphanumeric Format

```
HCNVRT(datetime, '(format)', length, output)
```

where:

datetime

Date-time

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

format

Alphanumeric

Is the format of the date-time field enclosed in parentheses and single quotation marks. It must be a date-time format (data type H, up to H23).

length

Integer

Is the number of characters in the alphanumeric field that is returned. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value. If the *length* is smaller than the number of characters needed to display the alphanumeric field, the function returns a blank.

output

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in alphanumeric format and must be long enough to contain all of the characters returned.

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

HCNVRT converts the TRANSDATE field to alphanumeric format. The first function does not include date-time display options for the field; the second function does for readability. It also specifies the display of seconds in the input field.

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

The output is:

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

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

The HDATE function converts the date portion of a date-time value to the date format YYMD. You can then convert the result to other date formats.

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

```
HDATE(datetime, output)
```

where:

datetime

Date-time

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

output

Date

Is the format in single quotation marks or the field that contains the result.

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

HDATE converts the date portion of the TRANSDATE field to the date format YYMD:

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

The output is:

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

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

The HDIFF function calculates the number of date or time component units between two date-time values.

Usage Notes for HDIFF

HDIFF does its subtraction differently from DATEDIF, which subtracts date components stored in date fields. The DATEDIF calculation looks for full years or full months. Therefore, subtracting the following two dates and requesting the number of months or years, results in 0:

```
DATE1 12/25/2014, DATE2 1/5/2015
```

Performing the same calculation using HDIFF on date-time fields results in a value of 1 month or 1 year as, in this case, the month or year is first extracted from each date-time value, and then the subtraction occurs.

Find the Number of Units Between Two Date-Time Values

```
HDIFF(end_dt, start_dt, 'component', output)
```

where:

end_dt

Date-time

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

start_dt

Date-time

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

component

Alphanumeric

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

output

Floating-point double-precision

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be floating-point double-precision.

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

HDIFF calculates the number of days between the TRANSDATE and ADD_MONTH fields and stores the result in DIFF_PAYS, which has the format D12.2:

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

The output is:

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

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

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

Convert a Date Value to a Date-Time Value

```
HDTTM(date, length, output)
```

where:

date

Date

Is the date to be converted, the name of a date field that contains the value, or an expression that returns the value. It must be a full component format date. For example, it can be MDYY or YYJUL.

length

Integer

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

- **8** indicates a time value that includes milliseconds.
- **10** indicates a time value that includes microseconds.
- **12** indicates a time value that includes nanoseconds.

output

Date-time

Is the generated date-time value. It can be a field or the format of the output value enclosed in single quotation marks. The value must have a date-time format (data type H).

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

HDTTM converts the date field TRANSDATE_DATE to a date-time field:

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

```
DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
WHERE DATE EQ 2000;
END
```

The output is:

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

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

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

Extract Multiple Components From a Date-Time Value

```
HEXTR(datetime, 'componentstring', length, output)
```

where:

datetime

Date-time

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

componentstring

Alphanumeric

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

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

length

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

- **8** indicates a time value that includes milliseconds.
- **10** indicates a time value that includes microseconds.
- **12** indicates a time value that includes nanoseconds.

output

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in date-time format (data type H).

Extracting Hour and Minute Components Using HEXTR

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

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

The output is:

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

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

The HGETC function returns the current local date and time in the desired date-time format. If millisecond or microsecond values are not available in your operating environment, the function retrieves the value zero for these components.

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

```
HGETC(length, output)
```

where:

length

Integer

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

- **8** indicates a time value that includes milliseconds.
- **10** indicates a time value that includes microseconds.
- **12** indicates a time value that includes nanoseconds.

output

Date-time

Is the returned date-time value. Can be a field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

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

HGETC stores the current date and time in DT2:

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

The output is:

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

HGETZ: Storing the Current Coordinated Universal Time in a Date-Time Field

HGETZ provides the current Coordinated Universal Time (UTC/GMT time, often called Zulu time). UTC is the primary civil time standard by which the world regulates clocks and time.

The value is returned in the desired date-time format. If millisecond or microsecond values are not available in your operating environment, the function retrieves the value zero for these components.

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

```
HGETZ(length, output)
```

where:

length

Integer

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

- **8** indicates a time value that includes milliseconds.
- **10** indicates a time value that includes microseconds.
- **12** indicates a time value that includes nanoseconds.

output

Date-time

Is the returned date-time value. Can be a field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

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

HGETZ stores the current universal date and time in DT2:

```
TABLE FILE VIDEOTRK
PRINT CUSTID AND COMPUTE
DT2/HYYMDm = HGETZ(10, 'HYYMDm');
WHERE CUSTID GE '2000' AND CUSTID LE '3000';
END
```

The output is:

```
  CUSTID  DT2
  ------  ---
  2165    2015/05/08 14:43:08.740000
  2187    2015/05/08 14:43:08.740000
  2280    2015/05/08 14:43:08.740000
  2282    2015/05/08 14:43:08.740000
  2884    2015/05/08 14:43:08.740000
```

Calculating the Time Zone

The time zone can be calculated as a positive or negative hourly offset from GMT. Locations to the west of the prime meridian have a negative offset. The following request uses the HGETC function to retrieve the local time, and the HGETZ function to retrieve the GMT time. The HDIFF function calculates the number of boundaries between them in minutes. The zone is found by dividing the minutes by 60:

```
DEFINE FILE EMPLOYEE
LOCALTIME/HYYMDS = HGETC(8, LOCALTIME);
UTCTIME/HYYMDS = HGETZ(8, UTCTIME);
MINUTES/D4= HDIFF(LOCALTIME, UTCTIME, 'MINUTES', 'D4');
```

```
ZONE/P3 = MINUTES/60;
END
TABLE FILE EMPLOYEE
PRINT EMP_ID NOPRINT OVER
LOCALTIME  OVER
UTCTIME OVER
MINUTES OVER
ZONE
IF RECORDLIMIT IS 1
END
```

The output is:

```
  LOCALTIME  2015/05/12 12:47:04
  UTCTIME    2015/05/12 16:47:04
  MINUTES                   -240
  ZONE                        -4
```

HHMMSS: Retrieving the Current Time

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

A compiled MODIFY procedure must use HHMMSS to obtain the time; it cannot use the &TOD variable, which also returns the time. The &TOD variable is made current only when you execute a MODIFY, SCAN, or FSCAN procedure.

Retrieve the Current Time

```
HHMMSS(output)
```

where:

output

Alphanumeric, at least A8

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

Retrieving the Current Time

HHMMSS retrieves the current time and displays it in the page footing:

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

The output is:

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

SALARY REPORT RUN AT TIME 15.21.14
```

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

The HHMS function converts a date-time value to a time value.

Convert a Date-Time Value to a Time Value

```
HHMS(datetime, length, output)
```

where:

datetime

Date-time

Is the date-time value to be converted.

length

Numeric

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

- **8** indicates a time value that includes milliseconds.
- **10** indicates a time value that includes microseconds.
- **12** indicates a time value that includes nanoseconds.

output

Time

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

Converting a Date-Time Value to a Time value

The following example converts the date-time field TRANSDATE to a time field with the time format HHIS,

```
DEFINE FILE VIDEOTR2
TRANSYEAR/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
END
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
TRANS_TIME/HHIS = HHMS(TRANSDATE, 8, 'HHIS');
WHERE TRANSYEAR EQ 2000;
END
```

The output is:

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

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

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

Convert an Alphanumeric String to a Date-Time Value

```
HINPUT(source_length, 'source_string', output_length, output)
```

where:

source_length

Integer

Is the number of characters in the source string to be converted. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value.

source_string

Alphanumeric

Is the string to be converted enclosed in single quotation marks, the name of an alphanumeric field that contains the string, or an expression that returns the string. The string can consist of any valid date-time input value.

output_length

Integer

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

- **8** indicates a time value that includes one to three decimal digits (milliseconds).
- **10** indicates a time value that includes four to six decimal digits (microseconds).
- **12** indicates a time value that includes seven to nine decimal digits (nanoseconds).

output

Date-time

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

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

HCNVRT converts the TRANSDATE field to alphanumeric format, then HINPUT converts the alphanumeric string to a date-time value:

```
TABLE FILE VIDEOTR2
PRINT CUSTID TRANSDATE AS 'DATE-TIME' AND COMPUTE
ALPHA_DATE_TIME/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20');
DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
WHERE DATE EQ 2000;
END
```

The output is:

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

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

The HMIDNT function changes the time portion of a date-time value to midnight (all zeros by default). This allows you to compare a date field with a date-time field.

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

```
HMIDNT(datetime, length, output)
```

where:

datetime

Date-time

Is the date-time value whose time is to be set to midnight, the name of a date-time field that contains the value, or an expression that returns the value.

length

Integer

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

- **8** indicates a time value that includes milliseconds.
- **10** indicates a time value that includes microseconds.
- **12** indicates a time value that includes nanoseconds.

output

Date-time

Is the date-time return value whose time is set to midnight and whose date is copied from timestamp. Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

Setting the Time to Midnight (Reporting)

HMIDNT sets the time portion of the TRANSDATE field to midnight first in the 24-hour system and then in the 12-hour system:

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

The output is:

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

HMASK: Extracting Date-Time Components and Preserving Remaining Components

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

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

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

where:

source

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

componentstring

Is a string of codes, in any order, that indicates which components are to be extracted and moved to the output date-time field. The following table shows the valid values.

The string is considered to be terminated by any character not in this list:

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

input

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

length

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

- **8** indicates a time value that includes one to three decimal digits (milliseconds).
- **10** indicates a time value that includes four to six decimal digits (microseconds).

- **12** indicates a time value that includes seven to nine decimal digits (nanoseconds).

output

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. This field must be in date-time format (data type H).

Usage Notes for the HMASK Function

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

Changing a Date-Time Field Using HMASK

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

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

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

The output is

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

HNAME: Retrieving a Date-Time Component in Alphanumeric Format

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

Retrieve a Date-Time Component in Alphanumeric Format

```
HNAME(datetime, 'component', output)
```

where:

datetime

Date-time

Is the date-time value from which a component value is to be extracted, the name of a date-time field containing the value that contains the value, or an expression that returns the value.

component

Alphanumeric

Is the name of the component to be retrieved enclosed in single quotation marks.

output

Alphanumeric, at least A2

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

The function converts a month argument to an abbreviation of the month name and converts and all other components to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

Retrieving the Week Component in Alphanumeric Format (Reporting)

HNAME returns the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST parameter setting changes the value of the component.

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

When WEEKFIRST is set to seven, the output is:

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

When WEEKFIRST is set to three, the output is:

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

For details on WEEKFIRST, see the *ibi™ FOCUS® Developing Applications* manual.

Retrieving the Day Component in Alphanumeric Format (Reporting)

HNAME retrieves the day in alphanumeric format from the TRANSDATE field:

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

The output is:

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

HPART: Retrieving a Date-Time Component as a Numeric Value

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

Retrieve a Date-Time Component in Numeric Format

```
HPART(datetime, 'component', output)
```

where:

datetime

Date-time

Is the date-time value from which the component is to be extracted, the name of a date-time field that contains the value, or an expression that returns the value.

component

Alphanumeric

Is the name of the component to be retrieved enclosed in single quotation marks.

output

Integer

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

Retrieving the Day Component in Numeric Format (Reporting)

HPART retrieves the day in integer format from the TRANSDATE field:

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

The output is:

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

HSETPT: Inserting a Component Into a Date-Time Value

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

Insert a Component Into a Date-Time Value

```
HSETPT(datetime, 'component', value, length, output)
```

where:

datetime

Date-time

Is the date-time value in which to insert the component, the name of a date-time field that contains the value, or an expression that returns the value.

component

Alphanumeric

Is the name of the component to be inserted enclosed in single quotation marks.

value

Integer

Is the numeric value to be inserted for the requested component, the name of a numeric field that contains the value, or an expression that returns the value.

length

Integer

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

- **8** indicates a time value that includes one to three decimal digits (milliseconds).
- **10** indicates a time value that includes four to six decimal digits (microseconds).

- **12** indicates a time value that includes seven to nine decimal digits (nanoseconds).

output

Date-time

Is the returned date-time value whose chosen component is updated. All other components are copied from the source date-time value.

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be in date-time format (data type H).

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

HSETPT inserts the day as 28 into the ADD_MONTH field and stores the result in INSERT_ DAY:

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

The output is:

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

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

The HTIME function converts the time portion of a date-time value to the number of milliseconds if the length argument is eight, microseconds if the length argument is ten, or nanoseconds if the length argument is 12.

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

```
HTIME(length, datetime, output)
```

where:

length

Integer

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

- **8** indicates a time value that includes one to three decimal digits (milliseconds).
- **10** indicates a time value that includes four to six decimal digits (microseconds).
- **12** indicates a time value that includes seven to nine decimal digits (nanoseconds).

datetime

Date-time

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

output

Floating-point double-precision

Is the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be floating-point double-precision.

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

HTIME converts the time portion of the TRANSDATE field to the number of milliseconds:

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

The output is:

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

HTMTOTS or TIMETOTS: Converting a Time to a Timestamp

The HTMTOTS function returns a timestamp using the current date to supply the date components of its value, and copies the time components from its input date-time value.

Note: TIMETOTS is a synonym for HTMTOTS.

Convert a Time to a Timestamp

```
HTMTOTS(time, length, output)
```

or

```
TIMETOTS(time, length, output)
```

where:

time

Date-Time

Is the date-time value whose time will be used. The date portion will be ignored.

length

Integer

Is the length of the result. This can be one of the following:

- **8** for input time values including milliseconds.
- **10** for input time values including microseconds.
- **12** for input time values including nanoseconds.

output_format

Date-Time

Is the timestamp whose date is set to the current date, and whose time is copied from time.

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Converting a Time to a Timestamp

HTMTOTS converts the time portion of the TRANSDATE field to a timestamp, using the current date for the date portion of the returned value:

```
DEFINE FILE VIDEOTR2
  TSTMPSEC/HYYMDS = HTMTOTS(TRANSDATE, 8, 'HYYMDS');
END
TABLE FILE VIDEOTR2
PRINT TRANSDATE TSTMPSEC
BY LASTNAME BY FIRSTNAME
WHERE DATE EQ '1991'
END
```

The output is:

```
LASTNAME         FIRSTNAME   TRANSDATE         TSTMPSEC
--------         ---------   ---------         --------
CRUZ             IVY         1991/06/27 02:45  2011/01/11 02:45:00
GOODMAN          JOHN        1991/06/25 01:19  2011/01/11 01:19:00
GREEVEN          GEORGIA     1991/06/24 10:27  2011/01/11 10:27:00
HANDLER          EVAN        1991/06/20 05:15  2011/01/11 05:15:00
                             1991/06/21 07:11  2011/01/11 07:11:00
KRAMER           CHERYL      1991/06/21 01:10  2011/01/11 01:10:00
                             1991/06/19 07:18  2011/01/11 07:18:00
                             1991/06/19 04:11  2011/01/11 04:11:00
MONROE           CATHERINE   1991/06/25 01:17  2011/01/11 01:17:00
                 PATRICK     1991/06/27 01:17  2011/01/11 01:17:00
SPIVEY           TOM         1991/11/17 11:28  2011/01/11 11:28:00
WILLIAMS         KENNETH     1991/06/24 04:43  2011/01/11 04:43:00
                             1991/06/24 02:08  2011/01/11 02:08:00
```

HYYWD: Returning the Year and Week Number From a Date-Time Value

The week number returned by HNAME and HPART can actually be in the year preceding or following the input date.

The HYYWD function returns both the year and the week number from a given date-time value.

The output is edited to conform to the ISO standard format for dates with week numbers, yyyy-Www-d.

Return the Year and Week Number From a Date-Time Value

```
HYYWD(dtvalue, output)
```

where:

dtvalue

Date-time

Is the date-time value to be edited, the name of a date-time field that contains the value, or an expression that returns the value.

output

Alphanumeric

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

The output format must be at least 10 characters long. The output is in the following format:

```
yyyy-Www-d
```

where:

yyyy

Is the four-digit year.

ww

Is the two-digit week number (01 to 53).

d

Is the single-digit day of the week (1 to 7). The d value is relative to the current WEEKFIRST setting. If WEEKFIRST is 2 or ISO2 (Monday), then Monday is represented in the output as 1, Tuesday as 2.

Using the EDIT function, you can extract the individual subfields from this output.

Returning the Year and Week Number From a Date-Time Value

The following request against the VIDEOTR2 data source calls HYYWD to convert the TRANSDATE date-time field to the ISO standard format for dates with week numbers. WEEKFIRST is set to ISO2, which produces ISO standard week numbering:

```
SET WEEKFIRST = ISO2
TABLE FILE VIDEOTR2
SUM TRANSTOT QUANTITY
COMPUTE ISODATE/A10 = HYYWD(TRANSDATE, 'A10');
BY TRANSDATE
WHERE QUANTITY GT 1
END
```

The output is:

```
TRANSDATE          TRANSTOT  QUANTITY  ISODATE
---------          --------  --------  -------
1991/06/24 04:43      16.00         2  1991-W26-1
1991/06/25 01:17       2.50         2  1991-W26-2
1991/06/27 02:45      16.00         2  1991-W26-4
1996/08/17 05:11       5.18         2  1996-W33-6
1998/02/04 04:11      12.00         2  1998-W06-3
1999/01/30 04:16      13.00         2  1999-W04-6
1999/04/22 06:19       3.75         3  1999-W16-4
1999/05/06 05:14       1.00         2  1999-W18-4
1999/08/09 03:17      15.00         2  1999-W32-1
1999/09/09 09:18      14.00         2  1999-W36-4
1999/10/16 09:11       5.18         2  1999-W41-6
1999/11/05 11:12       2.50         2  1999-W44-5
1999/12/09 09:47       5.18         2  1999-W49-4
1999/12/15 04:04       2.50         2  1999-W50-3
```

Extracting a Component From a Date Returned by HYYWD

The following request against the VIDEOTR2 data source calls HYYWD to convert the TRANSDATE date-time field to the ISO standard format for dates with week numbers. It then uses the EDIT function to extract the week component from this date. WEEKFIRST is set to ISO2, which produces ISO standard week numbering:

```
SET WEEKFIRST = ISO2
TABLE FILE VIDEOTR2
SUM TRANSTOT QUANTITY
COMPUTE ISODATE/A10 = HYYWD(TRANSDATE, 'A10');
COMPUTE WEEK/A2 = EDIT(ISODATE, '$$$$$$99$$');
```

```
BY TRANSDATE
WHERE QUANTITY GT 1 AND DATE EQ 1991
END
```

The output is:

```
TRANSDATE          TRANSTOT  QUANTITY  ISODATE     WEEK
---------          --------  --------  -------     ----
1991/06/24 04:43      16.00         2  1991-W26-1  26
1991/06/25 01:17       2.50         2  1991-W26-2  26
1991/06/27 02:45      16.00         2  1991-W26-4  26
```

Simplified Conversion Functions

Simplified conversion functions have streamlined parameter lists, similar to those used by SQL functions. In some cases, these simplified functions provide slightly different functionality than previous versions of similar functions.

The simplified functions do not have an output argument. Each function returns a value that has a specific data type.

When used in a request against a relational data source, these functions are optimized (passed to the RDBMS for processing).

CHAR: Returning a Character Based on a Numeric Code

The CHAR function accepts a decimal integer and returns the character identified by that number converted to ASCII or EBCDIC, depending on the operating environment. The output is returned as variable length alphanumeric. If the number is above the range of valid characters, a null value is returned.

For a chart of printable characters and their decimal equivalents, see Character Chart for ASCII and EBCDIC.

Return a Character Based on a Numeric Code

```
CHAR(number_code)
```

where:

number_code

Integer

Is a field, number, or numeric expression whose whole absolute value will be used as a number code to retrieve an output character.

For example, a TAB character is returned by CHAR(9) in ASCII environments, or by CHAR (5) in EBCDIC environments.

Using the CHAR Function to Insert Control Characters Into a String

The following request defines a field with carriage return (CHAR(13)) and line feed (CHAR (10)) characters inserted between the words HELLO and GOODBYE (in an ASCII environment). To show that these characters were inserted, the output is generated in PDF format and the StyleSheet attribute LINEBREAK='CRLF' is used to have these characters respected and print the field value on two lines.

```
DEFINE FILE WFLITE
MYFIELD/A20 WITH COUNTRY_NAME='HELLO' | CHAR(13) | CHAR(10) | 'GOODBYE';
END
TABLE FILE WFLITE
SUM MYFIELD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CRLF',$
ENDSTYLE
END
```

The output is shown in the following image.


MYFIELD

HELLO
GOODBYE

COMPACTFORMAT: Displaying Numbers in an Abbreviated Format

COMPACTFORMAT displays numbers in a compact format where:

- K is an abbreviation for thousands.
- M is an abbreviation for millions.
- B is an abbreviation for billions.
- T is an abbreviation for trillions.

COMPACTFORMAT computes which abbreviation to use, based on the order of magnitude of the largest value in the column. The returned value is an alphanumeric string. Attempting to output this value to a numeric format will result in a format error, and the value zero (0) will be displayed.

Display Numbers in an Abbreviated Format

```
COMPACTFORMAT(input)
```

where:

input

Is the name of a numeric field.

Displaying Numbers in an Abbreviated Format

The following example uses the COMPACTFORMAT function to abbreviate the display of the summed values of the DAYSDELAYED, QUANTITY_SOLD, and COGS_US fields.

```
TABLE FILE WFLITE
SUM DAYSDELAYED QUANTITY_SOLD COGS_US
COMPUTE
CDAYS/A30= COMPACTFORMAT(DAYSDELAYED);
CQUANT/A30= COMPACTFORMAT(QUANTITY_SOLD);
CCOGS/A30= COMPACTFORMAT(COGS_US);
ON TABLE SET PAGE NOPAGE
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Days Delayed	Quantity Sold	Cost of Goods	CDAYS	CQUANT	CCOGS
5,355	13,923	$2,950,358.00	5,355	14K	$3M

CTRLCHAR: Returning a Non-Printable Control Character

The CTRLCHAR function returns a nonprintable control character specific to the running operating environment, based on a supported list of keywords. The output is returned as a variable-length alphanumeric.

Return a Non-Printable Control Character

```
CTRLCHAR(ctrl_char)
```

where:

ctrl_char

Is one of the following keywords.

- **NUL** returns a null character.
- **SOH** returns a start of heading character.
- **STX** returns a start of text character.
- **ETX** returns an end of text character.
- **EOT** returns an end of transmission character.
- **ENQ** returns an enquiry character.
- **ACK** returns an acknowledge character.
- **BEL** returns a bell or beep character.
- **BS** returns a backspace character.

- **TAB** or **HT** returns a horizontal tab character.
- **LF** returns a line feed character.
- **VT** returns a vertical tab character.
- **FF** returns a form feed (top of page) character.
- **CR** returns a carriage control character.
- **SO** returns a shift out character.
- **SI** returns a shift in character.
- **DLE** returns a data link escape character.
- **DC1** or **XON** returns a device control 1 character.
- **DC2** returns a device control 2 character.
- **DC3** or **XOFF** returns a device control 3 character.
- **DC4** returns a device control 4 character.
- **NAK** returns a negative acknowledge character.
- **SYN** returns a synchronous idle character.
- **ETB** returns an end of transmission block character.
- **CAN** returns a cancel character.
- **EM** returns an end of medium character.
- **SUB** returns a substitute character.
- **ESC** returns an escape, prefix, or altmode character.
- **FS** returns a file separator character.
- **GS** returns a group separator character.
- **RS** returns a record separator character.
- **US** returns a unit separator character.
- **DEL** returns a delete, rubout, or interrupt character.

Using the CTRLCHAR Function to Insert Control Characters Into a String

The following request defines a field with carriage return (CTRLCHAR(CR)) and line feed (CTRLCHAR(LF)) characters inserted between the words HELLO and GOODBYE. To show that these characters were inserted, the output is generated in PDF format and the StyleSheet attribute LINEBREAK='CRLF' is used to have these characters respected and print the field value on two lines.

```
DEFINE FILE WFLITE
MYFIELD/A20 WITH COUNTRY_NAME='HELLO' | CTRLCHAR(CR) | CTRLCHAR(LF) |
'GOODBYE';
END
TABLE FILE WFLITE
SUM MYFIELD
ON TABLE PCHOLD FORMAT PDF
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
TYPE=REPORT,LINEBREAK='CRLF',$
ENDSTYLE
END
```

The output is shown in the following image.

MYFIELD
HELLO GOODBYE

FPRINT: Displaying a Value in a Specified Format

Given an output format, the simplified conversion function FPRINT converts a value to alphanumeric format for display.

Note: A legacy FPRINT function also exists and is still supported. For information, see FPRINT: Converting Fields to Alphanumeric Format. The legacy function has an additional argument for the name or format of the returned value.

Display a Value in a Specified Format

```
FPRINT(value, 'out_format')
```

where:

value

Any data type

Is the value to be converted.

'out_format'

Fixed length alphanumeric

Is the display format. For information about valid display formats, see the manual.

Displaying a Value in a Specified Format

The following request displays COGS_US as format 'D9M', and TIME_DATE as format 'YYMtrD', by converting them to alphanumeric using FPRINT.

```
DEFINE FILE WFLITE
COGS_A/A25 = FPRINT(COGS_US, 'D9M');
DATE1/A25 = FPRINT(TIME_DATE, 'YYMtrD');
END
TABLE FILE WFLITE
PRINT LST.COGS_US COGS_A  DATE1
BY TIME_DATE
WHERE RECORDLIMIT EQ 10
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Sale Date	LST Cost of Goods	COGS_A	DATE1
01/03/2009	$234.00	$234	2009, January 3
	$46.00	$46	2009, January 3
	$380.00	$380	2009, January 3
	$374.00	$374	2009, January 3
	$310.00	$310	2009, January 3
	$83.00	$83	2009, January 3
	$312.00	$312	2009, January 3
	$548.00	$548	2009, January 3
	$400.00	$400	2009, January 3
	$131.00	$131	2009, January 3

HEXTYPE: Returning the Hexadecimal View of an Input Value

The HEXTYPE function returns the hexadecimal view of an input value of any data type. The result is returned as an variable length alphanumeric. The alphanumeric field to which the hexidecimal value is returned must be large enough to hold two characters for each input character. The value returned depends on the running operating environment.

Returning the Hexadecimal View of an Input Value

```
HEXTYPE(in_value)
```

where:

in_value

Is an alphanumeric or integer field, constant, or expression.

Returning a Hexadecimal View

The following request returns a hexadecimal view of the country names and the sum of the days delayed.

```
DEFINE FILE WFLITE
Days/I8 = DAYSDELAYED;
Country/A20 = COUNTRY_NAME;
HexCountry/A30 = HEXTYPE(Country);
END
TABLE FILE WFLITE
SUM COUNTRY_NAME NOPRINT  Country HexCountry Days
COMPUTE HexDays/A40 = HEXTYPE(Days);
BY COUNTRY_NAME NOPRINT
WHERE COUNTRY_NAME LT 'P'
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.

Country	HexCountry	Days	HexDays
Argentina	417267656E74696E61202020202020	84	00000054
Australia	4175737472616C6961202020202020	27	0000001B
Austria	41757374726961202020202020202020	798	0000031E
Belgium	42656C6769756D2020202020202020	14	0000000E
Brazil	4272617A696C202020202020202020	204	000000CC
Canada	43616E616461202020202020202020	584	00000248
Chile	4368696C652020202020202020202020	45	0000002D
China	4368696E612020202020202020202020	1	00000001
Colombia	436F6C6F6D6269612020202020202020	114	00000072
Denmark	44656E6D61726B2020202020202020	0	00000000
Egypt	45677970742020202020202020202020	3	00000003
Finland	46696E6C616E64202020202020202020	3	00000003
France	4672616E636520202020202020202020	49	00000031
Germany	4765726D616E79202020202020202020	498	000001F2
Greece	47726565636520202020202020202020	9	00000009
Hungary	48756E67617279202020202020202020	7	00000007
India	496E6469612020202020202020202020	23	00000017
Ireland	4972656C616E64202020202020202020	7	00000007
Israel	49737261656C20202020202020202020	2	00000002
Italy	4974616C792020202020202020202020	7	00000007
Japan	4A6170616E2020202020202020202020	12	0000000C
Luxembourg	4C7578656D626F7572672020202020	0	00000000
Malaysia	4D616C61797369612020202020202020	20	00000014
Mexico	4D657869636F20202020202020202020	170	000000AA
Netherlands	4E65746865726C616E647320202020	8	00000008
Norway	4E6F7277617920202020202020202020	0	00000000

PHONETIC: Returning a Phonetic Key for a String

PHONETIC calculates a phonetic key for a string, or a null value on failure. Phonetic keys are useful for grouping alphanumeric values, such as names, that may have spelling variations. This is done by generating an index number that will be the same for the variations of the same name based on pronunciation. One of two phonetic algorithms can be used for indexing, Metaphone and Soundex. Metaphone is the default algorithm, except on z/OS where the default is Soundex.

You can set the algorithm to use with the following command.

```
SET PHONETIC_ALGORITHM = {METAPHONE|SOUNDEX}
```

Most phonetic algorithms were developed for use with the English language. Therefore, applying the rules to words in other languages may not give a meaningful result.

Metaphone is suitable for use with most English words, not just names. Metaphone algorithms are the basis for many popular spell checkers.

Note: Metaphone is not optimized in generated SQL. Therefore, if you need to optimize the request for an SQL DBMS, the SOUNDEX setting should be used.

Soundex is a legacy phonetic algorithm for indexing names by sound, as pronounced in English.

Return a Phonetic Key

```
PHONETIC(string)
```

where:

string

Alphanumeric

Is a string for which to create the key. A null value will be returned on failure.

Generating a Phonetic Key

The following request changes the spelling of the last name for MARY SMITH to SMYTHE and generates a phonetic key for each last name.

```
DEFINE FILE EMPLOYEE
LAST_NAME2/A16 = IF LAST_NAME EQ 'SMITH' AND FIRST_NAME EQ 'MARY' THEN
'SMYTHE' ELSE LAST_NAME;
PKEY/A10 = PHONETIC(LAST_NAME2);
```

```
END
TABLE FILE EMPLOYEE
PRINT FIRST_NAME LAST_NAME2
BY PKEY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. Note that the two spellings for SMITH are assigned the same index number.

PKEY	FIRST_NAME	LAST_NAME2
B423	ROSEMARIE	BLACKWOOD
B552	JOHN	BANNING
C620	BARBARA	CROSS
G652	MARY	GREENSPAN
I615	JOAN	IRVING
J520	DIANE	JONES
M200	JOHN	MCCOY
M252	ROGER	MCKNIGHT
R552	ANTHONY	ROMANS
S315	ALFRED	STEVENS
S530	MARY	SMYTHE
	RICHARD	SMITH

TO_INTEGER: Converting a Character String to an Integer Value

TO_INTEGER converts a character string that contains a valid number consisting of digits and an optional decimal point to an integer value. If the value contains a decimal point, the value after the decimal point is truncated. If the value does not represent a valid number, zero (0) is returned.

Convert a Character String to an Integer

```
TO_INTEGER(string)
```

where:

string

Is a character string enclosed in single quotation marks or a character field that represents a number containing digits and an optional decimal point.

Converting a Character String to an Integer Value

The following request converts character strings to integers. Digits following the decimal point are truncated.

```
DEFINE FILE WFLITE
INT1/I8 = TO_INTEGER('56.78');
INT2/I8 = TO_INTEGER('.5678');
INT3/I8 = TO_INTEGER('5678');
END
TABLE FILE WFLITE
PRINT INT1 INT2 INT3
BY BUSINESS_REGION AS Region
WHERE READLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Region	INT1	INT2	INT3
EMEA	56	0	5678

TO_NUMBER: Converting a Character String to a Numeric Value

TO_NUMBER converts a character string that contains a valid number consisting of digits and an optional decimal point to the numeric format most appropriate to the context. If the value does not represent a valid number, zero (0) is returned.

Convert a Character String to a Number

```
TO_NUMBER(string)
```

where:

string

Is a character string enclosed in single quotation marks or a character field that represents a number containing digits and an optional decimal point. This string will be converted to a double-precision floating point number.

Converting a Character String to a Number

The following request converts character strings to double-precision floating point numbers.

```
DEFINE FILE WFLITE
NUM1/D12.1 = TO_NUMBER('56.78');
NUM2/D12.2 = TO_NUMBER('0.5678');
END
TABLE FILE WFLITE
PRINT NUM1 NUM2
BY BUSINESS_REGION AS Region
WHERE READLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Region	NUM1	NUM2
EMEA	56.8	.57

Format Conversion Functions

Format conversion functions convert fields from one format to another.

For information on field formats see the *ibi™ FOCUS® Describing Data* manual.

For many functions, the *output* argument can be supplied either as a field name or as a format enclosed in single quotation marks. However, if a function is called from a Dialogue Manager command, this argument must always be supplied as a format.

ATODBL: Converting an Alphanumeric String to Double-Precision Format

The ATODBL function converts a number in alphanumeric format to decimal (double-precision) format.

Convert an Alphanumeric String to Double-Precision Format

```
ATODBL(source_string, length, output)
```

where:

source_string

Alphanumeric

Is the string consisting of digits and, optionally, one sign and one decimal point to be converted, or a field or variable that contains the string.

length

Alphanumeric

Is the two-character length of the source string in bytes. This can be a numeric constant, or a field or variable that contains the value. If you specify a numeric constant, enclose it in single quotation marks, for example '12'.

output

Double precision floating-point

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Converting an Alphanumeric Field to Double-Precision Format

ATODBL converts the EMP_ID field into double-precision format and stores the result in D_EMP_ID:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME AND
EMP_ID AND
COMPUTE D_EMP_ID/D12.2 = ATODBL(EMP_ID, '09', D_EMP_ID);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME FIRST_NAME EMP_ID    D_EMP_ID
--------- ---------- ------    --------
SMITH     MARY       112847612 112,847,612.00
JONES     DIANE      117593129 117,593,129.00
MCCOY     JOHN       219984371 219,984,371.00
BLACKWOOD ROSEMARIE  326179357 326,179,357.00
GREENSPAN MARY       543729165 543,729,165.00
CROSS     BARBARA    818692173 818,692,173.00
```

Converting an Alphanumeric Value to Double-Precision Format With MODIFY

In the following example, the Master File contains the MISSING attribute for the CURR_SAL field. If you do not enter a value for this field, it is interpreted as the default value, a period.

```
FILENAME=EMPLOYEE, SUFFIX=FOC
SEGNAME=EMPINFO,  SEGTYPE=S1
 FIELDNAME=EMP_ID,       ALIAS=EID,FORMAT=A9,                 $
    .
    .
    .
 FIELDNAME=CURR_SAL,     ALIAS=CSAL,FORMAT=D12.2M, MISSING=ON,$
    .
    .
    .
```

ATODBL converts the value supplied for TCSAL to double-precision format:

```
MODIFY FILE EMPLOYEE
COMPUTE TCSAL/A12=;
PROMPT EID
MATCH EID
ON NOMATCH REJECT
ON MATCH TYPE "EMPLOYEE <D.LAST_NAME <D.FIRST_NAME"
ON MATCH TYPE "ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE"
ON MATCH PROMPT TCSAL
ON MATCH COMPUTE
CSAL MISSING ON = IF TCSAL EQ 'N/A' THEN MISSING
                ELSE ATODBL(TCSAL, '12', 'D12.2');
ON MATCH TYPE "SALARY NOW <CSAL"
DATA
```

A sample execution is:

```
 EMPLOYEE    ON 11/14/96 AT 13.42.55
 DATA FOR TRANSACTION    1
 EMP_ID      =
071382660
 EMPLOYEE STEVENS ALFRED
```

```
 ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE
 TCSAL       =
N/A
 SALARY NOW              .
 DATA FOR TRANSACTION    2
 EMP_ID      =
112847612
 EMPLOYEE SMITH MARY
 ENTER CURRENT SALARY OR 'N/A' IF NOT AVAILABLE
 TCSAL       =
45000
 SALARY NOW      $45,000.00
 DATA FOR TRANSACTION    3
 EMP_ID      =
end
 TRANSACTIONS:         TOTAL =     2  ACCEPTED=     2  REJECTED=     0
 SEGMENTS:             INPUT =     0  UPDATED =     0  DELETED =     0
```

The procedure processes as follows:

1. For the first transaction, the procedure prompts for an employee ID. You enter 071382660.
2. The procedure displays the last and first name of the employee, STEVENS ALFRED.
3. The procedure prompts for a current salary. You enter N/A.
4. A period displays.
5. For the second transaction, the procedure prompts for an employee ID. You enter 112847612.
6. The procedure displays the last and first name of the employee, SMITH MARY.
7. Then it prompts for a current salary. Enter 45000.
8. $45,000.00 displays.

EDIT: Converting the Format of a Field

The EDIT function converts an alphanumeric field that contains numeric characters to numeric format or converts a numeric field to alphanumeric format.

This function is useful for manipulating a field in an expression that performs an operation that requires operands in a particular format.

When EDIT assigns a converted value to a new field, the format of the new field must correspond to the format of the returned value. For example, if EDIT converts a numeric field to alphanumeric format, you must give the new field an alphanumeric format:

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE);
```

EDIT deals with a symbol in the following way:

- When an alphanumeric field is converted to numeric format, a sign or decimal point in the field is stored as part of the numeric value.

  Any other non-numeric characters are invalid, and EDIT returns the value zero.

- When converting a floating-point or packed-decimal field to alphanumeric format, EDIT removes the sign, the decimal point, and any number to the right of the decimal point. It then right-justifies the remaining digits and adds leading zeros to achieve the specified field length. Converting a number with more than nine significant digits in floating-point or packed-decimal format may produce an incorrect result.

Convert the Format of a Field

```
EDIT(fieldname);
```

where:

fieldname

Alphanumeric or Numeric

Is the field name.

Converting From Numeric to Alphanumeric Format

EDIT converts HIRE_DATE (a legacy date format) to alphanumeric format. CHGDAT is then able to use the field, which it expects in alphanumeric format:

```
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AND COMPUTE
ALPHA_HIRE/A17 = EDIT(HIRE_DATE); NOPRINT AND COMPUTE
```

```
HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME  FIRST_NAME  HIRE_DATE  HIRE_MDY
---------  ----------  ---------  --------
BLACKWOOD  ROSEMARIE   82/04/01   APRIL 01 1982
CROSS      BARBARA     81/11/02   NOVEMBER 02 1981
GREENSPAN  MARY        82/04/01   APRIL 01 1982
JONES      DIANE       82/05/01   MAY 01 1982
MCCOY      JOHN        81/07/01   JULY 01 1981
SMITH      MARY        81/07/01   JULY 01 1981
```

FPRINT: Converting Fields to Alphanumeric Format

The FPRINT function converts any type of field except for a text field to its alphanumeric equivalent for display. The alphanumeric representation includes any display options that are specified in the format of the original field.

Convert Fields Using FPRINT

```
FPRINT(in_value, 'usageformat', output)
```

where:

in_value

Any format except TX

Is the value to be converted.

usageformat

Alphanumeric

Is the usage format of the value to be converted, including display options. The format must be enclosed in single quotation marks.

output

Alphanumeric

Is the name of the output field or its format enclosed in single quotation marks.

The output format must be long enough to hold the converted number itself, with a sign and decimal point, plus any additional characters generated by display options, such as commas, a currency symbol, or a percent sign.

For example, the D12.2 format is converted to A14 because it outputs two decimal digits, a decimal point, a possible minus sign, up to eight integer digits, and two commas. If the output format is not large enough, excess right-hand characters may be truncated.

Usage Notes for the FPRINT Function

- The USAGE format must match the actual data in the field.
- The output of FPRINT for numeric values is right-justified within the area required for the maximum number of characters corresponding to the supplied format. This ensures that all possible values are aligned vertically along the decimal point or units digit.
- By default, the column title is left justified for alphanumeric fields. To right justify the column title, use the /R reformatting option for the field.

Converting Numeric Fields to Alphanumeric Format

The following request against the EMPLOYEE data source uses FPRINT to convert the CURR_SAL, ED_HRS, and BANK_ACCT fields to alphanumeric for display on the report output. Then, the STRREP function replaces the blanks in the alphanumeric representation of CURR_SAL with asterisks. CURR_SAL has format D12.2M, so the alphanumeric representation has format A15. The ED_HRS field has format F6.2, so the alphanumeric representation has format A6. The BANK_ACCT field has format I9S, so the alphanumeric representation has format A9. The alphanumeric representations of the numeric fields are right-justified. The /R options in the PRINT command cause the column titles to be right-justified over the values:

```
DEFINE FILE EMPLOYEE
ASAL/A15 = FPRINT(CURR_SAL, 'D12.2M', ASAL);
ASAL/A15 = STRREP(15, ASAL, 1, ' ', 1, '*', 15, ASAL);
AED/A6 = FPRINT(ED_HRS, 'F6.2', AED);
ABANK/A9 = FPRINT(BANK_ACCT, 'I9S', ABANK);
END
TABLE FILE EMPLOYEE
PRINT CURR_SAL ASAL
ED_HRS AED/R
BANK_ACCT ABANK/R
WHERE BANK_NAME NE ' '
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
  CURR_SAL  ASAL              ED_HRS     AED  BANK_ACCT      ABANK
  --------  ----              ------  ------  ---------  ---------
$18,480.00  *****$18,480.00    50.00   50.00   40950036   40950036
$29,700.00  *****$29,700.00      .00     .00     160633     160633
$26,862.00  *****$26,862.00    30.00   30.00  819000702  819000702
$21,780.00  *****$21,780.00    75.00   75.00  122850108  122850108
$16,100.00  *****$16,100.00    50.00   50.00  136500120  136500120
$27,062.00  *****$27,062.00    45.00   45.00  163800144  163800144
```

Converting Alphanumeric and Numeric Date Fields to Alphanumeric Format

The following request against the EMPLOYEE data source converts the HIRE_DATE field to alphanumeric format. It also creates an alphanumeric date field named ADATE and converts it to its alphanumeric representation. The HIRE_DATE field has format I6YMD and the ADATE field has format A6YMD, so the alphanumeric representations have format A8 to account for the slashes between the date components. The /R option right-justifies the column titles over the field values:

```
DEFINE FILE EMPLOYEE
AHDATE/A8 = FPRINT(HIRE_DATE,'I6YMD', AHDATE);
```

```
ADATE/A6YMD = EDIT(HIRE_DATE);
AADATE/A8 = FPRINT(ADATE,'A6YMD', AADATE);
END
TABLE FILE EMPLOYEE
PRINT HIRE_DATE AHDATE/R
ADATE AADATE/R
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
HIRE_DATE    AHDATE  ADATE       AADATE
---------  --------  -----     --------
 80/06/02  80/06/02  80/06/02  80/06/02
 81/07/01  81/07/01  81/07/01  81/07/01
 82/05/01  82/05/01  82/05/01  82/05/01
 82/01/04  82/01/04  82/01/04  82/01/04
 82/08/01  82/08/01  82/08/01  82/08/01
 82/01/04  82/01/04  82/01/04  82/01/04
 82/07/01  82/07/01  82/07/01  82/07/01
 81/07/01  81/07/01  81/07/01  81/07/01
 82/04/01  82/04/01  82/04/01  82/04/01
 82/02/02  82/02/02  82/02/02  82/02/02
 82/04/01  82/04/01  82/04/01  82/04/01
 81/11/02  81/11/02  81/11/02  81/11/02
```

Converting a Date Field to Alphanumeric Format

The following request against the VIDEOTRK data source converts the TRANSDATE (YMD) field to alphanumeric format. The alphanumeric representation has format A8 to account for the slashes between the date components:

```
DEFINE FILE VIDEOTRK
ALPHA_DATE/A8  = FPRINT(TRANSDATE,'YMD', ALPHA_DATE);
END
TABLE FILE VIDEOTRK
PRINT TRANSDATE ALPHA_DATE
WHERE TRANSDATE LE '91/06/20'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
TRANSDATE  ALPHA_DATE
---------  ----------
91/06/20   91/06/20
91/06/19   91/06/19
91/06/18   91/06/18
91/06/19   91/06/19
91/06/18   91/06/18
91/06/19   91/06/19
91/06/17   91/06/17
91/06/20   91/06/20
91/06/20   91/06/20
91/06/19   91/06/19
91/06/18   91/06/18
91/06/20   91/06/20
91/06/18   91/06/18
91/06/17   91/06/17
91/06/17   91/06/17
91/06/17   91/06/17
91/06/19   91/06/19
91/06/17   91/06/17
```

Converting a Date-Time Field to Alphanumeric Format and Creating a HOLD File

The following request against the VIDEOTR2 data source converts the TRANSDATE (HYYMDI) field to alphanumeric format. The alphanumeric representation has format A16 to account for a four-digit year, two-digit month, two-digit day, two slashes between the date components, a space between the date and time, a two-digit hour, a colon between the hour and minute components, and a two-digit minute:

```
DEFINE FILE VIDEOTR2
DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
ALPHA_DATE/A16 = FPRINT(TRANSDATE,'HYYMDI', ALPHA_DATE);
END
TABLE FILE VIDEOTR2
PRINT TRANSDATE ALPHA_DATE/R
WHERE DATE EQ '1991'
ON TABLE SET PAGE NOPAGE
END
```

The output is:

```
TRANSDATE               ALPHA_DATE
---------         ----------------
1991/06/27 02:45  1991/06/27 02:45
1991/06/20 05:15  1991/06/20 05:15
1991/06/21 07:11  1991/06/21 07:11
1991/06/21 01:10  1991/06/21 01:10
1991/06/19 07:18  1991/06/19 07:18
1991/06/19 04:11  1991/06/19 04:11
1991/06/25 01:19  1991/06/25 01:19
1991/06/24 04:43  1991/06/24 04:43
1991/06/24 02:08  1991/06/24 02:08
1991/06/25 01:17  1991/06/25 01:17
1991/06/27 01:17  1991/06/27 01:17
1991/11/17 11:28  1991/11/17 11:28
1991/06/24 10:27  1991/06/24 10:27
```

If you hold the output in a comma-delimited or other alphanumeric output file, you can see that while the original field propagates only the numeric representation of the value, the converted field propagates the display options as well:

```
DEFINE FILE VIDEOTR2
DATE/I4 = HPART(TRANSDATE, 'YEAR', 'I4');
ALPHA_DATE/A16 = FPRINT(TRANSDATE,'HYYMDI', ALPHA_DATE);
END
TABLE FILE VIDEOTR2
PRINT TRANSDATE ALPHA_DATE/R
WHERE DATE EQ '1991'
ON TABLE HOLD FORMAT COMMA
END
```

The HOLD file follows. The first field represents the original data, and the second field contains the converted values with display options:

```
"19910627024500000","1991/06/27 02:45"
"19910620051500000","1991/06/20 05:15"
"19910621071100000","1991/06/21 07:11"
"19910621011000000","1991/06/21 01:10"
"19910619071800000","1991/06/19 07:18"
"19910619041100000","1991/06/19 04:11"
"19910625011900000","1991/06/25 01:19"
"19910624044300000","1991/06/24 04:43"
"19910624020800000","1991/06/24 02:08"
"19910625011700000","1991/06/25 01:17"
"19910627011700000","1991/06/27 01:17"
```

```
"19911117112800000","1991/11/17 11:28"
"19910624102700000","1991/06/24 10:27"
```

FTOA: Converting a Number to Alphanumeric Format

The FTOA function converts a number up to 16 digits long from numeric format to alphanumeric format. It retains the decimal positions of the number and right-justifies it with leading spaces. You can also add edit options to a number converted by FTOA.

When using FTOA to convert a number containing decimals to a character string, you must specify an alphanumeric format large enough to accommodate both the integer and decimal portions of the number. For example, a D12.2 format is converted to A14. If the output format is not large enough, decimals are truncated.

Convert a Number to Alphanumeric Format

```
FTOA(number, '(format)', output)
```

where:

number

Numeric F or D (single and double precision floating-point)

Is the number to be converted, or the name of the field that contains the number.

format

Alphanumeric

Is the format of the number to be converted enclosed in parentheses. Only floating point single-precision and double-precision formats are supported. Include any edit options that you want to appear in the output. The D (floating-point double-precision) format automatically supplies commas.

If you use a field name for this argument, specify the name without quotation marks or parentheses. If you specify a format, the format must be enclosed in single quotation marks and parentheses.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The length of this argument must be greater than the length of *number* and must account for edit options and a possible negative sign.

Converting From Numeric to Alphanumeric Format

FTOA converts the GROSS field from floating point double-precision to alphanumeric format and stores the result in ALPHA_GROSS:

```
TABLE FILE EMPLOYEE
PRINT GROSS AND COMPUTE
ALPHA_GROSS/A15 = FTOA(GROSS, '(D12.2)', ALPHA_GROSS);
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME
WHERE (GROSS GT 800) AND (GROSS LT 2300);
END
```

The output is:

```
LAST_NAME  GROSS       ALPHA_GROSS
---------  -----       -----------
BLACKWOOD  $1,815.00      1,815.00
CROSS      $2,255.00      2,255.00
IRVING     $2,238.50      2,238.50
JONES      $1,540.00      1,540.00
MCKNIGHT   $1,342.00      1,342.00
ROMANS     $1,760.00      1,760.00
SMITH      $1,100.00      1,100.00
STEVENS      $916.67        916.67
```

HEXBYT: Converting a Decimal Integer to a Character

The HEXBYT function obtains the ASCII, EBCDIC, or Unicode character equivalent of a decimal integer, depending on your configuration and operating environment. The decimal value you specify must be the value associated with the character on the configured code page. HEXBYT returns a single alphanumeric character in the ASCII, EBCDIC, or Unicode character set. You can use this function to produce characters that are not on your keyboard, similar to the CTRAN function.

In Unicode configurations, this function uses values in the range:

- 0 to 255 for 1-byte characters.
- 256 to 65535 for 2-byte characters.
- 65536 to 16777215 for 3-byte characters.
- 16777216 to 4294967295 for 4-byte characters (primarily for EBCDIC).

The display of special characters depends on your software and hardware; not all special characters may appear.

Convert a Decimal Integer to a Character

```
HEXBYT(decimal_value, output)
```

where:

decimal_value

Integer

Is the decimal integer to be converted to a single character. In non-Unicode environments, a value greater than 255 is treated as the remainder of *decimal_value* divided by 256. The decimal value you specify must be the value associated with the character on the configured code page.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks (').

Converting a Decimal Integer to a Character in ASCII and Unicode

The following request uses HEXBYT to convert the decimal integer value 130 to the comma character on ASCII code page 1252. The comma is then concatenated between LAST_NAME and FIRST_NAME to create the NAME field:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND
COMPUTE COMMA1/A1 = HEXBYT(130, COMMA1); NOPRINT
COMPUTE NAME/A40 = LAST_NAME || COMMA1| ' '| FIRST_NAME;
BY LAST_NAME NOPRINT
BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

FIRST_NAME	LAST_NAME	NAME
ROSEMARIE	BLACKWOOD	BLACKWOOD, ROSEMARIE
BARBARA	CROSS	CROSS, BARBARA
MARY	GREENSPAN	GREENSPAN, MARY
DIANE	JONES	JONES, DIANE
JOHN	MCCOY	MCCOY, JOHN
MARY	SMITH	SMITH, MARY

To produce the same output in a Unicode environment configured for code page 65001, replace the COMPUTE command for the field COMMA1 with the following syntax, in which the call to HEXBYT converts the integer value 14844058 to the comma character:

```
COMPUTE COMMA1/A1 = HEXBYT(14844058, COMMA1); NOPRINT
```

Converting a Decimal Integer to a Character

HEXBYT converts LAST_INIT_CODE to its character equivalent and stores the result in LAST_INIT:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3');
COMPUTE LAST_INIT/A1 = HEXBYT(LAST_INIT_CODE, LAST_INIT);
WHERE DEPARTMENT EQ 'MIS';
END
```

The output for an ASCII platform is:

```
LAST_NAME LAST_INIT_CODE LAST_INIT
--------- -------------- ---------
SMITH                 83 S
JONES                 74 J
MCCOY                 77 M
BLACKWOOD             66 B
GREENSPAN             71 G
CROSS                 67 C
```

The output for an EBCDIC platform is:

```
LAST_NAME LAST_INIT_CODE LAST_INIT
--------- -------------- ---------
SMITH                226 S
JONES                209 J
MCCOY                212 M
BLACKWOOD            194 B
GREENSPAN            199 G
CROSS                195 C
```

Inserting Braces for Mainframe

HEXBYT converts the decimal integer 192 to its EBCDIC character equivalent, which is a left brace; and the decimal integer 208 to its character equivalent, which is a right brace. If the value of CURR_SAL is less than 12000, the value of LAST_NAME is enclosed in braces.

```
DEFINE FILE EMPLOYEE
BRACE/A17 = HEXBYT(192, 'A1') | LAST_NAME | HEXBYT(208, 'A1');
BNAME/A17 = IF CURR_SAL LT 12000 THEN BRACE
ELSE LAST_NAME;
END
TABLE FILE EMPLOYEE
PRINT BNAME CURR_SAL BY EMP_ID
END
```

The output is:

```
EMP_ID     BNAME                          CURR_SAL
------     -----                          --------
071382660  {STEVENS        }            $11,000.00
112847612  SMITH                        $13,200.00
117593129  JONES                        $18,480.00
119265415  {SMITH          }             $9,500.00
119329144  BANNING                      $29,700.00
123764317  IRVING                       $26,862.00
126724188  ROMANS                       $21,120.00
219984371  MCCOY                        $18,480.00
326179357  BLACKWOOD                    $21,780.00
451123478  MCKNIGHT                     $16,100.00
543729165  {GREENSPAN      }             $9,000.00
818692173  CROSS                        $27,062.00
```

ITONUM: Converting a Large Binary Integer to Double-Precision Format

The ITONUM function converts a large binary integer in a non-FOCUS data source to double-precision format.

Some programming languages and some non-FOCUS data storage systems use large binary integer formats. However, large binary integers (more than 4 bytes in length) are not supported in the Master File so they require conversion to double-precision format.

You must specify how many of the right-most bytes in the input field are significant. The result is an 8-byte double-precision field.

Convert a Large Binary Integer to Double-Precision Format

```
ITONUM(maxbytes, infield, output)
```

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.

output

Double precision floating-point (D*n*)

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be D*n*.

Converting a Large Binary Integer to Double-Precision Format

Suppose a binary number in an external file has the following COBOL format:

```
PIC 9(8)V9(4) COMP
```

It is defined in the EUROCAR Master File as a field named BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The following request converts the field to double-precision format:

```
DEFINE FILE EUROCAR
MYFLD/D14 = ITONUM(6, BINARYFLD, MYFLD);
END
TABLE FILE EUROCAR
PRINT MYFLD BY CAR
END
```

ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format

The ITOPACK function converts a large binary integer in a non-FOCUS data source to packed-decimal format.

Some programming languages and some non-FOCUS data storage systems use double-word binary integer formats. These are similar to the single-word binary integers used by FOCUS, but they allow larger numbers. However, large binary integers (more than 4 bytes in length) are not supported in the Master File so they require conversion to packed-decimal format.

You must specify how many of the right-most bytes in the input field are significant. The result is an 8-byte packed-decimal field of up to 15 significant numeric positions (for example, P15 or P16.2).

Limit: For a field defined as 'PIC 9(15) COMP' or the equivalent (15 significant digits), the maximum number that can be converted is 167,744,242,712,576.

Convert a Large Binary Integer to Packed-Decimal Format

```
ITOPACK(maxbytes, infield, output)
```

where:

maxbytes

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign.

Valid values are:

- **5** ignores the left-most 3 bytes (up to 11 significant positions).
- **6** ignores the left-most 2 bytes (up to 14 significant positions).
- **7** ignores the left-most byte (up to 15 significant positions).

infield

A8

Is the field that contains the binary number. Both the USAGE and ACTUAL formats of the field must be A8.

output

Numeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The format must be P*n* or P*n.d*.

Converting a Large Binary Integer to Packed-Decimal Format

Suppose a binary number in an external file has the following COBOL format:

```
PIC 9(8)V9(4) COMP
```

It is defined in the EUROCAR Master File as a field named BINARYFLD. Its field formats are USAGE=A8 and ACTUAL=A8, since its length is greater than 4 bytes.

The following request converts the field to packed-decimal format:

```
DEFINE FILE EUROCAR
PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, PACKFLD);
END
TABLE FILE EUROCAR
PRINT PACKFLD BY CAR
END
```

ITOZ: Converting a Number to Zoned Format

The ITOZ function converts a number in numeric format to zoned-decimal format. Although a request cannot process zoned numbers, it can write zoned fields to an extract file for use by an external program.

Convert a Number to Zoned Format

```
ITOZ(length, in_value, output)
```

where:

length

Integer

Is the length of *in_value* in bytes. The maximum number of bytes is 15. The last byte includes the sign.

in_value

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.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Converting a Number to Zoned Format

The following request creates an extract file containing employee IDs and salaries in zoned format for a COBOL program:

```
DEFINE FILE EMPLOYEE
ZONE_SAL/A8 = ITOZ(8, CURR_SAL, ZONE_SAL);
END
```

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL ZONE_SAL BY EMP_ID
ON TABLE SAVE AS SALARIES
END
```

The resulting extract file is:

```
NUMBER OF RECORDS IN TABLE= 12 LINES= 12

[EBCDIC|ALPHANUMERIC]  RECORD  NAMED  SALARIES
FIELDNAME                                   ALIAS       FORMAT
LENGTH
EMP_ID                                      EID         A9           9
CURR_SAL                                    CSAL        D12.2M      12
ZONE_SAL                                                A8           8
TOTAL                                                               29
DCB USED WITH FILE SALARIES IS DCB=(RECFM=FB,LRECL=00029,BLKSIZE=00580)
```

If you remove the SAVE command and run the request, the output for an EBCDIC platform follows. The left brace in EBCDIC is hexadecimal C0; this indicates a positive sign and a final digit of 0. The capital B in EBCDIC is hexadecimal C2; this indicates a positive sign and a final digit of 2.

```
EMP_ID             URR_SAL  ZONE_SAL
------            --------  --------
071382660       $11,000.00  0001100{
112847612       $13,200.00  0001320{
117593129       $18,480.00  0001848{
119265415        $9,500.00  0000950{
119329144       $29,700.00  0002970{
123764317       $26,862.00  0002686B
```

```
126724188       $21,120.00  0002112{
219984371       $18,480.00  0001848{
326179357       $21,780.00  0002178{
451123478       $16,100.00  0001610{
543729165        $9,000.00  0000900{
818692173       $27,062.00 0002706B
```

PCKOUT: Writing a Packed Number of Variable Length

The PCKOUT function writes a packed-decimal number of variable length to an extract file. When a request saves a packed number to an extract file, it typically writes it as an 8- or 16-byte field regardless of its format specification. With PCKOUT, you can vary the field's length between 1 to 16 bytes.

Write a Packed Number of Variable Length

```
PCKOUT(in_value, length, output)
```

where:

in_value

Numeric

Is the input field that contains the values. It can be in packed, integer, single- or double-precision floating point format. If it is not in integer format, it is rounded to the nearest whole number.

length

Numeric

Is the length of the output value, from 1 to 16 bytes.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The function returns the field as alphanumeric although it contains packed data.

Writing a Packed Number of Variable Length

PCKOUT converts the CURR_SAL field to a 5-byte packed field and stores the result in SHORT_SAL:

```
DEFINE FILE EMPLOYEE
SHORT_SAL/A5 = PCKOUT(CURR_SAL, 5, SHORT_SAL);
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME SHORT_SAL HIRE_DATE
ON TABLE SAVE
END
```

The resulting extract file is:

```
>
NUMBER OF RECORDS IN TABLE=       12 LINES=     12
[EBCDIC|ALPHANUMERIC]  RECORD NAMED   SAVE
FIELDNAME                         ALIAS         FORMAT          LENGTH
LAST_NAME                         LN            A15               15
SHORT_SAL                                       A5                 5
HIRE_DATE                         HDT           I6YMD              6
TOTAL                                                             26
DCB USED WITH FILE SAVE IS DCB=(RECFM=FB,LRECL=00026,BLKSIZE=00520)
```

PTOA: Converting a Packed-Decimal Number to Alphanumeric Format

The PTOA function converts a packed-decimal number from numeric format to alphanumeric format. It retains the decimal positions of the number and right-justifies it with leading spaces. You can also add edit options to a number converted by PTOA.

When using PTOA to convert a number containing decimals to a character string, you must specify an alphanumeric format large enough to accommodate both the integer and decimal portions of the number. For example, a P12.2C format is converted to A14. If the output format is not large enough, the right-most characters are truncated.

Convert a Packed-Decimal Number to Alphanumeric Format

```
PTOA(number, '(format)', output)
```

where:

number

Numeric P (packed-decimal)

Is the number to be converted, or the name of the field that contains the number.

format

Alphanumeric

Is the format of the number enclosed in both single quotation marks and parentheses.

Only packed-decimal format is supported. Include any edit options that you want to display in the output.

The format value does not require the same length or number of decimal places as the original field. If you change the number of decimal places, the result is rounded. If you make the length too short to hold the integer portion of the number, asterisks appear instead of the number.

If you use a field name for this argument, specify the name without quotation marks or parentheses. However, parentheses must be included around the format stored in this field. For example:

```
FMT/A10 = '(P12.2C)';
```

You can then use this field as the format argument when using the function in your request:

```
COMPUTE ALPHA_GROSS/A20 = PTOA(PGROSS, FMT, ALPHA_GROSS);
```

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The length of this argument must be greater than the length of *number* and must account for edit options and a possible negative sign.

Converting From Packed to Alphanumeric Format

PTOA is called twice to convert the PGROSS field from packed-decimal to alphanumeric format. The format specified in the first call to the function is stored in a virtual field named FMT. The format specified in the second call to the function does not include decimal places, so the value is rounded when it appears:

```
DEFINE FILE EMPLOYEE
PGROSS/P18.2=GROSS;
FMT/A10='(P14.2C)';
END
TABLE FILE EMPLOYEE PRINT PGROSS NOPRINT
COMPUTE AGROSS/A17 = PTOA(PGROSS, FMT, AGROSS); AS ''
COMPUTE BGROSS/A37 = '<- THIS AMOUNT IS' |
                     PTOA(PGROSS, '(P5C)', 'A6') |
                     ' WHEN ROUNDED'; AS '' IN +1
BY HIGHEST 1 PAY_DATE NOPRINT
BY LAST_NAME NOPRINT
END
```

The output is:

```
2,475.00 <- THIS AMOUNT IS 2,475 WHEN ROUNDED
1,815.00 <- THIS AMOUNT IS 1,815 WHEN ROUNDED
2,255.00 <- THIS AMOUNT IS 2,255 WHEN ROUNDED
  750.00 <- THIS AMOUNT IS   750 WHEN ROUNDED
2,238.50 <- THIS AMOUNT IS 2,239 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,540.00 <- THIS AMOUNT IS 1,540 WHEN ROUNDED
1,342.00 <- THIS AMOUNT IS 1,342 WHEN ROUNDED
1,760.00 <- THIS AMOUNT IS 1,760 WHEN ROUNDED
1,100.00 <- THIS AMOUNT IS 1,100 WHEN ROUNDED
  791.67 <- THIS AMOUNT IS   792 WHEN ROUNDED
  916.67 <- THIS AMOUNT IS   917 WHEN ROUNDED
```

UFMT: Converting an Alphanumeric String to Hexadecimal

The UFMT function converts characters in an alphanumeric source string to their hexadecimal representation. This function is useful for examining data of unknown format. As long as you know the length of the data, you can examine its content.

Convert an Alphanumeric String to Hexadecimal

```
UFMT(source_string, length, output)
```

where:

source_string

Alphanumeric

Is the alphanumeric string to convert enclosed in single quotation marks ('), or the field that contains the string.

length

Integer

Is the number of characters in *source_string*.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks ('). The format of *output* must be alphanumeric and its length must be twice that of *length*.

Converting an Alphanumeric String to Hexadecimal

UFMT converts each value in JOBCODE to its hexadecimal representation and stores the result in HEXCODE:

```
DEFINE FILE JOBFILE
HEXCODE/A6 = UFMT(JOBCODE, 3, HEXCODE);
END
TABLE FILE JOBFILE
PRINT JOBCODE HEXCODE
END
```

The output is:

```
JOBCODE  HEXCODE
-------  -------
A01      C1F0F1
A02      C1F0F2
A07      C1F0F7
A12      C1F1F2
A14      C1F1F4
A15      C1F1F5
A16      C1F1F6
A17      C1F1F7
B01      C2F0F1
B02      C2F0F2
B03      C2F0F3
B04      C2F0F4
B14      C2F1F4
```

XTPACK: Writing a Packed Number With Up to 31 Significant Digits to an Output File

The XTPACK function stores packed numbers with up to 31 significant digits in an alphanumeric field, retaining decimal data. This permits writing a short or long packed field of any length, 1 to 16 bytes, to an output file.

Store Packed Values in an Alphanumeric Field

```
XTPACK(in_value, outlength, outdec, output)
```

where:

infield

Numeric

Is the packed value.

outlength

Numeric

Is the length of the alphanumeric field that will hold the converted packed field. Can be from 1 to 16.

outdec

Numeric

Is the number of decimal positions for *output*.

output

Alphanumeric

Is the name of the field to contain the result or the format of the field enclosed in single quotation marks.

Writing a Long Packed Number to an Output File

The following request creates a long packed decimal field named LONGPCK. ALPHAPCK (format A13) is the result of applying XTPACK to the long packed field. PCT_INC, LONGPCK, and ALPHAPCK are then written to a SAVE file named XTOUT.

```
DEFINE FILE EMPLOYEE
LONGPCK/P25.2 = PCT_INC + 1111111111111111111;
ALPHAPCK/A13 = XTPACK(LONGPCK,13,2,'A13');
END
TABLE FILE EMPLOYEE
```

```
PRINT PCT_INC LONGPCK ALPHAPCK
WHERE PCT_INC GT 0
  ON TABLE SAVE AS XTOUT
END
```

The SAVE file has the following fields and formats:

```
ALPHANUMERIC RECORD NAMED  XTOUT
FIELDNAME                        ALIAS         FORMAT        LENGTH
PCT_INC                          PI            F6.2            6
LONGPCK                                        P25.2          25
ALPHAPCK                                       A13            13
TOTAL                                                         44
SAVED...
```

Simplified Numeric Functions

Numeric functions have been developed that make it easier to understand and enter the required arguments. These functions have streamlined parameter lists, similar to those used by SQL functions. In some cases, these simplified functions provide slightly different functionality than previous versions of similar functions.

The simplified functions do not have an output argument. Each function returns a value that has a specific data type.

When used in a request against a relational data source, these functions are optimized (passed to the RDBMS for processing).

Note:

- The simplified numeric functions are supported in Dialogue Manager.

ASCII: Returning the ASCII Code for the Leftmost Character in a String

ASCII takes a character string and returns the ASCII code in integer format for the leftmost character in the string.

Return the ASCII Code for the Leftmost Character in a String

```
ASCII(charexp)
```

where:

charexp

Is any character string.

Returning the ASCII Code for the Leftmost Character in a String

In the following request, ASCII returns the ASCII code for the leftmost character in the CATEGORY field.

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT
AND COMPUTE
ASCII_CODE/I9 = ASCII(CATEGORY);
BY CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Category	ASCII_CODE
Coffee	67
Food	70
Gifts	71

CEILING: Returning the Smallest Integer Value Greater Than or Equal to a Value

CEILING returns the smallest integer value that is greater than or equal to a number.

Return the Smallest Integer Greater Than or Equal to a Number

```
CEILING(number)
```

where:

number

Numeric

Is the number whose ceiling will be returned. The output data type is the same as the input data type.

Returning the Ceiling of a Number

In the following request, CEILING returns the smallest integer greater than or equal to the GROSS_PROFIT_US value.

```
DEFINE FILE WFLITE
CEIL1/D7.2= CEILING(GROSS_PROFIT_US);
END
TABLE FILE WFLITE
PRINT GROSS_PROFIT_US/D9.2  CEIL1
ON TABLE SET PAGE NOPAGE
END
```

The partial output follows. Note that even though the value returned is an integer, it is returned with the same format as the CEIL1 field (D7.2):

```
  Gross Profit     CEIL1
  ------------     -----
        165.00    165.00
         13.99     14.00
         60.99     61.00
        225.98    226.00
         79.99     80.00
         44.59     45.00
         94.30     95.00
        238.50    239.00
```

```
         199.99    200.00
          68.99     69.00
          63.58     64.00
         129.99    130.00
          37.49     38.00
          75.99     76.00
          13.99     14.00
         119.00    119.00
         -30.01    -30.00
          54.99     55.00
         189.98    190.00
          44.59     45.00
          91.98     92.00
          89.00     89.00
          59.50     60.00
         129.99    130.00
          54.00     54.00
         109.98    110.00
          98.99     99.00
          98.99     99.00
          99.99    100.00
          44.59     45.00
```

EXPONENT: Raising e to a Power

EXPONENT raises the constant **e** to a power.

Raise the Constant e to a Power

```
EXPONENT(power)
```

where:

power

Numeric

Is the power to which to raise **e**. The output data type is numeric.

Raising e to a Power

The following request prints the value of e and the value of e raised to the fifth power.

```
DEFINE FILE WFLITE
EXP1/D12.5 = EXPONENT(1);
EXP2/D12.5 = EXPONENT(5);
END
TABLE FILE WFLITE
PRINT EXP1 EXP2
BY BUSINESS_REGION AS Region
WHERE BUSINESS_REGION EQ 'EMEA'
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOPAGE
END
```

The output is shown in the following image.

Region	EXP1	EXP2
EMEA	2.71828	148.41316

FLOOR: Returning the Largest Integer Less Than or Equal to a Value

FLOOR returns the largest integer value that is less than or equal to a number.

Return the Largest Integer Less Than or Equal to a Number

```
FLOOR(number)
```

where:

number

Numeric

Is the number whose floor will be returned. The output data type is the same as the input data type.

Returning the Floor of a Number

In the following request, FLOOR returns the largest integer less than or equal to the GROSS_PROFIT_US value.

```
DEFINE FILE WFLITE
FLOOR1/D7.2= FLOOR(GROSS_PROFIT_US);
END
TABLE FILE WFLITE
PRINT GROSS_PROFIT_US/D9.2  FLOOR1
ON TABLE SET PAGE NOPAGE
END
```

Partial output follows. Note that even though the value returned is an integer, it is returned with the same format as the FLOOR1 field (D7.2):

```
  Gross Profit    FLOOR1
  ------------    ------
        165.00    165.00
         13.99     13.00
         60.99     60.00
        225.98    225.00
         79.99     79.00
         44.59     44.00
         94.30     94.00
        238.50    238.00
        199.99    199.00
         68.99     68.00
         63.58     63.00
        129.99    129.00
         37.49     37.00
         75.99     75.00
         13.99     13.00
        119.00    119.00
        -30.01    -31.00
         54.99     54.00
        189.98    189.00
```

```
         44.59     44.00
         91.98     91.00
         89.00     89.00
         59.50     59.00
        129.99    129.00
         54.00     54.00
        109.98    109.00
         98.99     98.00
         98.99     98.00
         99.99     99.00
         44.59     44.00
```

LOG10: Calculating the Base 10 Logarithm

LOG10 returns the base-10 logarithm of a numeric expression.

Calculate the Base 10 Logarithm

```
LOG10(num_exp)
```

where:

num_exp

Numeric

Is the numeric value for which to calculate the base 10 logarithm.

Calculating the Base 10 Logarithm

The following request calculates the base 10 logarithm of current salaries.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
LOG_CURR_SAL/D12.6 = LOG10(CURR_SAL);
BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
```

```
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

LAST_NAME	FIRST_NAME	CURR_SAL	LOG_CURR_SAL
BANNING	JOHN	$29,700.00	4.472756
IRVING	JOAN	$26,862.00	4.429138
MCKNIGHT	ROGER	$16,100.00	4.206826
ROMANS	ANTHONY	$21,120.00	4.324694
SMITH	RICHARD	$9,500.00	3.977724
STEVENS	ALFRED	$11,000.00	4.041393

MOD: Calculating the Remainder From a Division

MOD calculates the remainder from a division. The output data type is the same as the input data type.

Calculate the Remainder From a Division

```
MOD(dividend, divisor)
```

where:

dividend

Numeric

Is the value to divide.

> **Note:** The sign of the returned value will be the same as the sign of the dividend.

divisor

Numeric

Is the value to divide by.

If the divisor is zero (0), the dividend is returned.

Calculating the Remainder From a Division

In the following request, MOD returns the remainder of PRICE_DOLLARS divided by DAYSDELAYED:

```
DEFINE FILE WFLITE
MOD1/D7.2= MOD(PRICE_DOLLARS, DAYSDELAYED);
END
TABLE FILE WFLITE
PRINT PRICE_DOLLARS/D7.2  DAYSDELAYED/I5 MOD1
WHERE DAYSDELAYED GT 1
ON TABLE SET PAGE NOPAGE
ON TABLE PCHOLD FORMAT WP
END
```

Partial output follows:

```
  Price     Days
Dollars  Delayed      MOD1
-------  -------      ----
 399.00        3       .00
 489.99        3       .99
 786.50        2       .50
 599.99        4      3.99
  29.99        4      1.99
 169.00        2      1.00
 219.99        2      1.99
 280.00        3      1.00
  79.99        4      3.99
 145.99        2      1.99
 399.99        3       .99
```

```
 349.99        3      1.99
 169.00        3      1.00
```

POWER: Raising a Value to a Power

POWER raises a base value to a power.

Raise a Value to a Power

```
POWER(base, power)
```

where:

base

Numeric

Is the value to raise to a power. The output value has the same data type as the base value. If the base value is integer, negative power values will result in truncation.

power

Numeric

Is the power to which to raise the base value.

Raising a Base Value to a Power

In the following request, POWER returns the value COGS_US/20.00 raised to the power stored in DAYSDELAYED:

```
DEFINE FILE WFLITE
BASE=COGS_US/20.00;
POWER1= POWER(COGS_US/20.00,DAYSDELAYED);
END
TABLE FILE WFLITE
```

```
PRINT BASE IN 15 DAYSDELAYED POWER1
BY PRODUCT_CATEGORY
WHERE PRODUCT_CATEGORY EQ 'Computers'
WHERE DAYSDELAYED NE 0
ON TABLE SET PAGE NOPAGE
END
```

Partial output follows:

```
Product                        Days
Category            BASE       Delayed         POWER1
--------            ----       -------         ------
Computers          12.15             3       1,793.61
                   16.70             2         278.89
                    8.35             1           8.35
                    8.10             2          65.61
                    4.05             1           4.05
                    4.05             2          16.40
                    4.05             4         269.04
                    8.35             1           8.35
                   16.70             1          16.70
                    8.35             3         582.18
                    8.35             1           8.35
                    4.05             1           4.05
                    4.05             1           4.05
                    8.35             4       4,861.23
                    8.35            -1            .12
                    8.35             1           8.35
                    8.35             3         582.18
```

ROUND: Rounding a Number to a Given Number of Decimal Places

Given a numeric expression and an integer count, ROUND returns the numeric expression rounded to that number of decimal places. If the number of decimal places is negative, it rounds to the left of the decimal point.

Round a Number to a Given Number of Decimal Places

```
ROUND(num_exp, count)
```

where:

num_exp

Numeric

Is the numeric expression to be rounded.

count

Numeric

Is the number of decimal places to which the numeric expression is to be rounded. If the number of decimal places is negative, ROUND rounds to the left of the decimal point.

Rounding a Number to a Given Number of Decimal Places

The following request rounds the LISTPR field to zero decimal places and the NEWLISTPR field 1 decimal place and to -2 decimal places.

```
TABLE FILE MOVIES
PRINT LISTPR
AND  COMPUTE
NEWLISTPR/D12.3 = LISTPR * 99;
ROUND_ZERO/D12.3 = ROUND(LISTPR, 0);
ROUND_PLUS1/D12.3 = ROUND(NEWLISTPR, 1);
ROUND_MINUS1/D12.3 = ROUND(NEWLISTPR, -2);
BY MOVIECODE
WHERE RECORDLIMIT EQ 3
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

MOVIECODE	LISTPR	NEWLISTPR	ROUND_ZERO	ROUND_PLUS1	ROUND_MINUS2
001MCA	19.95	1,975.050	20.000	1,975.100	2,000.000
005WAR	24.98	2,473.020	25.000	2,473.000	2,500.000
020TUR	39.99	3,959.010	40.000	3,959.000	4,000.000

SIGN: Returning the Sign of a Number

SIGN takes a numeric argument and returns the value -1 if the number is negative, 0 (zero) if the number is zero, and 1 if the number is positive.

Return the Sign of a Number

```
SIGN(number)
```

where:

number

Is a field containing a numeric value or a number.

Returning the Sign of a Number

The following request returns the sign of positive numbers, negative numbers, and zero (0).

```
TABLE FILE GGSALES
SUM DOLLARS NOPRINT AND COMPUTE
PLUSDOLL/I9 = IF DOLLARS GT 12000000 THEN DOLLARS ELSE 0;
SIGN1/I5 = SIGN(PLUSDOLL);
NEGDOLL/I9 = IF DOLLARS LT 12000000 THEN 0 ELSE -DOLLARS;
SIGN2/I5 = SIGN(NEGDOLL);
BY CATEGORY
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Category	PLUSDOLL	SIGN1	NEGDOLL	SIGN2
Coffee	17231455	1	-17231455	-1
Food	17229333	1	-17229333	-1
Gifts	0	0	0	0

TRUNCATE: Truncating a Number to a Given Number of Decimal Places

Given a numeric expression and an integer count, TRUNCATE returns the numeric expression truncated to that number of decimal places. If the number of decimal places is negative, it truncates to the left of the decimal point.

Truncate a Number to a Given Number of Decimal Places

```
TRUNCATE(num_exp, count)
```

where:

num_exp

Numeric

Is the numeric expression to be truncated.

count

Numeric

Is the number of decimal places to which the numeric expression is to be truncated. If the number of decimal places is negative, TRUNCATE truncates to the left of the decimal point.

Truncating a Number to a Given Number of Decimal Places

The following request truncates the LISTPR field to 1 decimal place and to -1 decimal places.

```
TABLE FILE MOVIES
PRINT LISTPR
AND  COMPUTE
TRUNCATE_PLUS1/D12.3 = TRUNCATE(LISTPR, 1);
TRUNCATE_MINUS1/D12.3 = TRUNCATE(LISTPR, -1);
BY MOVIECODE
WHERE RECORDLIMIT EQ 3
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

MOVIECODE	LISTPR	TRUNCATE_PLUS1	TRUNCATE_MINUS1
001MCA	19.95	19.900	10.000
005WAR	24.98	24.900	20.000
020TUR	39.99	39.900	30.000

Numeric Functions

Numeric functions perform calculations on numeric constants and fields.

For many functions, the output argument can be supplied either as a field name or as a format enclosed in single quotation marks. However, if a function is called from a Dialogue Manager command, this argument must always be supplied as a forma. For detailed information about calling a function and supplying arguments, see Accessing and Calling a Function.

Note: With CDN ON, numeric arguments must be delimited by a comma followed by a space.

ABS: Calculating Absolute Value

The ABS function returns the absolute value of a number.

Calculate Absolute Value

```
ABS(in_value)
```

where:

in_value

Numeric

Is the value for which the absolute value is returned, the name of a field that contains the value, or an expression that returns the value. If you use an expression, use parentheses as needed to ensure the correct order of evaluation.

Calculating Absolute Value

The COMPUTE command creates the DIFF field, then ABS calculates the absolute value of DIFF:

```
TABLE FILE SALES
PRINT UNIT_SOLD AND DELIVER_AMT AND
COMPUTE DIFF/I5 = DELIVER_AMT - UNIT_SOLD; AND
COMPUTE ABS_DIFF/I5 = ABS(DIFF);BY PROD_CODE
WHERE DATE LE '1017';
END
```

The output is:

```
PROD_CODE  UNIT_SOLD  DELIVER_AMT  DIFF  ABS_DIFF
---------  ---------  -----------  ----  --------
B10               30           30     0         0
B17               20           40    20        20
B20               15           30    15        15
C17               12           10    -2         2
D12               20           30    10        10
E1                30           25    -5         5
E3                35           25   -10        10
```

ASIS: Distinguishing Between a Blank and a Zero

The ASIS function distinguishes between a blank and a zero in Dialogue Manager. It differentiates between a numeric string constant or variable defined as a numeric string, and a field defined simply as numeric.

For details on ASIS, see ASIS: Distinguishing Between Space and Zero.

BAR: Producing a Bar Chart

The BAR function produces a horizontal bar chart using repeating characters to form each bar. Optionally, you can create a scale to clarify the meaning of a bar chart by replacing the title of the column containing the bar with a scale.

Produce a Bar Chart

```
BAR(barlength, infield, maxvalue, 'char', output)
```

where:

barlength

Numeric

Is the maximum length of the bar, in characters. If this value is less than or equal to 0, the function does not return a bar.

infield

Numeric

Is the data field plotted as a bar chart.

maxvalue

Numeric

Is the maximum value of a bar. This value must be greater than the maximum value stored in the *infield*. If *infield* is larger than *maxvalue*, the function uses *maxvalue* and returns a bar of maximum length.

'char'

Alphanumeric

Is the repeating character that creates the bars enclosed in single quotation marks. If you specify more than one character, only the first character is used.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The output field must be large enough to contain a bar of maximum length as defined by *barlength*.

Producing a Bar Chart

BAR creates a bar chart for the CURR_SAL field, and stores the output in SAL_BAR. The bar created can be no longer than 30 characters long, and the value it represents can be no greater than 30,000.

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);BY LAST_NAME BY
FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME       FIRST_NAME         CURR_SAL  SAL_BAR
---------       ----------         --------  -------
BANNING         JOHN             $29,700.00  ===========================
IRVING          JOAN             $26,862.00  ==========================
MCKNIGHT        ROGER            $16,100.00  ================
ROMANS          ANTHONY          $21,120.00  =====================
SMITH           RICHARD           $9,500.00  ==========
STEVENS         ALFRED           $11,000.00  ===========
```

Creating a Bar Chart With a Scale

BAR creates a bar chart for the CURR_SAL field. The request then replaces the field name SAL_BAR with a scale using the AS phrase.

To run this request on a platform for which the default font is proportional, use a non-proportional font or issue SET STYLE=OFF.

```
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';
ON TABLE SET PAGE-NUM OFF

END
```

The output is:

```
CURRENT SALARIES OF EMPLOYEES IN PRODUCTION DEPARTMENT
GRAPHED IN THOUSANDS OF DOLLARS
                                                 5 10 15 20 25 30
LAST NAME    FIRST NAME    CURRENT SALARY ----+----+----+----+----+----+
---------    ----------    -------------- ------------------------------
BANNING      JOHN          $29,700.00     =============================
IRVING       JOAN          $26,862.00     ===========================
MCKNIGHT     ROGER         $16,100.00     ================
ROMANS       ANTHONY       $21,120.00     =====================
SMITH        RICHARD        $9,500.00     ==========
STEVENS      ALFRED        $11,000.00     ===========
```

CHKPCK: Validating a Packed Field

The CHKPCK function validates the data in a field described as packed format (if available on your platform). The function prevents a data exception from occurring when a request reads a field that is expected to contain a valid packed number but does not.

To use CHKPCK:

1. Ensure that the Master File (USAGE and ACTUAL attributes) or the MODIFY FIXFORM command defines the field as alphanumeric, not packed. This does *not* change the field data, which remains packed, but it enables the request to read the data without a data exception.
2. Call CHKPCK to examine the field. The function returns the output to a field defined as packed. If the value it examines is a valid packed number, the function returns the value; if the value is not packed, the function returns an error code.

Validate a Packed Field

```
CHKPCK(length, in_value, error, output)
```

where:

length

Numeric

Is the length of the packed field. It can be between 1 and 16 bytes.

infield

Alphanumeric

Is the name of the packed field or the value to be verified as packed decimal. The value must be described as alphanumeric, not packed.

error

Numeric

Is the error code that the function returns if a value is not packed. Choose an error code outside the range of data. The error code is first truncated to an integer, then converted to packed format. However, it may appear on a report with a decimal point depending on the output format.

output

Packed-decimal

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Validating Packed Data

1. Prepare a data source that includes invalid packed data. The following example creates TESTPACK, which contains the PACK_SAL field. PACK_SAL is defined as alphanumeric but actually contains packed data. The invalid packed data is stored as AAA.

```
DEFINE FILE EMPLOYEE
PACK_SAL/A8 = IF EMP_ID CONTAINS '123'
      THEN 'AAA' ELSE PCKOUT(CURR_SAL, 8, 'A8');
END
```

```
TABLE FILE EMPLOYEE
PRINT DEPARTMENT PACK_SAL BY EMP_ID
ON TABLE SAVE AS TESTPACK
END
```

The output is:

```
> NUMBER OF RECORDS IN TABLE=         12  LINES=      12
[EBCDIC|ALPHANUMERIC]  RECORD NAMED  TESTPACK
FIELDNAME                        ALIAS         FORMAT
LENGTH
EMP_ID                           EID           A9                9
DEPARTMENT                       DPT           A10              10
PACK_SAL                                       A8                8
TOTAL                                                           27
[DCB USED WITH FILE TESTPACK IS DCB=
(RECFM=FB,LRECL=00027,BLKSIZE=00540)] SAVED... >
```

2. Create a Master File for the TESTPACK data source. Define the PACK_SAL field as alphanumeric in the USAGE and ACTUAL attributes.

```
FILE  = TESTPACK,  SUFFIX = FIX
FIELD = EMP_ID    ,ALIAS = EID,USAGE = A9 ,ACTUAL = A9 ,$
FIELD = DEPARTMENT,ALIAS = DPT,USAGE = A10,ACTUAL = A10,$
FIELD = PACK_SAL  ,ALIAS = PS ,USAGE = A8 ,ACTUAL = A8 ,$
```

3. Create a request that uses CHKPCK to validate the values in the PACK_SAL field, and stores the result in the GOOD_PACK field. Values not in packed format return the error code -999. Values in packed format appear accurately.

```
DEFINE FILE TESTPACK
GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
END
```

```
TABLE FILE TESTPACK
```

```
PRINT DEPARTMENT GOOD_PACK BY EMP_ID
END
```

The output is:

```
EMP_ID     DEPARTMENT    GOOD_PACK
------     ----------    ---------
071382660  PRODUCTION      $11,000
112847612  MIS             $13,200
117593129  MIS             $18,480
119265415  PRODUCTION       $9,500
119329144  PRODUCTION      $29,700
123764317  PRODUCTION        -$999
126724188  PRODUCTION      $21,120
219984371  MIS             $18,480
326179357  MIS             $21,780
451123478  PRODUCTION        -$999
543729165  MIS              $9,000
818692173  MIS             $27,062
```

DMOD, FMOD, and IMOD: Calculating the Remainder From a Division

The MOD functions calculate the remainder from a division. Each function returns the remainder in a different format.

The functions use the following formula.

```
remainder = dividend - INT(dividend/divisor) * divisor
```

- *DMOD* returns the remainder as a decimal number.
- *FMOD* returns the remainder as a floating-point number.
- *IMOD* returns the remainder as an integer.

Calculate the Remainder From a Division

```
function(dividend, divisor, output)
```

where:

function

Is one of the following:

`DMOD` returns the remainder as a decimal number.

`FMOD` returns the remainder as a floating-point number.

`IMOD` returns the remainder as an integer.

dividend

Numeric

Is the number being divided.

divisor

Numeric

Is the number dividing the dividend.

output

Numeric

Is the result whose format is determined by the function used. Can be the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

If the divisor is zero (0), the dividend is returned.

Calculating the Remainder From a Division

IMOD divides ACCTNUMBER by 1000 and returns the remainder to LAST3_ACCT:

```
TABLE FILE EMPLOYEE
PRINT ACCTNUMBER AND COMPUTE
```

```
LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
BY LAST_NAME BY FIRST_NAME
WHERE (ACCTNUMBER NE 000000000) AND (DEPARTMENT EQ 'MIS');
END
```

The output is:

```
LAST_NAME        FIRST_NAME  ACCTNUMBER  LAST3_ACCT
---------        ----------  ----------  ----------
BLACKWOOD        ROSEMARIE   122850108          108
CROSS            BARBARA     163800144          144
GREENSPAN        MARY        150150302          302
JONES            DIANE       040950036          036
MCCOY            JOHN        109200096          096
SMITH            MARY        027300024          024
```

EXP: Raising *e* to the Nth Power

The EXP function raises the value "e" (approximately 2.72) to a specified power. This function is the inverse of the LOG function, which returns the logarithm of the argument.

EXP calculates the result by adding terms of an infinite series. If a term adds less than .000001 percent to the sum, the function ends the calculation and returns the result as a double-precision number.

Raise *e* to the Nth Power

```
EXP(power, output)
```

where:

power

Numeric

Is the power to which "e" is raised.

output

Double-precision floating-point

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Raising *e* to the Nth Power

EXP raises "e" to the power designated by the &POW variable, specified here as 3. The result is then rounded to the nearest integer with the .5 rounding constant and returned to the variable &RESULT. The format of the output value is D15.3.

```
-SET &POW = '3';
-SET &RESULT = EXP(&POW, 'D15.3') + 0.5;-TYPE E TO THE &POW POWER IS
APPROXIMATELY &RESULT
```

The output is:

```
E TO THE 3 POWER IS APPROXIMATELY 20
```

EXPN: Evaluating a Number in Scientific Notation

The EXPN function evaluates a numeric literal or Dialogue Manager variable expressed in scientific notation.

Evaluate a Number in Scientific Notation

```
EXPN(n.nn {E|D} {+|-} p)
```

where:

n.nn

Numeric

Is a numeric literal that consists of a whole number component, followed by a decimal point, followed by a fractional component.

E, D

Denotes scientific notation. E and D are interchangeable.

+, -

Indicates if **p** is positive or negative.

p

Integer

Is the power of 10 to which to raise **n.nn**.

> **Note:**
> EXPN does not use an output argument. The format of the result is floating-point double precision.

Evaluating a Number in Scientific Notation

EXPN evaluates 1.03E+2.

```
EXPN(1.03E+2)
```

The result is 103.

FMLCAP: Retrieving FML Hierarchy Captions

The FMLCAP function returns the caption value for each row in an FML hierarchy request. In order to retrieve caption values, the Master File must define an FML hierarchy and the request must use the GET CHILDREN, ADD, or WITH CHILDREN option to retrieve hierarchy data. If the FOR field in the request does not have a caption field defined, FMLCAP returns a blank string.

FMLCAP is supported for COMPUTE but is not recommended for use with DEFINE.

Retrieve Captions in an FML Request Using the FMLCAP Function

```
FMLCAP(fieldname|'format')
```

where:

fieldname

Is the name of the caption field.

'format'

Is the format of the caption field enclosed in single quotation marks.

Retrieving FML Hierarchy Captions Using FMLCAP

The following request retrieves and aggregates the FML hierarchy that starts with the parent value 2000. FMLCAP retrieves the captions, while the actual account numbers appear as the FOR values.

```
SET FORMULTIPLE = ON
TABLE FILE CENTSTMT
SUM ACTUAL_AMT
COMPUTE CAP1/A30= FMLCAP(GL_ACCOUNT_CAPTION);
FOR GL_ACCOUNT
2000 WITH CHILDREN 2 ADD
END
```

The output is:

```
                Actual  CAP1
                ------  ----
2000      313,611,852.  Gross Margin
  2100    187,087,470.  Sales Revenue
    2200   98,710,368.  Retail Sales
    2300   13,798,832.  Mail Order Sales
    2400   12,215,780.  Internet Sales
  2500    100,885,159.  Cost Of Goods Sold
    2600   54,877,250.  Variable Material Costs
    2700    6,176,900.  Direct Labor
    2800    3,107,742.  Fixed Costs
```

FMLFOR: Retrieving FML Tag Values

FMLFOR retrieves the tag value associated with each row in an FML request. If the FML row was generated as a sum of data records using the OR phrase, FMLFOR returns the first value specified in the list. If the OR phrase was generated by an FML Hierarchy ADD command, FMLFOR returns the tag value associated with the parent specified in the ADD command.

The FMLFOR function is supported for COMPUTE but not for DEFINE. Attempts to use it in a DEFINE result in blank values.

Retrieve FML Tag Values

```
FMLFOR(output)
```

where:

output

Is name of the field that will contain the result, or the format of the output value enclosed in single quotation marks.

Retrieving FML Tag Values With FMLFOR

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLFOR('A8');
FOR ACCOUNT
1010                  OVER
1020                  OVER
1030                  OVER
BAR                   OVER
1030 OR 1020 OR 1010
END
```

The output is:

```
      AMOUNT  RETURNEDFOR
      ------  -----------
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  --------
1010  21,239  1030
```

FMLINFO: Returning FOR Values

The FMLINFO function returns the FOR value associated with each row in an FML report. With FMLINFO, you can use the appropriate FOR value in a COMPUTE command to do drill-downs and sign changes for each row in the report, even when the row is a summary row created using an OR list or a Financial Modeling Language (FML) Hierarchy ADD command.

Note: You can use the SET parameter FORMULTIPLE=ON to enable an incoming record to be used on more than one line in an FML report.

Retain FOR Values in an FML Request

```
FMLINFO('FORVALUE', output)
```

where:

'FORVALUE'

Alphanumeric

Returns the FOR value associated with each row in an FML report. If the FML row was generated as a sum of data records using the OR phrase, FMLINFO returns the first FOR value specified in the list of values. If the OR phrase was generated by an FML Hierarchy ADD command, FMLINFO returns the FOR value associated with the parent specified in the ADD command.

output

Alphanumeric

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Retrieving FOR Values for FML Hierarchy Rows

The following request creates a field called PRINT_AMT that is the negative of the NAT_ AMOUNT field for account numbers less than 2500 in the CENTSYSF data source. The CENTGL data source contains the hierarchy information for CENTSYSF. Therefore, CENTGL is joined to CENTSYSF for the request:

```
SET FORMULTIPLE = ON
JOIN SYS_ACCOUNT IN CENTGL TO ALL SYS_ACCOUNT IN CENTSYSF
TABLE FILE CENTGL
SUM NAT_AMOUNT/D10 IN 30
COMPUTE PRINT_AMT/D10 = IF FMLINFO('FORVALUE','A7') LT '2500'
     THEN 0-NAT_AMOUNT ELSE NAT_AMOUNT;
COMPUTE FORV/A4 = FMLINFO('FORVALUE', 'A4');
COMPUTE ACTION/A9  = IF FORV LT '2500'
      THEN 'CHANGED' ELSE 'UNCHANGED';
FOR GL_ACCOUNT
```

```
2000 WITH CHILDREN 2 ADD AS CAPTION
END
```

Note:
The parent value specified in the WITH CHILDREN ADD command (2000) is returned for the first row on the report. Each subsequent row is also a consolidated subsection of the hierarchy with a parent value that is returned by FMLINFO:

```
                                  Month
                                 Actual     PRINT_AMT  FORV
ACTION
                                 ------     ---------  ----  -
-----
Gross Margin                -25,639,223    25,639,223  2000
CHANGED
  Sales Revenue             -62,362,490    62,362,490  2100
CHANGED
    Retail Sales            -49,355,184    49,355,184  2200
CHANGED
    Mail Order Sales         -6,899,416     6,899,416  2300
CHANGED
    Internet Sales           -6,107,890     6,107,890  2400
CHANGED
  Cost Of Goods Sold         36,723,267    36,723,267  2500
UNCHANGED
    Variable Material Costs  27,438,625    27,438,625  2600
UNCHANGED
    Direct Labor              6,176,900     6,176,900  2700
UNCHANGED
    Fixed Costs               3,107,742     3,107,742  2800
UNCHANGED
```

Using FMLINFO With an OR Phrase

The FOR value printed for the summary line is 1010, but FMLINFO returns the first value specified in the OR list, 1030:

```
SET FORMULTIPLE = ON
TABLE FILE LEDGER
```

```
SUM AMOUNT
COMPUTE RETURNEDFOR/A8 = FMLINFO('FORVALUE','A8');
FOR ACCOUNT
1010                  OVER
1020                  OVER
1030                  OVER
BAR                   OVER
1030 OR 1020 OR 1010
END
```

The output is:

	AMOUNT	RETURNEDFOR
1010 1020 1030	8,784 4,494 7,961	1010 1020 1030
	------	---------
1010	21,239	1030

FMLLIST: Returning an FML Tag List

FMLLIST returns a string containing the complete tag list for each row in an FML request. If a row has a single tag value, that value is returned.

The FMLLIST function is supported for COMPUTE but not for DEFINE. Attempts to use it in a DEFINE result in blank values.

Retrieve an FML Tag List

```
FMLLIST('A4096V')
```

where:

'A4096V'

Is the required argument.

Retrieving an FML Tag List With FMLLIST

```
SET FORMULTIPLE=ON
TABLE FILE LEDGER
HEADING
"TEST OF FMLLIST"
" "
SUM AMOUNT
COMPUTE LIST1/A36 = FMLLIST('A4096V');
FOR ACCOUNT
'1010'                 OVER
'1020'                 OVER
'1030'                 OVER
BAR                    OVER
'1030' OR '1020' OR '1010'
END
```

The output is:

```
TEST OF FMLLIST
      AMOUNT  LIST1
      ------  -----
1010   8,784  1010
1020   4,494  1020
1030   7,961  1030
      ------  ------------------------------------
1010  21,239  1010 OR 1020 OR 1030
```

INT: Finding the Greatest Integer

The INT function returns the integer component of a number.

Find the Greatest Integer

```
INT(in_value)
```

where:

in_value

Numeric

Is the value for which the integer component is returned, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation.

Finding the Greatest Integer

INT finds the greatest integer in the DED_AMT field and stores it in INT_DED_AMT:

```
TABLE FILE EMPLOYEE
SUM DED_AMT AND COMPUTE
INT_DED_AMT/I9 = INT(DED_AMT);BY LAST_NAME BY FIRST_NAME
WHERE (DEPARTMENT EQ 'MIS') AND (PAY_DATE EQ 820730);
END
```

The output is:

```
LAST_NAME         FIRST_NAME           DED_AMT  INT_DED_AMT
---------         ----------           -------  -----------
BLACKWOOD         ROSEMARIE          $1,261.40         1261
CROSS             BARBARA            $1,668.69         1668
GREENSPAN         MARY                 $127.50          127
JONES             DIANE                $725.34          725
SMITH             MARY                 $334.10          334
```

LOG: Calculating the Natural Logarithm

The LOG function returns the natural logarithm of a number.

Calculate the Natural Logarithm

```
LOG(in_value)
```

where:

in_value

Numeric

Is the value for which the natural logarithm is calculated, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation. If *in_value* is less than or equal to 0, LOG returns 0.

Calculating the Natural Logarithm

LOG calculates the logarithm of the CURR_SAL field:

```
TABLE FILE EMPLOYEE
PRINT CURR_SAL AND COMPUTE
LOG_CURR_SAL/D12.2 = LOG(CURR_SAL);BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'PRODUCTION';
END
```

The output is:

```
LAST_NAME        FIRST_NAME          CURR_SAL     LOG_CURR_SAL
---------        ----------          --------     ------------
BANNING          JOHN              $29,700.00            10.30
IRVING           JOAN              $26,862.00            10.20
MCKNIGHT         ROGER             $16,100.00             9.69
ROMANS           ANTHONY           $21,120.00             9.96
```

```
SMITH            RICHARD           $9,500.00             9.16
STEVENS          ALFRED           $11,000.00             9.31
```

MAX and MIN: Finding the Maximum or Minimum Value

The MAX and MIN functions return the maximum or minimum value, respectively, from a list of values.

Find the Maximum or Minimum Value

```
{MAX|MIN}(value1, value2, ...)
```

where:

MAX

Returns the maximum value.

MIN

Returns the minimum value.

value1, value2

Numeric

Are the values for which the maximum or minimum value is returned, the name of a field that contains the values, or an expression that returns the values. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation.

Determining the Minimum Value

MIN returns either the value of the ED_HRS field or the constant 30, whichever is lower:

```
TABLE FILE EMPLOYEE
PRINT ED_HRS AND COMPUTE
MIN_EDHRS_30/D12.2 = MIN(ED_HRS, 30);BY LAST_NAME BY FIRST_NAME
WHERE DEPARTMENT EQ 'MIS';
END
```

The output is:

```
LAST_NAME         FIRST_NAME  ED_HRS     MIN_EDHRS_30
---------         ---------- ------      ------------
BLACKWOOD         ROSEMARIE   75.00             30.00
CROSS             BARBARA     45.00             30.00
GREENSPAN         MARY        25.00             25.00
JONES             DIANE       50.00             30.00
MCCOY             JOHN          .00               .00
SMITH             MARY        36.00             30.00
```

MIRR: Calculating the Modified Internal Return Rate

The MIRR function calculates the modified internal rate of return for a series of periodic cash flows.

Calculate the Modified Internal Rate of Return

```
TABLE FILE ...
{PRINT|SUM} field ...COMPUTE rrate/fmt = MIRR(cashflow, finrate,
reinvrate, output);
WITHIN {sort_field|TABLE}
```

where:

field ...

Are fields that appear in the report output.

rrate

Is the field that contains the calculated return rate.

fmt

Is the format of the return rate. The data type must be D.

cashflow

Is a numeric field. Each value represents either a payment (negative value) or income (positive value) for one period. The values must be in the correct sequence in order for the sequence of cash flows to be calculated correctly. The dates corresponding to each cash flow should be equally spaced and sorted in chronological order. The calculation requires at least one negative value and one positive value in the *cashflow* field. If the values are all positive or all negative, a zero result is returned.

finrate

Is a finance rate for negative cash flows. This value must be expressed as a non-negative decimal fraction between 0 and 1. It must be constant within each sort group for which a return rate is calculated, but it can change between sort groups.

reinvrate

Is the reinvestment rate for positive cash flows. This value must be expressed as a non-negative decimal fraction between 0 and 1. It must be constant within each sort group but can change between sort groups. It must be constant within each sort group for which a return rate is calculated, but it can change between sort groups.

output

Is the name of the field that contains the return rate, or its format enclosed in single quotation marks.

sort_field

Is a field that sorts the report output and groups it into subsets of rows on which the function can be calculated separately. To calculate the function using every row of the report output, use the WITHIN TABLE phrase. A WITHIN phrase is required.

Usage Notes for the MIRR Function

- This function is only supported in a COMPUTE command with the WITHIN phrase.
- The cash flow field must contain at least one negative value and one positive value.
- Dates must be equally spaced.
- Missing cash flows or dates are not supported.

Calculating the Modified Internal Rate of Return

The following request calculates modified internal return rates for categories of products. It assumes a finance charge of ten percent and a reinvestment rate of ten percent. The request is sorted by date so that the correct cash flows are calculated. The rate returned by the function is multiplied by 100 in order to express it as a percent rather than a decimal value. Note that the format includes the % character. This causes a percent symbol to display, but it does not calculate a percent.

In order to create one cash flow value per date, the values are summed. NEWDOLL is defined in order to create negative values in each category as required by the function:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END
TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN CATEGORY
  BY CATEGORY
  BY SDATE
  WHERE SYEAR EQ 97
END
```

A separate rate is calculated for each category because of the WITHIN CATEGORY phrase. A portion of the output is shown:

```
Category     SDATE           NEWDOLL      RRATE
--------     -----           -------      -----
```

```
Coffee        1997/01     -801,123.00     15.11%
              1997/02     -682,340.00     15.11%
              1997/03     -765,078.00     15.11%
              1997/04      691,274.00     15.11%
              1997/05      720,444.00     15.11%
              1997/06      742,457.00     15.11%
              1997/07      747,253.00     15.11%
              1997/08      655,896.00     15.11%
              1997/09      730,317.00     15.11%
              1997/10      724,412.00     15.11%
              1997/11      620,264.00     15.11%
              1997/12      762,328.00     15.11%
Food          1997/01     -672,727.00     16.24%
              1997/02     -699,073.00     16.24%
              1997/03     -642,802.00     16.24%
              1997/04      718,514.00     16.24%
              1997/05      660,740.00     16.24%
              1997/06      734,705.00     16.24%
              1997/07      760,586.00     16.24%
```

To calculate one modified internal return rate for all of the report data, use the WITHIN TABLE phrase. In this case, the data does not have to be sorted by CATEGORY:

```
DEFINE FILE GGSALES
 SDATE/YYM = DATE;
 SYEAR/Y = SDATE;
 NEWDOLL/D12.2 = IF DATE LT '19970401' THEN -1 * DOLLARS ELSE DOLLARS;
END

TABLE FILE GGSALES
  SUM NEWDOLL
  COMPUTE RRATE/D7.2% = MIRR(NEWDOLL, .1, .1, RRATE) * 100;
  WITHIN TABLE
  BY SDATE
  WHERE SYEAR EQ 97
END
```

The output is:

```
SDATE            NEWDOLL      RRATE
-----            -------      -----
1997/01   -1,864,129.00     15.92%
1997/02   -1,861,639.00     15.92%
1997/03   -1,874,439.00     15.92%
1997/04    1,829,838.00     15.92%
```

```
1997/05    1,899,494.00     15.92%
1997/06    1,932,630.00     15.92%
1997/07    2,005,402.00     15.92%
1997/08    1,838,863.00     15.92%
1997/09    1,893,944.00     15.92%
1997/10    1,933,705.00     15.92%
1997/11    1,865,982.00     15.92%
1997/12    2,053,923.00     15.92%
```

NORMSDST and NORMSINV: Calculating Normal Distributions

The NORMSDST and NORMSINV functions perform calculations on a standard normal distribution curve. NORMSDST calculates the percentage of data values that are less than or equal to a normalized value; NORMSINV is the inverse of NORMSDST, calculates the normalized value that forms the upper boundary of a percentile in a standard normal distribution curve.

NORMSDST: Calculating Standard Cumulative Normal Distribution

The NORMSDST function performs calculations on a standard normal distribution curve, calculating the percentage of data values that are less than or equal to a normalized value. A normalized value is a point on the X-axis of a standard normal distribution curve in standard deviations from the mean. This is useful for determining percentiles in normally distributed data.

The NORMSINV function is the inverse of NORMSDST. For information about NORMSINV, see NORMSDST and NORMSINV: Calculating Normal Distributions.

The results of NORMSDST are returned as double-precision and are accurate to 6 significant digits.

A standard normal distribution curve is a normal distribution that has a mean of 0 and a standard deviation of 1. The total area under this curve is 1. A point on the X-axis of the standard normal distribution is called a normalized value. Assuming that your data is normally distributed, you can convert a data point to a normalized value to find the percentage of scores that are less than or equal to the raw score.

You can convert a value (raw score) from your normally distributed data to the equivalent normalized value (z-score) as follows:

```
z = (raw_score - mean)/standard_deviation
```

To convert from a z-score back to a raw score, use the following formula:

```
raw_score = z * standard_deviation + mean
```

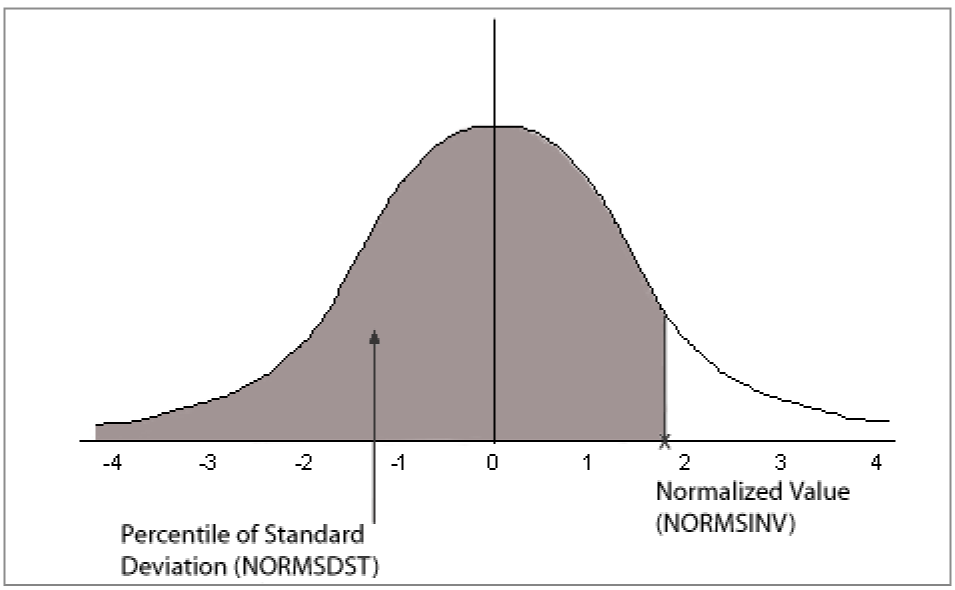
The mean of data points xi, where i is from 1 to n is:

$(\sum x_i)/n$

The standard deviation of data points xi, where i is from 1 to n is:

$SQRT(\ ((\sum x_i{}^2 - (\sum x_i)^2/n)/(n - 1)))$

The following diagram illustrates the results of the NORMSDST and NORMSINV functions.



Characteristics of the Normal Distribution

Many common measurements are normally distributed. A plot of normally distributed data values approximates a bell-shaped curve. The two measures required to describe any normal distribution are the mean and the standard deviation:

- The mean is the point at the center of the curve.
- The standard deviation describes the spread of the curve. It is the distance from the mean to the point of inflection (where the curve changes direction).

Calculate the Cumulative Standard Normal Distribution Function

```
NORMSDST(value, 'D8');
```

where:

value

Is a normalized value.

D8

Is the required format for the result. The value returned by the function is double-precision. You can assign it to a field with any valid numeric format.

Using the NORMSDST Function

NORMSDST calculates the Z value and finds its percentile:

```
DEFINE FILE GGPRODS
-* CONVERT SIZE FIELD TO DOUBLE PRECISION
X/D12.5 = SIZE;
END
TABLE FILE GGPRODS
SUM X NOPRINT CNT.X NOPRINT
-* CALCULATE MEAN AND STANDARD DEVIATION
```

```
COMPUTE NUM/D12.5 = CNT.X; NOPRINT
COMPUTE MEAN/D12.5 = AVE.X; NOPRINT
COMPUTE VARIANCE/D12.5 = ((NUM*ASQ.X) - (X*X/NUM))/(NUM-1); NOPRINT
COMPUTE STDEV/D12.5 = SQRT(VARIANCE); NOPRINT
PRINT SIZE X NOPRINT
-* COMPUTE NORMALIZED VALUES AND USE AS INPUT TO NORMSDST FUNCTION
COMPUTE Z/D12.5 = (X - MEAN)/STDEV;
COMPUTE NORMSD/D12.5 = NORMSDST(Z, 'D8');
BY PRODUCT_ID NOPRINT
END
```

The output is:

```
Size              Z         NORMSD
----              -         ------
  16        -.07298         .47091
  12        -.80273         .21106
  12        -.80273         .21106
  20         .65678         .74434
  24        1.38654         .91721
  20         .65678         .74434
  24        1.38654         .91721
  16        -.07298         .47091
  12        -.80273         .21106
   8       -1.53249         .06270
```

NORMSINV: Calculating Inverse Cumulative Normal Distribution

The NORMSINV function performs calculations on a standard normal distribution curve, finding the normalized value that forms the upper boundary of a percentile in a standard normal distribution curve. This is the inverse of NORMSDST. For information about NORMSDST, see NORMSDST and NORMSINV: Calculating Normal Distributions.

The results of NORMSINV are returned as double-precision and are accurate to 6 significant digits.

Calculate the Inverse Cumulative Standard Normal Distribution Function

```
NORMSINV(value, 'D8');
```

where:

value

Is a number between 0 and 1 (which represents a percentile in a standard normal distribution).

D8

Is the required format for the result. The value returned by the function is double-precision. You can assign it to a field with any valid numeric format.

Using the NORMSINV Function

NORMSDST finds the percentile for the Z field. NORMSINV then returns this percentile to a normalized value:

```
DEFINE FILE GGPRODS
-* CONVERT SIZE FIELD TO DOUBLE PRECISION
X/D12.5 = SIZE;
END
TABLE FILE GGPRODS
SUM X NOPRINT CNT.X NOPRINT
-* CALCULATE MEAN AND STANDARD DEVIATION
COMPUTE NUM/D12.5 = CNT.X; NOPRINT
COMPUTE MEAN/D12.5 = AVE.X; NOPRINT
COMPUTE VARIANCE/D12.5 = ((NUM*ASQ.X) - (X*X/NUM))/(NUM-1); NOPRINT
COMPUTE STDEV/D12.5 = SQRT(VARIANCE); NOPRINT
PRINT SIZE X NOPRINT
-* COMPUTE NORMALIZED VALUES AND USE AS INPUT TO NORMSDST FUNCTION
-* THEN USE RETURNED VALUES AS INPUT TO NORMSINV FUNCTION
-* AND CONVERT BACK TO DATA VALUES
COMPUTE Z/D12.5 = (X - MEAN)/STDEV;
COMPUTE NORMSD/D12.5 = NORMSDST(Z, 'D8');
COMPUTE NORMSI/D12.5 = NORMSINV(NORMSD, 'D8');
COMPUTE DSIZE/D12 = NORMSI * STDEV + MEAN;
```

```
BY PRODUCT_ID NOPRINT
END
```

The output shows that NORMSINV is the inverse of NORMSDST and returns the original values:

```
Size            Z        NORMSD      NORMSI       DSIZE
----            -        ------      ------       -----
  16      -.07298        .47091     -.07298          16
  12      -.80273        .21106     -.80273          12
  12      -.80273        .21106     -.80273          12
  20       .65678        .74434      .65678          20
  24      1.38654        .91721     1.38654          24
  20       .65678        .74434      .65678          20
  24      1.38654        .91721     1.38654          24
  16      -.07298        .47091     -.07298          16
  12      -.80273        .21106     -.80273          12
   8     -1.53249        .06270    -1.53249           8
```

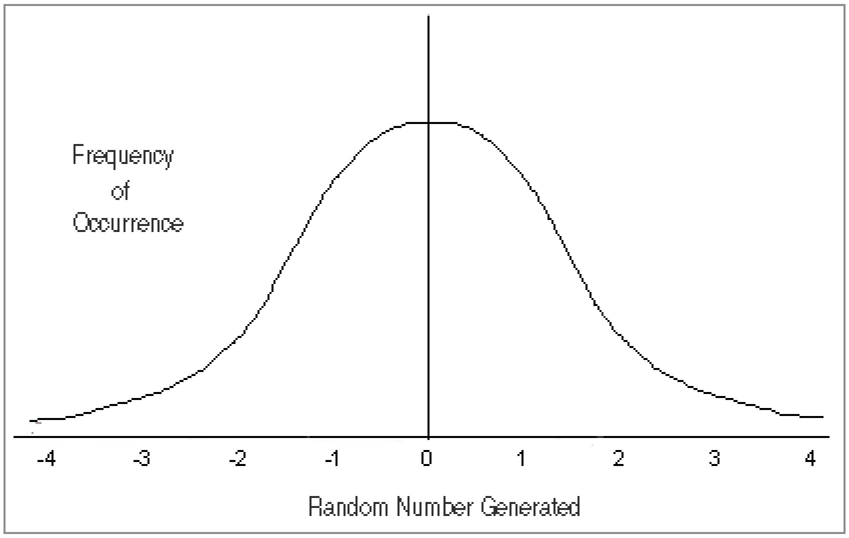
PRDNOR and PRDUNI: Generating Reproducible Random Numbers

The PRDNOR and PRDUNI functions generate reproducible random numbers:

- PRDNOR generates reproducible double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

  If PRDNOR generates a large set of numbers, they have the following properties:

  - The numbers lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, meaning that there are more numbers closer to 0 than farther away.



  - The average of the numbers is close to 0.
  - The numbers can be any size, but most are between 3 and -3.
- PRDUNI generates reproducible double-precision random numbers uniformly distributed between 0 and 1 (that is, any random number it generates has an equal probability of being anywhere between 0 and 1).

In z/OS, the numbers do not reproduce.

Generate Reproducible Random Numbers

```
{PRDNOR|PRDUNI}(seed, output)
```

where:

PRDNOR

Generates reproducible double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

PRDUNI

Generates reproducible double-precision random numbers uniformly distributed between 0 and 1.

seed

Numeric

Is the seed or the field that contains the seed, up to 9 digits. The seed is truncated to an integer.

On z/OS, the numbers do not reproduce.

output

Double-precision

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Generating Reproducible Random Numbers

PRDNOR assigns random numbers and stores them in RAND. These values are then used to randomly pick five employee records identified by the values in the LAST NAME and FIRST NAME fields. The seed is 40. To produce a different set of numbers, change the seed.

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = PRDNOR(40, RAND);END
```

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

The output is:

```
RAND  LAST_NAME        FIRST_NAME
----  ---------        ----------
1.38  STEVENS          ALFRED
1.12  MCCOY            JOHN
 .55  SMITH            RICHARD
 .21  JONES            DIANE
 .01  IRVING           JOAN
```

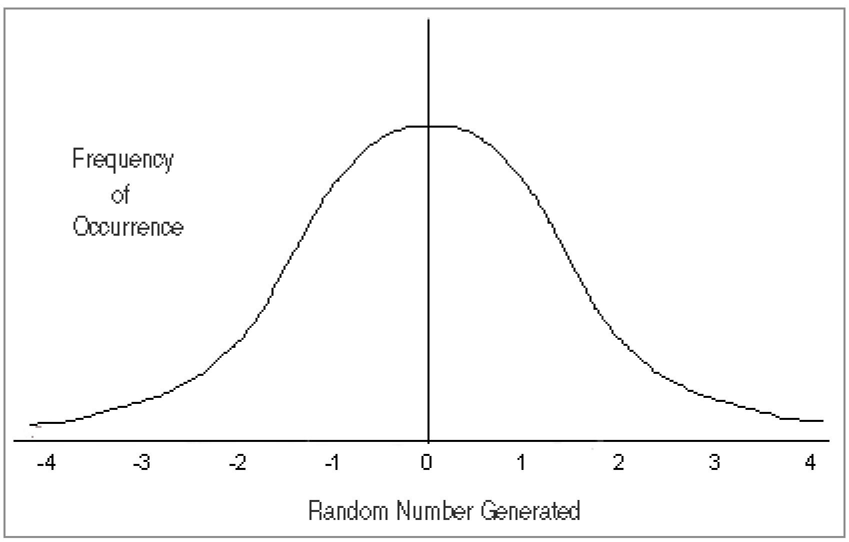
RDNORM and RDUNIF: Generating Random Numbers

The RDNORM and RDUNIF functions generate random numbers:

- RDNORM generates double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

  If RDNORM generates a large set of numbers (between 1 and 32768), they have the following properties:

  - The numbers lie roughly on a bell curve, as shown in the following figure. The bell curve is highest at the 0 mark, meaning that there are more numbers closer to 0 than farther away.

  - The average of the numbers is close to 0.
  - The numbers can be any size, but most are between 3 and -3.

- RDUNIF generates double-precision random numbers uniformly distributed between 0 and 1 (that is, any random number it generates has an equal probability of being anywhere between 0 and 1).

Generate Random Numbers

```
{RDNORM|RDUNIF}(output)
```

where:

RDNORM

Generates double-precision random numbers normally distributed with an arithmetic mean of 0 and a standard deviation of 1.

RDUNIF

Generates double-precision random numbers uniformly distributed between 0 and 1.

output

Double-precision

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks.

Generating Random Numbers

RDNORM assigns random numbers and stores them in RAND. These numbers are then used to randomly choose five employee records identified by the values in the LAST NAME and FIRST NAME fields.

```
DEFINE FILE EMPLOYEE
RAND/D12.2 WITH LAST_NAME = RDNORM(RAND);END
TABLE FILE EMPLOYEE
PRINT LAST_NAME AND FIRST_NAME
BY HIGHEST 5 RAND
END
```

The request produces output similar to the following:

```
RAND  LAST_NAME        FIRST_NAME
----  ---------        ----------
 .65  CROSS            BARBARA
 .20  BANNING          JOHN
```

```
 .19  IRVING           JOAN
 .00  BLACKWOOD        ROSEMARIE
-.14  GREENSPAN        MARY
```

SQRT: Calculating the Square Root

The SQRT function calculates the square root of a number.

Calculate the Square Root

```
SQRT(in_value)
```

where:

in_value

Numeric

Is the value for which the square root is calculated, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation. If you supply a negative number, the result is zero.

Calculating the Square Root

SQRT calculates the square root of LISTPR:

```
TABLE FILE MOVIES
PRINT LISTPR AND COMPUTE
SQRT_LISTPR/D12.2 = SQRT(LISTPR);BY TITLE
WHERE CATEGORY EQ 'MUSICALS';
END
```

The output is:

```
TITLE                                LISTPR      SQRT_LISTPR
-----                                ------      -----------
ALL THAT JAZZ                         19.98             4.47
CABARET                               19.98             4.47
CHORUS LINE, A                        14.98             3.87
FIDDLER ON THE ROOF                   29.95             5.47
```

XIRR: Calculating the Modified Internal Return Rate (Periodic or Non-Periodic)

The XIRR function calculates the internal rate of return for a series of cash flows that can be periodic or non-periodic.

Calculate the Internal Rate of Return

```
TABLE FILE ...
{PRINT|SUM} field ...
COMPUTE rrate/fmt = XIRR (cashflow, dates,guess, maxiterations, output);
WITHIN {sort_field|TABLE}
```

where:

field ...

Are fields that appear in the report output.

rrate

Is the field that contains the calculated return rate.

fmt

Is the format of the return rate. The data type must be D.

cashflow

Is a numeric field. Each value of this field represents either a payment (negative value) or income (positive value) for one period. The values must be in the correct sequence in order for the sequence of cash flows to be calculated correctly. The dates corresponding to each cash flow should be equally spaced and sorted in chronological order. The calculation requires at least one negative value and one positive value in the *cashflow* field. If the values are all positive or all negative, a zero result is returned.

dates

Is a date field containing the cash flow dates. The dates must be full component dates with year, month, and day components. Dates cannot be stored in fields with format A, I, or P. They must be stored in date fields (for example, format YMD, not AYMD). There must be the same number of dates as there are cash flow values. The number of dates must be the same as the number of cash flows.

guess

Is an (optional) initial estimate of the expected return rate expressed as a decimal. The default value is .1 (10%). To accept the default, supply the value 0 (zero) for this argument.

maxiterations

Is an (optional) number specifying the maximum number of iterations that can be used to resolve the rate using Newton's method. 50 is the default value. To accept the default, supply the value 0 (zero) for this argument. The rate is considered to be resolved when successive iterations do not differ by more than 0.0000003. If this level of accuracy is achieved within the maximum number of iterations, the calculation stops at that point. If it is not achieved after reaching the maximum number of iterations, the calculation stops and the value calculated by the last iteration is returned.

output

D

Is the name of the field that contains the return rate, or its format enclosed in single quotation marks.

sort_field

Is a field that sorts the report output and groups it into subsets of rows on which the function can be calculated separately. To calculate the function using every row of the report output, use the WITHIN TABLE phrase. A WITHIN phrase is required.

Usage Notes for the XIRR Function

- This function is only supported in a COMPUTE command with the WITHIN phrase.
- The cash flow field must contain at least one negative value and one positive value.
- Dates cannot be stored in fields with format A, I, or P. They must be stored in date fields (for example, format YMD, not AYMD).
- Cash flows or dates with missing values are not supported.

Calculating the Internal Rate of Return

The following request creates a FOCUS data source with cash flows and dates and calculates the internal return rate.

The Master File for the data source is:

```
FILENAME=XIRR01,SUFFIX=FOC
SEGNAME=SEG1,SEGTYPE=S1
FIELDNAME=DUMMY,FORMAT=A2,$
FIELDNAME=DATES,FORMAT=YYMD,$
FIELDNAME=CASHFL,FORMAT=D12.4,$
END
```

The procedure to create the data source is:

```
CREATE FILE XIRR01
MODIFY FILE XIRR01
FREEFORM DUMMY DATES CASHFL
DATA
AA,19980101,-10000. ,$
BB,19980301,2750.   ,$
CC,19981030,4250.   ,$
DD,19990215,3250.   ,$
EE,19990401,2750.   ,$
END
```

The request is sorted by date so that the correct cash flows can be calculated. The rate returned by the function is multiplied by 100 in order to express it as a percent rather than a decimal value. Note that the format includes the % character. This causes a percent symbol to display, but it does not calculate a percent:

```
TABLE FILE XIRR01
PRINT CASHFL
COMPUTE RATEX/D12.2%=XIRR(CASHFL, DATES, 0., 0., RATEX) * 100;
WITHIN TABLE
BY DATES
END
```

One rate is calculated for the entire report because of the WITHIN TABLE phrase:

```
DATES                CASHFL       RATEX
----                 ------       -----
1998/01/01     -10,000.0000      37.49%
1998/03/01       2,750.0000      37.49%
1998/10/30       4,250.0000      37.49%
1999/02/15       3,250.0000      37.49%
1999/04/01       2,750.0000      37.49%
```

Simplified Statistical Functions

Simplified statistical functions can be called in a COMPUTE command to perform statistical calculations on the internal matrix that is generated during TABLE request processing. The STDDEV and CORRELATION functions can also be called as a verb object in a display command. Prior to calling a statistical function, you need to establish the size of the partition on which these functions will operate, if the request contains sort fields.

Note: It is recommended that all numbers and fields used as parameters to these functions be double-precision.

Specify the Partition Size for Simplified Statistical Functions

```
SET PARTITION_ON = {FIRST|PENULTIMATE|TABLE}
```

where:

FIRST

Uses the first (also called the major) sort field in the request to partition the values.

PENULTIMATE

Uses the next to last sort field where the COMPUTE is evaluated to partition the values. This is the default value.

TABLE

Uses the entire internal matrix to calculate the statistical function.

CORRELATION: Calculating the Degree of Correlation Between Two Sets of Data

The CORRELATION function calculates the correlation coefficient between two numeric fields. The function returns a numeric value between zero (-1.0) and 1.0.

Calculate the Correlation Coefficient Between Two Fields

```
CORRELATION(field1, field2)
```

where:

field1

Numeric

Is the first set of data for the correlation.

field2

Numeric

Is the second set of data for the correlation.

> **Note:** Arguments for CORRELATION cannot be prefixed fields. If you need to work with fields that have a prefix operator applied, apply the prefix operators to the fields in COMPUTE commands and save the results in a HOLD file. Then, run the correlation against the HOLD file.

Calculating a Correlation

The following request calculates the correlation between the DOLLARS and BUDDOLLARS fields converted to double precision.

```
DEFINE FILE ibisamp/ggsales
DOLLARS/D12.2 = DOLLARS;
BUDDOLLARS/D12.2 = BUDDOLLARS;
END
TABLE FILE ibisamp/ggsales
SUM DOLLARS BUDDOLLARS
CORRELATION(DOLLARS, BUDDOLLARS)
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

DOLLARS	BUDDOLLARS	CORRELATION DOLLARS BUDDOLLARS
46,156,290.00	46,220,778.00	.895691073

KMEANS_CLUSTER: Partitioning Observations Into Clusters Based on the Nearest Mean Value

The KMEANS_CLUSTER function partitions observations into a specified number of clusters based on the nearest mean value. The function returns the cluster number assigned to the field value passed as a parameter.

Note: If there are not enough points to create the number of clusters requested, the value -10 is returned for any cluster that cannot be created.

Partition Observations Into Clusters Based on the Nearest Mean Value

```
KMEANS_CLUSTER(number, percent, iterations, tolerance,
        [prefix1.]field1[, [prefix1.]field2 ...])
```

where:

number

Integer

Is number of clusters to extract.

percent

Numeric

Is the percent of training set size (the percent of the total data to use in the calculations). The default value is AUTO, which uses the internal default percent.

iterations

Integer

Is the maximum number of times to recalculate using the means previously generated. The default value is AUTO, which uses the internal default number of iterations.

tolerance

Numeric

Is a weight value between zero (0) and 1.0. The value AUTO uses the internal default tolerance.

prefix1, prefix2

Defines an optional aggregation operator to apply to the field before using it in the calculation. Valid operators are:

- **SUM.** which calculates the sum of the field values. SUM is the default value.
- **CNT.** which calculates a count of the field values.
- **AVE.** which calculates the average of the field values.

- **MIN.** which calculates the minimum of the field values.
- **MAX.** which calculates the maximum of the field values.
- **FST.** which retrieves the first value of the field.
- **LST.** which retrieves the last value of the field.

> **Note:** The operators PCT., RPCT., TOT., MDN., MDE., RNK., and DST. are not supported.

field1

Numeric

Is the set of data to be analyzed.

field2

Numeric

Is an optional set of data to be analyzed.

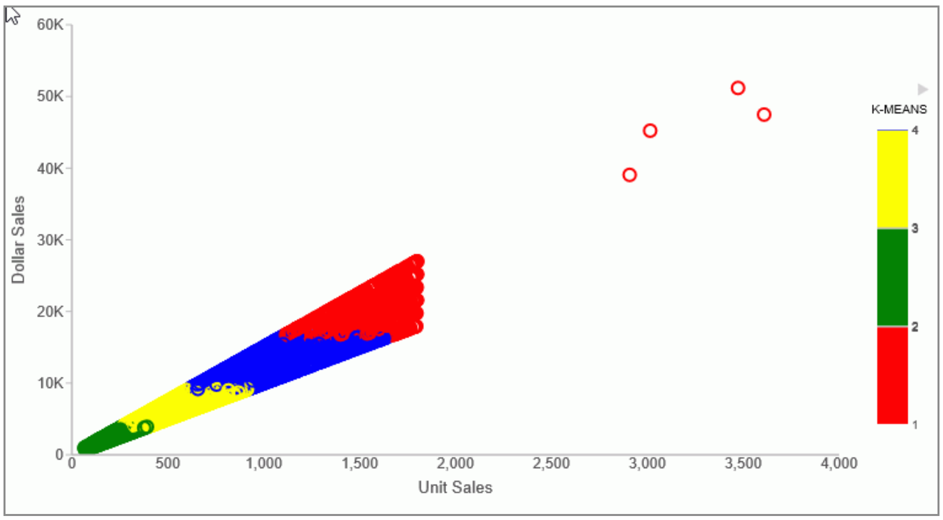
Partitioning Data Values Into Clusters

The following request partitions the DOLLARS field values into four clusters and displays the result as a scatter chart in which the color represents the cluster. The request uses the default values for the percent, iterations, and tolerance parameters by passing them as the value 0 (zero).

```
SET PARTITION_ON = PENULTIMATE
GRAPH FILE GGSALES
PRINT UNITS DOLLARS
COMPUTE KMEAN1/D20.2 TITLE 'K-MEANS'=  KMEANS_CLUSTER(4, AUTO, AUTO,
AUTO, DOLLARS);
ON GRAPH SET LOOKGRAPH SCATTER
ON GRAPH PCHOLD FORMAT JSCHART
ON GRAPH SET STYLE *
INCLUDE=Warm.sty,$
type = data, column = N2, bucket=y-axis,$
type=data, column= N1, bucket=x-axis,$
type=data, column=N3, bucket=color,$
GRID=OFF,$
*GRAPH_JS_FINAL
```

```
colorScale: {
                colorMode: 'discrete',
                colorBands: [{start: 1, stop: 1.99, color: 'red'}, {start: 2, stop:
2.99, color: 'green'},
                {start: 3, stop: 3.99, color: 'yellow'}, {start: 3.99,
stop: 4, color: 'blue'} ]
        }
*END
ENDSTYLE
END
```

The output is shown in the following image.



MULTIREGRESS: Creating a Multivariate Linear Regression Column

MULTIREGRESS derives a linear equation that best fits a set of numeric data points, and uses this equation to create a new column in the report output. The equation can be based on one or more independent variables.

The equation generated is of the following form, where y is the dependent variable and x1, x2, and x3 are the independent variables.

```
y = a1*x1 [+ a2*x2 [+ a3*x3] ...] + b
```

When there is one independent variable, the equation represents a straight line. When there are two independent variables, the equation represents a plane, and with three independent variables, it represents a hyperplane. You should use this technique when you have reason to believe that the dependent variable can be approximated by a linear combination of the independent variables.

Create a Multivariate Linear Regression Column

```
MULTIREGRESS(input_field1, [input_field2, ...])
```

where:

input_field1, input_field2 ...

Are any number of field names to be used as the independent variables. They should be independent of each other. If an input field is non-numeric, it will be categorized to transform it to numeric values that can be used in the linear regression calculation.

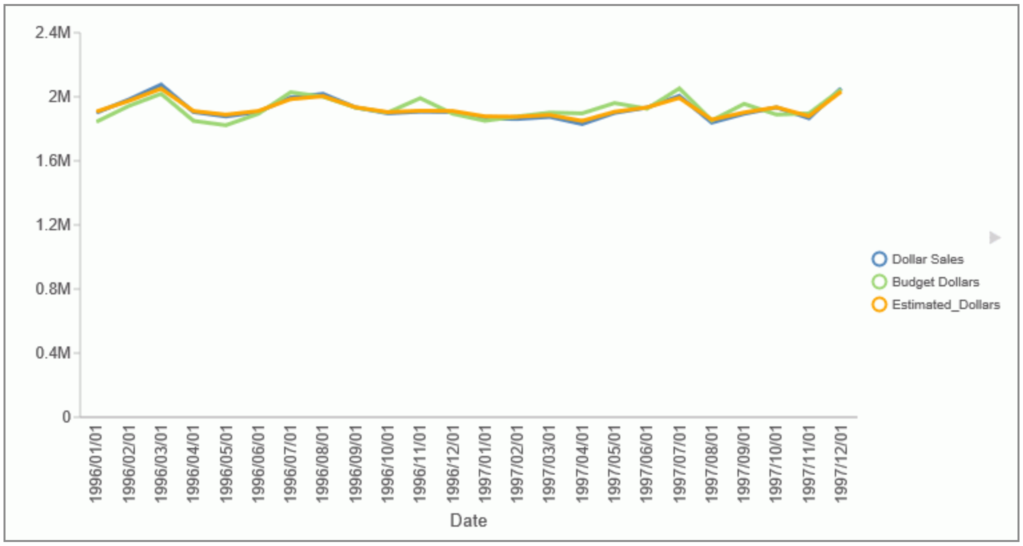
Creating a Multivariate Linear Regression Column

The following request uses the DOLLARS and BUDDOLLARS fields to generate a regression column named Estimated_Dollars.

```
GRAPH FILE GGSALES
SUM BUDUNITS UNITS BUDDOLLARS DOLLARS
COMPUTE Estimated_Dollars/F8 = MULTIREGRESS(DOLLARS, BUDDOLLARS);
BY DATE
ON GRAPH SET LOOKGRAPH LINE
ON GRAPH PCHOLD FORMAT JSCHART
ON GRAPH SET STYLE *
INCLUDE=Warm.sty,$
type=data, column = n1, bucket = x-axis,$
type=data, column= dollars, bucket=y-axis,$
type=data, column= buddollars, bucket=y-axis,$
type=data, column= Estimated_Dollars, bucket=y-axis,$
*GRAPH_JS
"series":[
{"series":2, "color":"orange"}]
*END
```

```
ENDSTYLE
END
```

The output is shown in the following image. The orange line represents the regression equation.



OUTLIER: Identifying Outliers in Numeric Data

The 1.5 * IQR (Inner Quartile Range) rule is a common way to identify outliers in data. This rule defines an outlier as a value that is above or below 1.5 times the inner quartile range in the data. The inner quartile range is based on sorting the data values, dividing it into equal quarters, and calculating the range of values between the first quartile (the value one quarter of the way through the sorted data) and third quartile (the value three quarters of the way through the sorted data). The value that is 1.5 times below the inner quartile range is called the *lower fence*, and the value that is 1.5 times above the inner quartile range is called the *upper fence*.

OUTLIER is not supported in a DEFINE expression. It can be used in a COMPUTE expression or a WHERE, WHERE TOTAL, or WHERE_GROUPED phrase.

Given a numeric field as input, OUTLIER returns one of the following values for each value of the field, using the 1.5 * IQR rule:

- **0 (zero).** The value is not an outlier.

- **-1.** The value is below the lower fence.
- **1.** The value is above the upper fence.

Identify Outliers in Numeric Data

```
OUTLIER(input_field)
```

where:

input_field

Numeric

Is the numeric field to be analyzed.

Identifying Outliers

The following request defines the SALES field to have different values depending on the store code, and uses OUTLIER to determine whether each field value is an outlier.

```
DEFINE FILE GGSALES
SALES/D12 = IF ((CATEGORY EQ 'Coffee') AND (STCD EQ 'R1019')) THEN 19000
  ELSE IF ((CATEGORY EQ 'Coffee') AND (STCD EQ 'R1020')) THEN 20000
  ELSE IF ((CATEGORY EQ 'Coffee') AND (STCD EQ 'R1040')) THEN 7000
  ELSE DOLLARS;
END
TABLE FILE GGSALES
SUM SALES
COMPUTE OUT1/I3 = OUTLIER(SALES);
BY CATEGORY
BY STCD
WHERE CATEGORY EQ 'Coffee'
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image. Values above 2 million are above the upper fence, values below 1 million are below the lower fence, and other values are not outliers:

Category	Store ID	SALES	OUT1
Coffee	R1019	2,280,000	1
	R1020	2,400,000	1
	R1040	840,000	-1
	R1041	1,576,915	0
	R1044	1,340,437	0
	R1088	1,375,040	0
	R1100	1,364,420	0
	R1109	1,459,160	0
	R1200	1,463,453	0
	R1244	1,553,962	0
	R1248	1,535,631	0
	R1250	1,386,124	0

STDDEV: Calculating the Standard Deviation for a Set of Data Values

The STDDEV function returns a numeric value that represents the amount of dispersion in the data. The set of data can be specified as the entire population or a sample. The standard deviation is the square root of the variance, which is a measure of how observations deviate from their expected value (mean). If specified as a population, the divisor in the standard deviation calculation (also called degrees of freedom) will be the total number of data points, N. If specified as a sample, the divisor will be N-1.

If x_i is an observation, N is the number of observations, and μ is the mean of all of the observations, the formula for calculating the standard deviation for a population is:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_1-\mu)^2}$$

To calculate the standard deviation for a sample, the mean is calculated using the sample observations, and the divisor is N-1 instead of N.

Calculate the Standard Deviation in a Set of Data

```
STDDEV(field, sampling)
```

where:

field

Numeric

Is the set of observations for the standard deviation calculation.

sampling

Keyword

Indicates the origin of the data set. Can be one of the following values.

- **P** Entire population.
- **S** Sample of population.

Note: Arguments for STDDEV cannot be prefixed fields. If you need to work with fields that have a prefix operator applied, apply the prefix operators to the fields in COMPUTE commands and save the results in a HOLD file. Then, run the standard deviation against the HOLD file.

Calculating a Standard Deviation

The following request calculates the standard deviation of the DOLLARS field converted to double precision.

```
DEFINE FILE ibisamp/ggsales
DOLLARS/D12.2 = DOLLARS;
END
TABLE FILE ibisamp/ggsales
SUM DOLLARS STDDEV(DOLLARS,S)
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
```

```
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

DOLLARS	STDS DOLLARS
46,156,290.00	6,157.711080272

Simplified System Functions

Simplified system functions have streamlined parameter lists, similar to those used by SQL functions. In some cases, these simplified functions provide slightly different functionality than previous versions of similar functions.

The simplified functions do not have an output argument. Each function returns a value that has a specific data type.

When used in a request against a relational data source, these functions are optimized (passed to the RDBMS for processing).

EDAPRINT: Inserting a Custom Message in the EDAPRINT Log File

The EDAPRINT function enables you to allocate a sequential file to DDNAME EDAPRINT, add a text message into it, and assign it a message type.

Insert a Message in the EDAPRINT Log File

```
EDAPRINT(message_type, 'message')
```

where:

message_type

Keyword

Can be one of the following message types.

- **I.** Informational message.
- **W.** Warning message.

- **E.** Error message.

'message'

Is the message to insert, enclosed in single quotation marks.

Inserting a Custom Message in the EDAPRINT Log File

The following procedure inserts three messages in the EDAPRINT log file.

```
DYNAM ALLOC DD EDAPRINT DA USER1.EDAPRINT.LOG SHR REU
```

```
-SET &I = EDAPRINT(I, 'This is a test informational message');
-SET &W = EDAPRINT(W, 'This is a test warning message');
-SET &E = EDAPRINT(E, 'This is a test error message');
```

The following is the contents of the file allocated to DDNAME EDAPRINT after running the request.

```
00001 04/04/2019 10:52:15.483 I This is a test informational message
00002 04/04/2019 10:52:15.490 W This is a test warning message
00003 04/04/2019 10:52:15.518 E This is a test error message
```

ENCRYPT: Encrypting a Password

The ENCRYPT function encrypts an alphanumeric input value using the encryption algorithm configured in FOCUS. The result is returned as a variable length alphanumeric.

Encrypt a Password

```
ENCRYPT(password)
```

where:

password

Fixed length alphanumeric

Is the value to be encrypted.

Encrypting a Password

The following request encrypts the value *guestpassword* using the encryption algorithm configured in FOCUS.

```
-SET &P1 = ENCRYPT('guestpassword');
-TYPE &P1
```

The returned encrypted value is {AES}963AFA754E1763ABE697E8C5E764115E.

GETENV: Retrieving the Value of an Environment Variable

The GETENV function takes the name of an environment variable and returns its value as a variable length alphanumeric value.

Retrieve the Value of an Environment Variable

```
GETENV(var_name)
```

where:

var_name

fixed length alphanumeric

Is the name of the environment variable whose value is being retrieved.

PUTENV: Assigning a Value to an Environment Variable

The PUTENV function assigns a value to an environment variable. The function returns an integer return code whose value is 1 (one) if the assignment is not successful or 0 (zero) if it is successful.

Assign a Value to an Environment Variable

```
PUTENV(var_name, var_value)
```

where:

var_name

Fixed length alphanumeric

Is the name of the environment variable to be set.

var_value

Alphanumeric

Is the value you want to assign to the variable.

Assigning a Value to the UNIX PS1 Variable

The following request assigns the value *FOCUS/Shell:* to the UNIX PS1 variable.

```
-SET &P1 = PUTENV('PS1','FOCUS/Shell:');
```

This causes UNIX to display the following prompt when the user issues the UNIX shell command SH:

```
FOCUS/Shell:
```

The following request creates a variable named xxxx and sets it to the value **this is a test**. It then retrieves the value using GETENV.

```
-SET &XXXX=PUTENV(xxxx,'this is a test');
-SET &YYYY=GETENV(xxxx);
-TYPE  Return Code: &XXXX,  Variable value: &YYYY
```

The output is:

```
Return Code: 0,  Variable value: this is a test
```

System Functions

System functions call the operating system to obtain information about the operating environment or to use a system service.

For many functions, the output argument can be supplied either as a field name or as a format enclosed in single quotation marks. However, if a function is called from a Dialogue Manager command, this argument must always be supplied as a format. For detailed information about calling a function and supplying arguments, see Accessing and Calling a Function.

CLSDDREC: Closing All Files Opened by the PUTDDREC Function

The CLSDDREC function closes all files opened by the PUTDDREC function. If PUTDDREC is called in a Dialogue Manager -SET command, the files opened by PUTDDREC are not closed automatically until the end of a request or connection. In this case, you can close the files and free the memory used to store information about the open file by calling the CLSDDREC function.

Close All Files Opened by the PUTDDREC Function

```
CLSDDREC(output)
```

where:

output

Integer

Is the return code, which can be one of the following values:

- **0,** which indicates that the files are closed.
- **1,** which indicates an error while closing the files.

Closing Files Opened by the PUTDDREC Function

This example closes files opened by the PUTDDREC function:

```
CLSDDREC('I1')
```

FEXERR: Retrieving an Error Message

The FEXERR function retrieves an error message. It is especially useful in a procedure using a command that suppresses the display of output messages.

An error message consists of up to four lines of text. The first line contains the message and the remaining three contain a detailed explanation, if one exists. FEXERR retrieves the first line of the error message.

Retrieve an Error Message

```
FEXERR(error, 'A72')
```

where:

error

Numeric

Is the error number, up to 5 digits long.

'A72'

Is the format of the output value enclosed in single quotation marks. The format is A72, the maximum length of an error message.

Retrieving an Error Message

FEXERR retrieves the error message whose number is contained in the &ERR variable, in this case 650. The result is returned to the variable &&MSGVAR and has the format A72.

```
-SET &ERR = 650;
-SET &&MSGVAR = FEXERR(&ERR, 'A72');
-TYPE &&MSGVAR
```

The output is:

```
(FOC650) THE DISK IS NOT ACCESSED
```

FGETENV: Retrieving the Value of an Environment Variable

The FGETENV function retrieves the value of an environment variable and returns it as an alphanumeric string.

Retrieve the Value of an Environment Variable

where:

Integer

varname

Alphanumeric

Is the name of the environment variable whose value is being retrieved.

Integer

Alphanumeric

FINDMEM: Finding a Member of a Partitioned Data Set

The FINDMEM function, available only on z/OS, determines if a specific member of a partitioned data set (PDS) exists. This function is used primarily in Dialogue Manager procedures.

To use this function, allocate the PDS to a ddname because the ddname is required in the function call. You can search multiple PDSs with one function call if they are concatenated to one ddname.

Find a Member of a Partitioned Data Set

```
FINDMEM(ddname, member, output)
```

where:

ddname

A8

Is the ddname to which the PDS is allocated. This value must be an eight-character literal enclosed in single quotation marks, or a variable that contains the ddname. If you supply a literal less than eight characters long, pad it with trailing spaces.

member

A8

Is the member for which you are searching. This value must be eight characters long. If you supply a literal that has less than eight characters, pad it with trailing spaces.

output

A1

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The result is one of the following:

`Y` indicates the member exists in the PDS.

N indicates the member does not exist in the PDS.

E indicates an error occurred. Either the data set is not allocated to the ddname, or the data set allocated to the ddname is not a PDS (and may be a sequential file).

Finding a Member of a Partitioned Data Set

FINDMEM searches for the EMPLOYEE Master File in the PDS allocated to ddname MASTER, and returns the result to the variable &FINDCODE. The result has the format A1:

```
-SET &FINDCODE = FINDMEM('MASTER  ', 'EMPLOYEE', 'A1');-IF &FINDCODE EQ
'N' GOTO NOMEM;
-IF &FINDCODE EQ 'E' GOTO NOPDS;
-TYPE MEMBER EXISTS, RETURN CODE = &FINDCODE
TABLE FILE EMPLOYEE
PRINT CURR_SAL BY LAST_NAME BY FIRST_NAME
WHERE RECORDLIMIT EQ 4;
END
-EXIT
-NOMEM
-TYPE EMPLOYEE NOT FOUND IN MASTER FILE PDS
-EXIT
-NOPDS
-TYPE ERROR OCCURRED IN SEARCH
-TYPE CHECK IF FILE IS A PDS ALLOCATED TO DDNAME MASTER
-EXIT
```

The output is:

```
MEMBER EXISTS, RETURN CODE = Y
>   NUMBER OF RECORDS IN TABLE=        4  LINES=      4
LAST_NAME        FIRST_NAME         CURR_SAL
---------        ----------         --------
JONES            DIANE            $18,480.00
SMITH            MARY             $13,200.00
                 RICHARD           $9,500.00
STEVENS          ALFRED           $11,000.00
```

GETPDS: Determining If a Member of a Partitioned Data Set Exists

Available Operating Systems: z/OS

The GETPDS function determines if a specific member of a partitioned data set (PDS) exists, and if it does, returns the PDS name. This function is used primarily in Dialogue Manager procedures.

To use this function, allocate the PDS to a ddname because the ddname is required in the function call. You can search multiple PDSs with one function call if they are concatenated to one ddname.

GETPDS is almost identical to FINDMEM, except that GETPDS provides either the PDS name or returns a different set of status codes.

Determine If a PDS Member Exists

```
GETPDS(ddname, member, output)
```

where:

ddname

A8

Is the ddname to which the PDS is allocated. This value must be an eight-character literal enclosed in single quotation marks, or a variable that contains the ddname. If you supply a literal less than eight characters long, pad it with trailing spaces.

member

A8

Is the member for which the function searches. This value must be eight characters long. If you supply a literal with less than eight characters, pad it with trailing spaces.

output

A44

Is the name of the field that contains the result, or the format of the output value enclosed in single quotation marks. The maximum length of a PDS name is 44. The result is one of the following:

`PDS name` is the name of the PDS that contains the member, if it exists.

`*D` indicates the ddname is not allocated to a data set.

`*M` indicates the member does not exist in the PDS.

`*E` indicates an error occurred. For example, the data set allocated to the ddname is not a PDS (and may be a sequential file).

Determining If a PDS Member Exists

GETPDS searches for the member specified by &MEMBER in the PDS allocated to &DDNAME, and returns the result to &PNAME. The result has the format A44.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '                                            ';
-SET &PNAME = GETPDS(&DDNAME, &MEMBER, 'A44');
-IF &PNAME EQ '*D' THEN GOTO DDNOAL;
-IF &PNAME EQ '*M' THEN GOTO MEMNOF;
-IF &PNAME EQ '*E' THEN GOTO DDERROR;
-*
-TYPE MEMBER &MEMBER IS FOUND IN
-TYPE THE PDS &PNAME
-TYPE ALLOCATED TO &DDNAME
-*
-EXIT
-DDNOAL
-*
-TYPE DDNAME &DDNAME NOT ALLOCATED
-*
-EXIT
-MEMNOF
-*
-TYPE MEMBER &MEMBER NOT FOUND UNDER DDNAME &DDNAME
-*
-EXIT
-DDERROR
-*
-TYPE ERROR IN GETPDS; DATA SET PROBABLY NOT A PDS.
```

```
-*
-EXIT
```

The output is similar to the following:

```
MEMBER EMPLOYEE IS FOUND IN
THE PDS USER1.MASTER.DATA
ALLOCATED TO MASTER
```

Copying a Member for Editing in TED

GETPDS searches for the member specified by &MEMBER in the PDS allocated to &DDNAME, and returns the result to &PNAME. The DYNAM commands copy the member from the production PDS to the local PDS. Then the TED editor enables you to edit the member. The ddnames are allocated earlier in the session: the production PDS is allocated to the ddname MASTER; the local PDS to ddname MYMASTER.

```
-* If the Master File in question is in the production PDS, it must
-* be copied to a local PDS, which has been allocated previously to the
-* ddname MYMASTER before any changes can be made.
-* Assume the Master File in question is supplied via a -CRTFORM, with
-* a length of 8 characters, as &MEMBER.
-*
```

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = &MEMBER;
```

```
-SET &PNAME = '                                            ';
-SET &PNAME = GETPDS(&DDNAME, &MEMBER, 'A44');-IF &PNAME EQ '*D' OR '*M'
OR '*E' THEN GOTO DDERROR;
-*
DYNAM ALLOC FILE XXXX DA -
   &PNAME MEMBER &MEMBER SHR
DYNAM COPY XXXX MYMASTER MEMBER &MEMBER
-RUN
TED MYMASTER(&MEMBER)
-EXIT
-*
-DDERROR
```

```
-*
-TYPE Error in GETPDS; Check allocation for &DDNAME for
-TYPE proper allocation.
-*
-EXIT
```

Earlier in the session, allocate the ddnames:

```
>  > tso alloc f(master) da('prod720.master.data') shr
>  > tso alloc f(mymaster) da('user1.master.data') shr
```

Run the procedure, and specify the EMPLOYEE member. It is copied to your local PDS, and you access TED.

```
PLEASE SUPPLY VALUES REQUESTED

MEMBER=  > EMPLOYEE


MYMASTER(EMPLOYEE)                    SIZE=37    LINE=0

00000 * * * TOP OF FILE * * *
00001 FILENAME=EMPLOYEE, SUFFIX=FOC
00002 SEGNAME=EMPINFO,  SEGTYPE=S1
00003  FIELDNAME=EMP_ID,       ALIAS=EID,     FORMAT=A9,       $
00004  FIELDNAME=LAST_NAME,    ALIAS=LN,      FORMAT=A15,      $
00005  FIELDNAME=FIRST_NAME,   ALIAS=FN,      FORMAT=A10,      $
00006  FIELDNAME=HIRE_DATE,    ALIAS=HDT,     FORMAT=I6YMD,    $
00007  FIELDNAME=DEPARTMENT,   ALIAS=DPT,     FORMAT=A10,      $
```

Displaying the Attributes of a PDS

To view the attributes of the PDS that contains a specific member, this Dialogue Manager procedure can search for the EMPLOYEE member in the PDS allocated to the ddname MASTER and, based on its existence, allocate the PDS to the ddname TEMPMAST. Dialogue Manager system variables are used to display the attributes.

```
-SET &DDNAME = 'MASTER  ';
-SET &MEMBER = 'EMPLOYEE';
-SET &PNAME = '                                            ';
```

```
-SET &PNAME = GETPDS(&DDNAME, &MEMBER, 'A44');
-IF &PNAME EQ '*D' OR '*M' OR '*E' THEN GOTO DDERROR;
-*
DYNAM ALLOC FILE TEMPMAST DA -
   &PNAME SHR
-RUN
-? MVS DDNAME TEMPMAST
-TYPE The data set attributes include:
-TYPE Data set name is: &DSNAME
-TYPE Volume is: &VOLSER
-TYPE Disposition is: &DISP
-EXIT
-*
-DDERROR
-TYPE Error in GETPDS; Check allocation for &DDNAME for
-TYPE proper allocation.
-*
-EXIT
```

The sample output is:

```
THE DATA SET ATTRIBUTES INCLUDE:
DATA SET NAME IS: USER1.MASTER.DATA
VOLUME IS: USERM0
DISPOSITION IS: SHR
```

GETUSER: Retrieving a User ID

The GETUSER function retrieves the ID of the connected user. GETUSER can also retrieve the name of a z/OS batch job if you run the function from the batch job.

Retrieve a User ID

```
GETUSER(output)
```

where:

output

Alphanumeric, at least A8

Is the result field, whose length depends on the platform on which the function is issued. Provide a length as long as required for your platform; otherwise the output will be truncated.

Retrieving a User ID

GETUSER retrieves the user ID of the person running the request:

```
DEFINE FILE EMPLOYEE
USERID/A8 WITH EMP_ID = GETUSER(USERID);
END
```

```
TABLE FILE EMPLOYEE
SUM CURR_SAL AS 'TOTAL SALARIES'
BY DEPARTMENT
HEADING
"SALARY REPORT RUN FROM USERID: <USERID"
" "
END
```

The output is:

```
SALARY REPORT RUN FROM USERID: USER1

DEPARTMENT   TOTAL SALARIES
----------   --------------
MIS             $108,002.00
PRODUCTION      $114,282.00
```

JOBNAME: Retrieving the Current Process Identification String

The JOBNAME function retrieves the raw identification string of the current process from the operating system. This is also commonly known as a process PID at the operating system level. The function is valid in all environments, but is typically used in Dialogue Manager and returns the value as an alphanumeric string (even though a PID is pure numeric on some operating systems).

Note: JOBNAME strings differ between some operating systems in terms of look and length. For example, Windows, UNIX, and z/OS job names are pure numeric (typically a maximum of 8 characters long), while an OpenVMS job name is a hex number (always 8 characters long), and an IBM i job name is a three-part string that has a 26 character maximum length. Since an application may eventually be run in another (unexpected) environment in the future, it is good practice to use the maximum length of 26 to avoid accidental length truncation in the future. Applications using this function for anything more than simple identification may also need to account for the difference in the application code.

Retrieve the Current Process Identification String

```
JOBNAME(length, output)
```

where:

length

Integer

Is the maximum number of characters to return from the PID system call.

output

Alphanumeric

Is the returned process identification string, whose length depends on the platform on which the function is issued. Provide a length as long as required for your platform. Otherwise, the output will be truncated.

Retrieving a Process Identification String

The following example uses the JOBNAME function to retrieve the current process identification string to an A26 string and then truncate it for use in a -TYPE statement:

```
-SET &JOBNAME = JOBNAME(26, 'A26');
-SET &JOBNAME = TRUNCATE(&JOBNAME);
-TYPE The Current system PID &JOBNAME is processing.
```

For example, on Windows, the output is similar to the following:

```
The Current system PID 2536 is processing.
```

MVSDYNAM: Passing a DYNAM Command to the Command Processor

Available Operating Systems: z/OS

The MVSDYNAM function transfers a FOCUS DYNAM command to the DYNAM command processor. It is useful in passing allocation commands to the processor in a compiled MODIFY procedure after the CASE AT START command.

Pass a DYNAM Command to the Command Processor

```
MVSDYNAM(command, length, outfield)
```

where:

command

Alphanumeric

Is the DYNAM command enclosed in single quotation marks, or a field or variable that contains the command. The function converts lowercase input to uppercase.

length

Numeric

Is the maximum length of the command, in characters, between 1 and 256.

outfield

I4

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

MVSDYNAM returns one of the following codes:

`0` indicates the DYNAM command transferred and executed successfully.

positive number is the error number corresponding to a FOCUS error.

negative number is the FOCUS error number corresponding to a DYNAM failure.

In Dialogue Manager, you must specify the format.

Passing a DYNAM Command to the Command Processor

MVSDYNAM passes the DYNAM command contained in LINE to the processor. The return code is stored in RES.

```
-* THE RESULT OF ? TSO DDNAME CAR WILL BE BLANK AFTER ENTERING
-* 'FREE FILE CAR' AS YOUR COMMAND
DYNAM ALLOC FILE CAR DS USER1.CAR.FOCUS SHR REUSE
? TSO DDNAME CAR
-RUN
-PROMPT &XX.ENTER A SPACE TO CONTINUE.
MODIFY FILE CAR
COMPUTE LINE/A60=;
        RES/I4 = 0;
CRTFORM
" ENTER DYNAM COMMAND BELOW:"
" <LINE>"
```

```
COMPUTE
RES = MVSDYNAM(LINE, 60, RES);GOTO DISPLAY
 CASE DISPLAY
 CRTFORM LINE 1
" THE RESULT OF DYNAM WAS <D.RES"
GOTO EXIT
ENDCASE
DATA
END
? TSO DDNAME CAR
```

The first query command displays the allocation that results from the DYNAM ALLOC command:

```
DDNAME      =  CAR
DSNAME      =  USER1.CAR.FOCUS
DISP        =  SHR
DEVICE      =  DISK
VOLSER      =  USERMN
DSORG       =  PS
RECFM       =  F
SECONDARY   =   100
ALLOCATION  =  BLOCKS
BLKSIZE     =      4096
LRECL       =      4096
TRKTOT      =         8
EXTENTSUSED =         1
BLKSPERTRK  =        12
TRKSPERCYL  =        15
CYLSPERDISK =      2227
BLKSWRITTEN =        96
FOCUSPAGES  =         8
ENTER A SPACE TO CONTINUE  >
```

Type one space and press **Enter** to continue. Then enter the DYNAM FREE command (the DYNAM keyword is assumed):

```
ENTER DYNAM COMMAND BELOW:
 free file car
```

The function successfully passes the DYNAM FREE command to the processor and the return code displays:

```
THE RESULT OF DYNAM WAS     0
```

Press **Enter** to continue. The second query command indicates that the allocation was freed:

```
DDNAME      =  CAR
DSNAME      =
DISP        =
DEVICE      =
VOLSER      =
DSORG       =
RECFM       =
SECONDARY   =  ****
ALLOCATION  =
BLKSIZE     =         0
LRECL       =         0
TRKTOT      =         0
EXTENTSUSED =         0
BLKSPERTRK  =         0
TRKSPERCYL  =         0
CYLSPERDISK =         0
BLKSWRITTEN =         0
>
```

PUTDDREC: Writing a Character String as a Record in a Sequential File

The PUTDDREC function writes a character string as a record in a sequential file. The file must be identified with a DYNAM command. TSO ALLOCATE does not work. If the file is defined as an existing file (with the APPEND option), the new record is appended. If the file is defined as NEW and it already exists, the new record overwrites the existing file.

For information about the DYNAM command, see the *ibi™ FOCUS® Overview and Operating Environments* manual.

PUTDDREC opens the file if it is not already open. Each call to PUTDDREC can use the same file or a new one. All of the files opened by PUTDDREC remain open until the end of a request or session. At the end of the request or session, all files opened by PUTDDREC are automatically closed.

- The open, close, and write operations are handled by the operating system. Therefore, the requirements for writing to the file and the results of deviating from the instructions when calling PUTDDREC are specific to your operating environment. Make sure you are familiar with and follow the guidelines for your operating system when performing input/output operations.

- You can call PUTDDREC in a DEFINE FILE command or in a DEFINE in the Master File. However, PUTDDREC does not open the file until its field name is referenced in a request.

If PUTDDREC is called in a Dialogue Manager -SET command, the files opened by PUTDDREC are not closed automatically until the end of a request or session. In this case, you can close the files and free the memory used to store information about the open file by calling the CLSDDREC function.

Write a Character String as a Record in a Sequential File

```
PUTDDREC(ddname, dd_len, record_string, record_len, output)
```

where:

ddname

Alphanumeric

Is the logical name assigned to the sequential file in a DYNAM command.

dd_len

Numeric

Is the number of characters in the logical name.

record_string

Alphanumeric

Is the character string to be added as the new record in the sequential file.

record_len

Numeric

Is the number of characters to add as the new record.

It cannot be larger than the number of characters in *record_string*. To write all of *record_string* to the file, *record_len* should equal the number of characters in *record_string* and should not exceed the record length declared in the DYNAM command. If *record_len* is shorter than the declared length declared, the resulting file may contain extraneous characters at the end of each record. If *record_string* is longer than the declared length, *record_string* may be truncated in the resulting file.

output

Integer

Is the return code, which can have one of the following values:

```
 0 - Record is added.
```

```
-1 - FILEDEF statement is not found.
```

```
-2 - Error while opening the file.
```

```
-3 - Error while adding the record to the file.
```

Calling PUTDDREC in a TABLE Request

The following example defines a new file whose logical name is PUTDD1. The TABLE request then calls PUTDDREC for each employee in the EMPLOYEE data source and writes a record to the file composed of the employee's last name, first name, employee ID, current job code, and current salary (converted to alphanumeric using the EDIT function). The return code of zero (in OUT1) indicates that the calls to PUTDDREC were successful:

```
DYNAM ALLOC PUTDD1 DA USER1.PUTDD1.DATATABLE FILE EMPLOYEE
PRINT EMP_ID CURR_JOBCODE AS 'JOB' CURR_SAL
COMPUTE SALA/A12 = EDIT(CURR_SAL); NOPRINT
COMPUTE EMP1/A50= LAST_NAME|FIRST_NAME|EMP_ID|CURR_JOBCODE|SALA;
NOPRINT
COMPUTE OUT1/I1 = PUTDDREC('PUTDD1',6, EMP1, 50, OUT1);
BY LAST_NAME BY FIRST_NAME
END
```

The output is:

```
LAST_NAME        FIRST_NAME  EMP_ID     JOB         CURR_SAL  OUT1
---------        ----------  ---------  ---         --------  ----
BANNING          JOHN        119329144  A17       $29,700.00     0
BLACKWOOD        ROSEMARIE   326179357  B04       $21,780.00     0
CROSS            BARBARA     818692173  A17       $27,062.00     0
GREENSPAN        MARY        543729165  A07        $9,000.00     0
IRVING           JOAN        123764317  A15       $26,862.00     0
JONES            DIANE       117593129  B03       $18,480.00     0
MCCOY            JOHN        219984371  B02       $18,480.00     0
MCKNIGHT         ROGER       451123478  B02       $16,100.00     0
ROMANS           ANTHONY     126724188  B04       $21,120.00     0
SMITH            MARY        112847612  B14       $13,200.00     0
                 RICHARD     119265415  A01        $9,500.00     0
STEVENS          ALFRED      071382660  A07       $11,000.00     0
```

After running this request, the sequential file contains the following records:

```
BANNING        JOHN       119329144A17000000029700
BLACKWOOD      ROSEMARIE  326179357B04000000021780
CROSS          BARBARA    818692173A17000000027062
GREENSPAN      MARY       543729165A07000000009000
IRVING         JOAN       123764317A15000000026862
JONES          DIANE      117593129B03000000018480
MCCOY          JOHN       219984371B02000000018480
MCKNIGHT       ROGER      451123478B02000000016100
ROMANS         ANTHONY    126724188B04000000021120
SMITH          MARY       112847612B14000000013200
SMITH          RICHARD    119265415A01000000009500
STEVENS        ALFRED     071382660A07000000011000
```

Calling PUTDDREC and CLSDDREC in Dialogue Manager -SET Commands

The following example defines a new file whose logical name is PUTDD1. The first -SET command creates a record to add to this file. The second -SET command calls PUTDDREC to add the record. The last -SET command calls CLSDDREC to close the file. The return codes are displayed to make sure operations were successful:

```
DYNAM ALLOC PUTDD1 DA USER1.PUTDD1.DATA –SET &EMP1 =
'SMITH'|'MARY'|'A07'|'27000';
```

```
-TYPE DATA = &EMP1
-SET &OUT1 = PUTDDREC('PUTDD1',6, &EMP1, 17, 'I1');
-TYPE PUT RESULT = &OUT1
-SET &OUT1 = CLSDDREC('I1');
-TYPE CLOSE RESULT = &OUT1
```

The output is:

```
DATA = SMITHMARYA0727000
PUT RESULT = 0
CLOSE RESULT = 0
```

After running this procedure, the sequential file contains the following record:

```
SMITHMARYA0727000
```

SLEEP: Suspending Execution for a Given Number of Seconds

The SLEEP function suspends execution for the number of seconds you specify as its input argument.

This function is most useful in Dialogue Manager when you need to wait to start a specific procedure. For example, you can start a FOCUS Database Server and wait until the server is started before initiating a client application.

Suspend Execution for a Specified Number of Seconds

```
SLEEP(delay, output);
```

where:

delay

Numeric

Is the number of seconds to delay execution. The number can be specified down to the millisecond.

output

Numeric

Is the name of a field or a format enclosed in single quotation marks. The value returned is the same value you specify for delay.

Suspending Execution for Four Seconds

The following example computes the current date and time, suspends execution for 4 seconds, and computes the current date and time after the delay:

```
TABLE  FILE VIDEOTRK
PRINT TRANSDATE NOPRINT
COMPUTE
START_TIME/HYYMDSa = HGETC(8, START_TIME);
DELAY/I2 = SLEEP(4.0, 'I2');
END_TIME/HYYMDSa = HGETC(8, END_TIME);
IF RECORDLIMIT EQ 1
END
```

The output is:

```
START_TIME             DELAY  END_TIME
----------             -----  --------
2007/10/26  5:04:36pm      4  2007/10/26  5:04:40pm
```

SYSVAR: Retrieving the Value of a z/OS System Variable

Available Operating Systems: z/OS

The SYSVAR function populates a Dialogue Manager amper variable with the contents of any z/OS system variable. System variables are in the format [&]*name*[.], where the dot is optional. They can be provided by the operating system or can be user defined. The function can be called in a -SET command.

Retrieve the Value of a z/OS System Variable

```
-SET &dmvar = SYSVAR('length','[&]sysvar[.]','outfmt');
```

where:

&dmvar

Alphanumeric

Is the name of the Dialogue Manager variable to be populated with the value of the z/OS system variable.

length

Alphanumeric

Is the length of the next parameter in the call. Do not include the escape character in the length, if one is present in the *sysvar* argument.

[&|]sysvar[.]

Alphanumeric

Is the name of the system variable to be retrieved. Note that the ampersand (&) and the dot (.) are optional. If the ampersand is included, it must be followed by the escape character (|).

outfmt

Alphanumeric

Is the format of the returned value enclosed in single quotation marks.

Retrieving the Value of the z/OS SYSNAME Variable

The following example populates the Dialogue Manager variable named &MYSNAME2 with the value of the z/OS SYSNAME variable:

```
-SET &MYSNAME2=SYSVAR('7','SYSNAME','A8');
-TYPE SYSNAME:&MYSNAME2
```

The output is similar to the following:

```
SYSNAME:IBI1
```

SQL Character Functions

SQL character functions manipulate alphanumeric fields and character strings.

They can be used in SQL Translator requests and, where supported by the DBMS, in Direct SQL Passthru requests.

LOCATE: Returning the Position of a Substring in a String

Given a substring, a source string and a starting position (the default is 1), LOCATE returns the position of the first occurrence of the substring, starting the search at the starting position. If the substring is not found, LOCATE returns zero (0). The search is case insensitive.

Return the Position of a Substring in a String

```
LOCATE(substr, source [,start])
```

where:

substr

Alphanumeric

Is the search string.

source

Alphanumeric

Is the source string.

start

Numeric

Is the optional starting position for the search. If omitted, it defaults to 1.

Returning the Position of a Substring in a String

The following SQL SELECT statement searches for the character *a* in FULLNAME, starting at position 3, and starting at position 1.

```
SQL
SELECT FULLNAME,
LOCATE('a', FULLNAME, 3) AS 'START AT 3',
LOCATE('a', FULLNAME) AS 'START AT 1'
FROM
WF_RETAIL_CUSTOMER T1
FETCH FIRST 5 ROWS ONLY;
TABLE
HEADING CENTER
"Search for the Character 'a'"
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
TYPE=HEADING, STYLE=BOLD, SIZE=16, $
ENDSTYLE
END
```

The output is shown in the following image.

Search for the Character 'a'

Full Name	START AT 3	START AT 1
Tyler Nolan	10	10
Joshua Bull	6	6
Zara Wood	4	2
Callum McKenzie	0	2
Bradley Charlton	3	3

SQL Miscellaneous Functions

The SQL functions described in this chapter perform a variety of conversions, tests, and manipulations.

They can be used in SQL Translator requests and, where supported by the DBMS, in Direct SQL Passthru requests.

CHR: Returning the ASCII Character Given a Numeric Code

Given a number code as an argument, CHR returns the ASCII character.

Return the ASCII Character Given a Numeric Code

```
CHR(number)
```

where:

number

Numeric

Is the numeric code to be translated to an ASCII character.

Returning the ASCII Character Given a Numeric Code

The following SELECT statement places a colon character between last name and first name.

```
SQL
SELECT
LAST_NAME AS ' ', CHR(58) AS ' ', FIRST_NAME AS ' '
FROM EMPLOYEE
;
TABLE
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

STEVENS	:	ALFRED
SMITH	:	MARY
JONES	:	DIANE
SMITH	:	RICHARD
BANNING	:	JOHN
IRVING	:	JOAN
ROMANS	:	ANTHONY
MCCOY	:	JOHN
BLACKWOOD	:	ROSEMARIE
MCKNIGHT	:	ROGER
GREENSPAN	:	MARY
CROSS	:	BARBARA

Trigonometric Functions

The trigonometric functions provide trigonometric calculations, inverse trigonometric calculations, and angle conversion functions.

ACOS: Calculating an Angle Given its Cosine

Given the cosine of an angle in radians, ACOS (arccosine) returns an angle between 0 (zero) and pi radians.

Calculate an Angle Given its Cosine

```
ACOS(expression)
```

where:

expression

Numeric

Is the cosine of an angle.

Calculating the Arccosine of a Value

The following request calculates the arccosine of 0, PI/2 radians, PI/4 radians, and PI radians.

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
COS1 = COS(0);
COS2 = COS(PI2);
```

```
COS3 = COS(PI4);
COS4 = COS(PI1);
END

TABLE FILE ggsales
PRINT
COS1 COS2 COS3 COS4
OVER
COMPUTE
ARCCOS1/D12.2 = ACOS(COS1);
ARCCCOS2/D12.2 = ACOS(COS2);
ARCCOS3/D12.2 = ACOS(COS3);
ARCCOS4/D12.2 = ACOS(COS4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date								
1997/04/01	COS1	1.00	COS2	.00	COS3	.71	COS4	-1.00
	ARCCOS1	.00	ARCCCOS2	1.57	ARCCOS3	.79	ARCCOS4	3.14

ASIN: Calculating an Angle Given its Sine

Given the sine of an angle in radians, ASIN (arcsine) returns an angle between -(pi/2) and pi/2 radians.

Calculate an Angle Given its Sine

```
ASIN(expression)
```

where:

expression

Numeric

Is the sine of an angle.

Calculating the Arcsine of a Value

The following request calculates the arcsine of 0, PI/2 radians, PI/4 radians, and PI radians.

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
SIN1 = SIN(0);
SIN2 = SIN(PI2);
SIN3 = SIN(PI4);
SIN4 = SIN(PI1);
END

TABLE FILE ggsales
PRINT
SIN1 SIN2 SIN3 SIN4
OVER
COMPUTE
ARCSIN1/D12.2 = ASIN(SIN1);
ARCCSIN2/D12.2 = ASIN(SIN2);
ARCSIN3/D12.2 = ASIN(SIN3);
ARCSIN4/D12.2 = ASIN(SIN4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date								
1997/04/01	SIN1	.00	SIN2	1.00	SIN3	.71	SIN4	.00
	ARCSIN1	.00	ARCCSIN2	1.57	ARCSIN3	.79	ARCSIN4	.00

ATAN: Calculating an Angle Given its Tangent

Given the tangent of an angle in radians, ATAN (arctangent) returns an angle between -(pi/2) and pi/2 radians.

Calculate an Angle Given its Tangent

```
ATAN(expression)
```

where:

expression

Numeric

Is the tangent of an angle.

Calculating the Arctangent of a Value

The following request calculates the arctangent of 0, PI/4 radians, and PI radians.

```
DEFINE FILE ggsales
PI1 = PI();
PI4 = PI()/4;
TAN1 = TAN(0);
TAN3 = TAN(PI4);
TAN4 = TAN(PI1);
END

TABLE FILE ggsales
PRINT
TAN1 TAN2 TAN3 TAN4
OVER
COMPUTE
ARCTAN1/D12.2 = ATAN(TAN1);
ARCTAN3/D12.2 = ATAN(TAN3);
ARCTAN4/D12.2 = ATAN(TAN4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
```

```
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date						
1997/04/01	TAN1	.00	TAN3	1.00	TAN4	.00
	ARCTAN1	.00	ARCTAN3	.79	ARCTAN4	.00

ATAN2: Calculating an Angle Given the Coordinates of its Tangent

Given the coordinates of the tangent of an angle in radians, ATAN2 (arctangent2) returns an angle between -pi and pi radians.

Calculate an Angle Given the Coordinates of its Tangent

```
ATAN2(x,y)
```

where:

x

Numeric

Is the x-coordinate in radians of the tangent of an angle.

y

Numeric

Is the y-coordinate in radians of the tangent of an angle.

Calculating the Arctangent of a Set of Coordinates

The following request calculates the arctangent of (PI,0), (PI/4,PI/4), and (PI,PI).

```
DEFINE FILE ggsales
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT
TAN1 TAN2 TAN3 TAN4
OVER
COMPUTE
ATAN2A/D12.2 = ATAN2(PI(),0);
ATAN2B/D12.2 = ATAN2(PI4,PI4);
ATAN2C/D12.2 = ATAN2(PI(),PI());
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	ATAN2A	ATAN2B	ATAN2C
1997/04/01	1.57	.79	.79

COS: Calculating the Cosine of an Angle

Given an angle in radians, COS calculates the cosine of the angle.

Calculate the Cosine of an Angle

```
COS(expression)
```

where:

expression

Numeric

Is an angle in radians.

Calculating the Cosine of an Angle

The following request calculates the cosine of 0, PI/2 radians, PI/4 radians, and PI radians, then calculates the arccosine of those cosines.

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
COSINE1 = COS(0);
COSINE2 = COS(PI2);
COSINE3 = COS(PI4);
COSINE4 = COS(PI1);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	COSINE1	COSINE2	COSINE3	COSINE4
1997/04/01	1.00	.00	.71	-1.00

COT: Calculating the Cotangent of an Angle

Given an angle in radians, COT calculates the cotangent of the angle.

Calculate the Cotangent of an Angle

```
COT(expression)
```

where:

expression

Numeric

Is an angle in radians.

Calculating the Cotangent of an Angle

The following request calculates the cotangent of PI/2 and PI/4 radians.

```
DEFINE FILE ggsales
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
COTGENT2 = COT(PI2);
COTGENT3 = COT(PI4);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	COT1	COT2
1997/04/01	.00	1.00

DEGREES: Converting Radians to Degrees

DEGREES converts an angle in radians to an angle in degrees.

Convert Radians to Degrees

```
DEGREES(expression)
```

where:

expression

Numeric

Is an angle in radians.

Converting Radians to Degrees

The following request converts 0, PI/2, PI/4, and PI radians to degrees.

```
DEFINE FILE ggsales
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
DEG1/D12.2 = DEGREES(0);
DEG2/D12.2 = DEGREES(PI2);
DEG3/D12.2 = DEGREES(PI4);
DEG4/D12.2 = DEGREES(PI());
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	DEG1	DEG2	DEG3	DEG4
1997/04/01	.00	90.00	45.00	180.00

PI: Returning the Constant Pi

PI returns the constant pi as a floating-point number.

Returning the Value Pi

```
PI()
```

Returning the Constant Pi

The following request returns the constant pi rounded to two decimal places and rounded to 10 decimal places.

```
TABLE FILE ggsales
PRINT COMPUTE
PI2/D12.2 = PI();
PI10/D12.10 = PI();
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	PI2	PI10
1997/04/01	3.14	3.1415926536

RADIANS: Converting Degrees to Radians

RADIANS converts an angle in degrees to an angle in radians.

Convert Degrees to Radians

```
RADIANS(expression)
```

where:

expression

Numeric

Is an angle in degrees.

Converting Radians to Degrees

The following request converts 0, 45, 90, and 180 degrees to radians.

```
TABLE FILE ggsales
PRINT COMPUTE
RAD1/D12.2 = RADIANS(0);
RAD2/D12.2 = RADIANS(45);
RAD3/D12.2 = RADIANS(90);
RAD4/D12.2 = RADIANS(180);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	RAD1	RAD2	RAD3	RAD4
1997/04/01	.00	.79	1.57	3.14

SIN: Calculating the Sine of an Angle

Given an angle in radians, SIN calculates the sine of the angle.

Calculate the Sine of an Angle

```
SIN(expression)
```

where:

expression

Numeric

Is an angle in radians.

Calculating the Sine of an Angle

The following request calculates the sine of 0, PI/2 radians, PI/4 radians, and PI radians.

```
DEFINE FILE ggsales
PI1 = PI();
PI2 = PI()/2;
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
SINE1 = SIN(0);
SINE2 = SIN(PI2);
SINE3 = SIN(PI4);
SINE4 = SIN(PI1);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	SINE1	SINE2	SINE3	SINE4
1997/04/01	.00	1.00	.71	.00

TAN: Calculating the Tangent of an Angle

Given an angle in radians, TAN calculates the tangent of the angle.

Calculate the Tangent of an Angle

```
TAN(expression)
```

where:

expression

Numeric

Is an angle in radians.

Calculating the Tangent of an Angle

The following request calculates the tangent of 0, PI/4 radians, and PI radians.

```
DEFINE FILE ggsales
PI1 = PI();
PI4 = PI()/4;
END

TABLE FILE ggsales
PRINT COMPUTE
TANGENT1 = TAN(0);
TANGENT2 = TAN(PI4);
TANGENT3 = TAN(PI1);
BY DATE
WHERE RECORDLIMIT EQ 1
ON TABLE SET PAGE NOLEAD
```

```
ON TABLE SET STYLE *
GRID=OFF,$
ENDSTYLE
END
```

The output is shown in the following image.

Date	TANGENT1	TANGENT2	TANGENT3
1997/04/01	.00	1.00	.00

Creating a Subroutine

You can create custom subroutines to use in addition to the functions provided by FOCUS. The process of creating a subroutine consists of the following steps:

- Writing a subroutine using any language that supports subroutine calls. Some of the most common languages are FORTRAN, COBOL, PL/I, Assembler, and C. For details, see Writing a Subroutine.
- Compiling the subroutine. For details, see Compiling and Storing a Subroutine.
- Storing the subroutine in a separate file; do not include it in the main program. For details, see Compiling and Storing a Subroutine.
- Testing the subroutine. For details, see Testing the Subroutine.

Note: On z/OS, all subroutines called by FOCUS must be fully LE compliant.

Writing a Subroutine

You can write a subroutine in any language that supports subroutines. If you intend to make your subroutine available to other users, be sure to document what your subroutine does, what the arguments are, what formats they have, and in what order they must appear in the subroutine call.

When you write a subroutine you need to consider the requirements and limits that affect it. These are:

- Naming conventions. For details, see Writing a Subroutine.
- Argument considerations. For details, see Writing a Subroutine.
- Language considerations. For details, see Writing a Subroutine.
- Programming considerations. For details, see Writing a Subroutine.

If you write a program named INTCOMP that calculates the amount of money in an account earning simple interest, the program reads a record, tests if the data is acceptable, and then calls a subroutine called SIMPLE that computes the amount of money. The program and the subroutine are stored together in the same file.

The program and the subroutine shown here are written in the pseudocode (a method of representing computer code in a general way):

```
Begin program INTCOMP.
Execute this loop until end-of-file.
   Read next record, fields: PRINCPAL, DATE_PUT, YRRATE.
   If PRINCPAL is negative or greater than 100,000,
      reject record.
   If DATE_PUT is before January 1, 1975, reject record.
   If YRRATE is negative or greater than 20%, reject record.
   Call subroutine SIMPLE (PRINCPAL, DATE_PUT, YRRATE, TOTAL).
   Print PRINCPAL, YEARRATE, TOTAL.
End of loop.
End of program.
```

```
Subroutine SIMPLE (AMOUNT, DATE, RATE, RESULT).
Retrieve today's date from the system.
Let NO_DAYS = Days from DATE until today's date.
Let DAY_RATE = RATE / 365 days in a year.
Let RESULT = AMOUNT * (NO_DAYS * DAY_RATE + 1).
End of subroutine.
```

If you move the SIMPLE subroutine into a file separate from the main program and compile it, you can call the subroutine. The following report request shows how much money employees would accrue if they invested salaries in accounts paying 12%:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME DAT_INC SALARY AND COMPUTE
   INVESTED/D10.2 = SIMPLE(SALARY, DAT_INC, 0.12, INVESTED);
BY EMP_ID
END
```

Note: The subroutine is designed to return only the amount of the investment, not the current date because a subroutine can return only a single value each time it is called.

Naming a Subroutine

A subroutine name can be up to eight characters long unless the language you are using to write the subroutine requires a shorter name. A name must start with a letter and can consist of a combination of letters and/or numbers. Special symbols are not permitted.

Creating Arguments

When you create arguments for a subroutine, you must consider the following issues:

- **Maximum number of arguments.** A subroutine may contain up to 200 arguments. You can bypass this restriction by creating a subroutine that accepts multiple calls, as described in Writing a Subroutine.
- **Argument types.** You can use the same types of arguments in a subroutine as in a function. For details on these argument types, see Supplying an Argument in a Function.
- **Input arguments.** Input arguments are passed to a subroutine using standard conventions. Register one points to the list of arguments.

  You should not assume that input parameters are stored in contiguous memory.
- **Output arguments.** A subroutine returns only one output argument. This argument must be the last in the subroutine. You can choose any format for the output argument except in Dialogue Manager which requires the argument to have the format of the output field.
- **Internal processing.** A subroutine's arguments are processed as follows:
  - An alphanumeric argument is not changed.
  - A numeric argument is converted to floating-point double-precision format except in an operating system RUN command or when storing the output in a variable.
- **Dialogue Manager requirements.** If you are writing a subroutine specifically for Dialogue Manager, the subroutine may need to perform a conversion. For details on using a subroutine with Dialogue Manager, see Calling a Function From a Dialogue Manager Command.
- **COBOL requirements.** All parameters must be defined at the same level in the COBOL FD. Alternatively, you can concatenate all of the parameters into one string,

and break them apart within the subroutine.

The lengths of the calling arguments as defined in FOCUS must match the lengths of the corresponding arguments defined in the subroutine.

Any deviation from these rules may result in problems in using the subroutine. It is recommended that you modify the subroutine to conform to the stated rules and then link it above the line. In order to load subroutines above the line, the following are the required link-edit options for compiling and storing the subroutine:

- AMODE 31 (Addressing Mode - 31-bit addressing)
- RMODE ANY (System can load this routine anywhere)

Language Considerations

When writing a subroutine, you must consider the following language issues:

Language and memory. If you write a subroutine in a language that brings libraries into memory (for example, FORTRAN and COBOL), the libraries reduce the amount of memory available to the subroutine.

FORTRAN. TSO supports FORTRAN input/output operations.

COBOL. When writing a subroutine in COBOL:

- The subroutine must use the GOBACK command to return to the calling program. STOPRUN is not supported.
- Numeric arguments received from a request must be declared as COMP-2 (double precision floating point).
- The format described in the DEFINE or COMPUTE command determines the format of the output argument:

FOCUSFormat	Picture
An	Xn

FOCUSFormat	Picture
`I`	`S9(9) COMP`
`P`	`S9(n)[V9(m)]` where: `(1+n+m)/2 = 8` for small packed numbers. `(1+n+m)/2 = 16` for large packed numbers.
`D`	`COMP-2`
`F`	`COMP-1`

PL/I. When writing a subroutine in PL/I:

- The RETURNS attribute cannot be used.
- The following attribute must be in the procedure (PROC) statement:

```
OPTIONS (COBOL)
```

- Alphanumeric arguments received from a request must be declared as

```
CHARACTER (n)
```

where:

n

Is the field length as defined by the request. Do not use the VARYING attribute.

- Numeric arguments received from a request must be declared as

```
DECIMAL FLOAT (16)
```

Or

```
BINARY FLOAT (53)
```

- The format described in the DEFINE or COMPUTE command determines the format of the output argument:

FOCUSFormat	PL/I Declaration for Output
`An`	`CHARACTER (n)`
`I`	`BINARY FIXED (31)`
`F`	`DECIMAL FLOAT (6) or BINARY FLOAT (21)`
`D`	`DECIMAL FLOAT (16) or BINARY FLOAT (53)`
`P`	`DECIMAL FIXED (15)` (for small packed numbers, 8 bytes) `DECIMAL FIXED (31)` (for large packed numbers, 16 bytes)

- Variables that are not arguments with the STATIC attribute must be declared. This avoids dynamically allocating these variables every time the subroutine is executed.

C language. When writing a subroutine in C:

- Do not return a value with the return statement.
- Declare double-precision fields as Double.
- The format defined in the DEFINE or COMPUTE command determines the format of the output argument:

FOCUSFormat	C Declaration for Output
`An`	`char *xxx n` Alphanumeric fields are not terminated with a null byte and cannot be processed by many of the string manipulation subroutines in the run-time library.
`I`	`long *xxx`
`F`	`float *xxx`
`D`	`double *xxx`
`P`	No equivalent in C.

Programming a Subroutine

Consider the following when planning your programming requirements:

- Write the subroutine to include an argument that specifies the output field.
- If the subroutine initializes a variable, it must initialize it each time it is executed (serial reusability).
- Since a single request may execute a subroutine numerous times, code the subroutine as efficiently as possible.
- If you create your subroutine in a text file or text library, the subroutine must be 31-bit addressable.
- The last argument, which is normally used for returning the result of the subroutine, can also be used to provide input from the subroutine.

You can add flexibility to your subroutine by using a programming technique. A programming technique can be one of the following:

- Executing a subroutine at an entry point. An entry point enables you to use one algorithm to produce different results. For details, see Writing a Subroutine.
- Creating a subroutine with multiple subroutine calls. Multiple calls enable the subroutine to process more than 200 arguments. For details, see Writing a Subroutine.

Executing a Subroutine at an Entry Point

A subroutine is usually executed starting from the first statement. However, a subroutine can be executed starting from any place in the code designated as an *entry point*. This enables a subroutine to use one basic algorithm to produce different results. For example, the DOWK subroutine calculates the day of the week on which a date falls. By specifying the subroutine name DOWK, you obtain a 3-letter abbreviation of the day. If you specify the entry name DOWKL, you obtain the full name. The calculation, however, is the same.

Each entry point has a name. To execute a subroutine at an entry point, specify the entry point name in the subroutine call instead of the subroutine name. How you designate an entry point depends on the language you are using.

Execute a Subroutine at an Entry Point

```
{subroutine|entrypoint}  (input1, input2,...outfield)
```

where:

subroutine

Is the name of the subroutine.

entrypoint

Is the name of the entry point to execute the subroutine at.

input1, input2,...

Are the subroutine's arguments.

outfield

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format.

Executing a Subroutine at an Entry Point

The FTOC subroutine, written in pseudocode below, converts Fahrenheit temperature to Centigrade. The entry point FTOK (designated by the Entry command) sets a flag that causes 273 to be subtracted from the Centigrade temperature to find the Kelvin temperature. The subroutine is:

```
Subroutine FTOC (FAREN, CENTI).
Let FLAG = 0.
Go to label X.
Entry FTOK (FAREN, CENTI).
Let FLAG = 1.
Label X.
Let CENTI = (5/9) * (FAREN - 32).
If FLAG = 1 then CENTI = CENTI - 273.
Return.
End of subroutine.
```

The following is a shorter way to write the subroutine. Notice that the *kelv* output argument listed for the entry point is different from the *centi* output argument listed at the beginning of the subroutine:

```
Subroutine FTOC (FAREN, CENTI).
Entry FTOK (FAREN, KELV).
Let CENTI = (5/9) * (FAREN - 32).
KELV = CENTI - 273.
Return.
End of Subroutine.
```

To obtain the Centigrade temperature, specify the subroutine name FTOC in the subroutine call. The subroutine processes as:

```
CENTIGRADE/D6.2 = FTOC (TEMPERATURE, CENTIGRADE);
```

To obtain the Kelvin temperature, specify the entry name FTOK in the subroutine call. The subroutine processes as:

```
KELVIN/D6.2 = FTOK (TEMPERATURE, KELVIN);
```

Including More Than 200 Arguments in a Subroutine Call

A subroutine can specify a maximum of 200 arguments including the output argument. To process more than 200 arguments, the subroutine must specify two or more call statements to pass the arguments to the subroutine.

Use the following technique for writing a subroutine with multiple calls:

1. Divide the subroutine into segments. Each segment receives the arguments passed by one corresponding subroutine call.

   The argument list in the beginning of your subroutine must represent the same number of arguments in the subroutine call, including a call number argument and an output argument.

   Each call contains the same number of arguments. This is because the argument list in each call must correspond to the argument list in the beginning of the subroutine. You may process some of the arguments as dummy arguments if you have an unequal number of arguments. For example, if you divide 32 arguments among six segments, each segment processes six arguments; the sixth segment processes two arguments and four dummy arguments.

   Subroutines may require additional arguments as determined by the programmer who creates the subroutine.

2. Include a statement at the beginning of the subroutine that reads the call number (first argument) and branches to a corresponding segment. Each segment processes the arguments from one call. For example, number one branches to the first segment, number two to the second segment, and so on.

3. Have each segment store the arguments it receives in other variables (which can be processed by the last segment) or accumulate them in a running total.

   End each segment with a command returning control back to the request (RETURN command).

4. The last segment returns the final output value to the request.

You can also use the entry point technique to write subroutines that process more than 200 arguments. For details, see Writing a Subroutine.

Create a Subroutine With Multiple Call Statements

```
field = subroutine (1, group1, field)
;field = subroutine (2, group2, field);
   .
   .
   .outfield = subroutine (n, groupn, outfield);
```

where:

field

Is the name of the field that contains the result of the segment or the format of the field enclosed in single quotation marks. This field must have the same format as the *outfield*.

Do not specify *field* for the last call statement; use *outfield*.

subroutine

Is the name of the subroutine up to eight characters long.

n

Is a number that identifies each subroutine call. It must be the first argument in each subroutine call. The subroutine uses this call number to branch to segments of code.

group1, group2,...

Are lists of input arguments passed by each subroutine call. Each group contains the same number of arguments, and no more than 26 arguments each.

The final group may contain dummy arguments.

outfield

Is the field that contains the result, or the format of the output value enclosed in single quotation marks.

In Dialogue Manager, you must specify the format.

Creating a Subroutine Divided Into Segments

The ADD32 subroutine, written in pseudocode, sums 32 numbers. It is divided into six segments, each of which adds six numbers from a subroutine call. (The total number of input arguments is 36 but the last four are dummy arguments.) The sixth segment adds two arguments to the SUM variable and returns the result. The sixth segment does not process any values supplied for the four dummy arguments.

The subroutine is:

```
Subroutine ADD32 (NUM, A, B, C, D, E, F, TOTAL).
If NUM is 1 then goto label ONE
else if NUM is 2 then goto label TWO
else if NUM is 3 then goto label THREE
else if NUM is 4 then goto label FOUR
else if NUM is 5 then goto label FIVE
else goto label SIX.

Label ONE.
Let SUM = A + B + C + D + E + F.
Return.

Label TWO
Let SUM = SUM + A + B + C + D + E + F
Return

Label THREE
Let SUM = SUM + A + B + C + D + E + F
Return

Label FOUR
Let SUM = SUM + A + B + C + D + E + F
Return

Label FIVE
Let SUM = SUM + A + B + C + D + E + F
Return
```

```
Label SIX
LET TOTAL = SUM + A + B
Return
End of subroutine
```

To use the ADD32 subroutine, list all six call statements, each call specifying six numbers. The last four numbers, represented by zeros, are dummy arguments. The DEFINE command stores the total of the 32 numbers in the SUM32 field.

```
DEFINE FILE EMPLOYEE
DUMMY/D10 = ADD32 (1, 5, 7, 13, 9, 4, 2, DUMMY);
DUMMY/D10 = ADD32 (2, 5, 16, 2, 9, 28, 3, DUMMY);
DUMMY/D10 = ADD32 (3, 17, 12, 8, 4, 29, 6, DUMMY);
DUMMY/D10 = ADD32 (4, 28, 3, 22, 7, 18, 1, DUMMY);
DUMMY/D10 = ADD32 (5, 8, 19, 7, 25, 15, 4, DUMMY);
SUM32/D10 = ADD32 (6, 3, 27, 0, 0, 0, 0, SUM32);
END
```

Compiling and Storing a Subroutine

After you write a subroutine, you need to compile and store it. This topic discusses compiling and storing your subroutine for z/OS.

Compiling and Storing a Subroutine on z/OS

Compile the subroutine, then link edit it and store the module in a load library. If your subroutine calls other subroutines, compile and link-edit all the subroutines together in a single module. Do not store the subroutine in the FUSELIB load library (FUSELIB.LOAD), as it may be overwritten when your site installs the next release of FOCUS.

If the subroutine is written in PL/I, include the following when link-editing the subroutine

```
ENTRY subroutine
```

where:

subroutine

Is the name of the subroutine.

Compiling and Storing a Subroutine on UNIX

Run the program GENCPGM, which creates a .DLL file. Then check the location of your dynamic link functions library file as specified by the IBICPG environment variable, and save the .DLL file to this location.

Compiling and Storing a Subroutine on Windows

Run the program GENCPGM, which creates a .DLL file. Then check the location of your dynamic link functions library file as specified by the IBICPG environment variable, and save the .DLL file to this location.

Testing the Subroutine

After compiling and storing a subroutine, you can test it in a report request. In order to access the subroutine, you need to issue the ALLOCATE command for z/OS.

If an error occurs during testing, check to see if the error is in the request or in the subroutine.

Determine the Location of Error

You can determine the location of an error with the following:

Procedure

1. Write a dummy subroutine that has the same arguments but returns a constant.

2. Execute the request with the dummy subroutine.

Result

If the request executes the dummy subroutine normally, the error is in your subroutine. If the request still generates an error, the error is in the request.

Using a Custom Subroutine: The MTHNAM Subroutine

This topic discusses the MTHNAM subroutine as an example. The MTHNAM subroutine converts a number representing a month to the full name of that month. The subroutine processes as follows:

1. Receives the input argument from the request as a double-precision number.
2. Adds .000001 to the number which compensates for rounding errors. Rounding errors can occur since floating-point numbers are approximations and may be inaccurate in the last significant digit.
3. Moves the number into an integer field.
4. If the number is less than one or greater than 12, it changes the number to 13.
5. Defines a list containing the names of months and an error message for the number 13.
6. Sets the index of the list equal to the number in the integer field. It then places the corresponding array element into the output argument. If the number is 13, the argument contains the error message.
7. Returns the result as an output field.

Writing the MTHNAM Subroutine

The MTHNAM subroutine can be written in FORTRAN, COBOL, PL/I, BAL Assembler, and C.

MTHNAM Subroutine Written in FORTRAN

This is a FORTRAN version of the MTHNAM subroutine where:

MTH

Is the double-precision number in the input argument.

MONTH

Is the name of the month. Since the character string 'September' contains nine letters, MONTH is a three element array. The subroutine passes the three elements back to your application which concatenates them into one field.

A

Is a two dimensional, 13 by 3 array, containing the names of the months. The last three elements contain the error message.

IMTH

Is the integer representing the month.

The subroutine is:

```
  SUBROUTINE MTHNAM (MTH,MONTH)
  REAL*8     MTH
  INTEGER*4  MONTH(3),A(13,3),IMTH
  DATA
 +     A( 1,1)/'JANU'/, A( 1,2)/'ARY '/, A( 1,3)/'    '/,
 +     A( 2,1)/'FEBR'/, A( 2,2)/'UARY'/, A( 2,3)/'    '/,
 +     A( 3,1)/'MARC'/, A( 3,2)/'H   '/, A( 3,3)/'    '/,
 +     A( 4,1)/'APRI'/, A( 4,2)/'L   '/, A( 4,3)/'    '/,
 +     A( 5,1)/'MAY '/, A( 5,2)/'    '/, A( 5,3)/'    '/,
 +     A( 6,1)/'JUNE'/, A( 6,2)/'    '/, A( 6,3)/'    '/,
 +     A( 7,1)/'JULY'/, A( 7,2)/'    '/, A( 7,3)/'    '/,
 +     A( 8,1)/'AUGU'/, A( 8,2)/'ST  '/, A( 8,3)/'    '/,
 +     A( 9,1)/'SEPT'/, A( 9,2)/'EMBE'/, A( 9,3)/'R   '/,
 +     A(10,1)/'OCTO'/, A(10,2)/'BER '/, A(10,3)/'    '/,
 +     A(11,1)/'NOVE'/, A(11,2)/'MBER'/, A(11,3)/'    '/,
 +     A(12,1)/'DECE'/, A(12,2)/'MBER'/, A(12,3)/'    '/,
 +     A(13,1)/'**ER'/, A(13,2)/'ROR*'/, A(13,3)/'*   '/
  IMTH=MTH+0.000001
  IF (IMTH .LT. 1 .OR. IMTH .GT. 12) IMTH=13
  DO 1 I=1,3
1 MONTH(I)=A(IMTH,I)
```

```
RETURN
END
```

Compiling the FORTRAN Version of MTHNAM Under LE on z/OS

The following example compiles and linkedits the FORTRAN version of MTHNAM:

```
//COMPILE   EXEC  PGM=FORTVS2,
//          PARM='LANGLVL(66),NODECK,NOLIST,OPT(0)'
//*         PARM='NODECK,NOLIST,OPT(0)'
//STEPLIB   DD  DSN=VSF2.VSF2COMP,DISP=SHR
//SYSLIB    DD  DSN=CEE.SCEESAMP,DISP=SHR
//SYSPRINT  DD  SYSOUT=*,DCB=BLKSIZE=3429
//SYSTERM   DD  SYSOUT=*
//SYSPUNCH  DD  SYSOUT=B,DCB=BLKSIZE=3440
//SYSLIN    DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSD
//          SPACE=(3200,(25,6)),DCB=BLKSIZE=3200
//SYSIN     DD  *
/*
/*      The subroutine source code goes here.
/*      Alternatively, your DD statement can point to the data set
/*      containing the source code.
//*
//          COND=(0,NE)
//SYSUT1    DD  DSN=&&LOADSET,DISP=(OLD,PASS)
//SYSUT2    DD  SYSOUT=*
//SYSPRINT  DD  DUMMY
//SYSIN     DD  DUMMY
//*
//LINKEDIT  EXEC  PGM=HEWL,
//          PARM='MAP,LIST,XREF,SIZE=(500K,65K),RMODE(ANY),AMODE(31)',
//          COND=(0,NE)
//SYSPRINT  DD  SYSOUT=*
//SYSLIB    DD  DSN=CEE.SAFHFORT,DISP=SHR
//          DD  DSN=CEE.SCEELKED,DISP=SHR
//SCEESAMP  DD  DSN=CEE.SCEESAMP,DISP=SHR
//SYSUT1    DD  UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSLMOD   DD  DSN=prefix.TSO.LOAD,DISP=SHR
//OBJECT    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//*         DD  DDNAME=SYSIN
//SYSLIN    DD  *
```

```
  INCLUDE OBJECT
  INCLUDE SYSLIB(CEESG007)
  NAME    MTHNAM(R)
/*
//*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

MTHNAM Subroutine Written in COBOL

This is a COBOL version of the MTHNAM subroutine where:

MONTH-TABLE

Is a field containing the names of the months and the error message.

MLINE

Is a 13-element array that redefines the MONTH-TABLE field. Each element (called A) contains the name of a month; the last element contains the error message.

A

Is one element in the MLINE array.

IX

Is an integer field that indexes MLINE.

IMTH

Is the integer representing the month.

MTH

Is the double-precision number in the input argument.

MONTH

Is the name of the month corresponding to the integer in IMTH.

The subroutine is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MTHNAM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
DATA DIVISION.
WORKING-STORAGE SECTION.
   01 MONTH-TABLE.
     05 FILLER PIC X(9) VALUE 'JANUARY  '.
     05 FILLER PIC X(9) VALUE 'FEBRUARY '.
     05 FILLER PIC X(9) VALUE 'MARCH    '.
     05 FILLER PIC X(9) VALUE 'APRIL    '.
     05 FILLER PIC X(9) VALUE 'MAY      '.
     05 FILLER PIC X(9) VALUE 'JUNE     '.
     05 FILLER PIC X(9) VALUE 'JULY     '.
     05 FILLER PIC X(9) VALUE 'AUGUST   '.
     05 FILLER PIC X(9) VALUE 'SEPTEMBER'.
     05 FILLER PIC X(9) VALUE 'OCTOBER  '.
     05 FILLER PIC X(9) VALUE 'NOVEMBER '.
     05 FILLER PIC X(9) VALUE 'DECEMBER '.
     05 FILLER PIC X(9) VALUE '**ERROR**'.
   01  MLIST REDEFINES MONTH-TABLE.
     05  MLINE OCCURS 13 TIMES INDEXED BY IX.
         10 A  PIC X(9).
   01  IMTH    PIC S9(5) COMP.
LINKAGE SECTION.
   01  MTH     COMP-2.
   01  MONTH   PIC X(9).
PROCEDURE DIVISION USING MTH, MONTH.
BEG-1.
      ADD 0.000001 TO MTH.
      MOVE MTH TO IMTH.
      IF IMTH < +1 OR > 12
        SET IX TO +13
      ELSE
        SET IX TO IMTH.
      MOVE A (IX) TO MONTH.
      GOBACK.
```

Compiling the COBOL Version of MTHNAM Under LE on z/OS

The following example compiles and linkedits the COBOL version of MTHNAM:

```
//COMPILE   EXEC  PGM=IGYCRCTL,
//          PARM='APOST,RES,RENT'
//STEPLIB   DD  DSN=IGY.V1R2M0.SIGYCOMP,DISP=SHR

//SYSPRINT  DD  SYSOUT=*
//SYSLIN    DD  DSNAME=&&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),
//          SPACE=(TRK,(3,3))
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSIN     DD  *
/*
/*    The subroutine source code goes here
/*    Alternatively, your DD statement can point to a data set
/*     that contains the source code.
/*
//*
//LINKEDIT  EXEC  PGM=IEWL,
//           PARM='REUS,MAP,LIST'
//STEPLIB   DD  DSN=CEE.SCEELKED,DISP=SHR
//OBJECT    DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//          DD  DDNAME=SYSIN
//SYSLIB    DD  DSN=CEE.SCEELKED,DISP=SHR
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT  DD  SYSOUT=*
//SYSLMOD   DD  DSN=prefix.TSO.LOAD,DISP=SHR
//SYSLIN    DD  *
 MODE AMODE(31),RMODE(ANY)
 INCLUDE OBJECT
 ENTRY MTHNAM
 NAME MTHNAM(R)
/*
//*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

Note:

- RENT is required for reentrancy.
- The linkedit parameter REUSE must be used to make the load module reusable.
- CEEUOPT is automatically included for compiles and linkedits.

MTHNAM Subroutine Written in PL/I

This is a PL/I version of the MTHNAM subroutine where:

MTHNUM

Is the double-precision number in the input argument.

FULLMTH

Is the name of the month corresponding to the integer in MONTHNUM.

MONTHNUM

Is the integer representing the month.

MONTH_TABLE

Is a 13-element array containing the names of the months. The last element contains the error message.

The subroutine is:

```
MTHNAM:   PROC(MTHNUM,FULLMTH) OPTIONS(COBOL);
DECLARE  MTHNUM  DECIMAL FLOAT (16) ;
DECLARE  FULLMTH CHARACTER (9) ;
DECLARE  MONTHNUM FIXED BIN (15,0)  STATIC ;
DECLARE  MONTH_TABLE(13) CHARACTER (9)   STATIC
                        INIT  ('JANUARY',
                               'FEBRUARY',
                               'MARCH',
```

```
                                'APRIL',
                                'MAY',
                                'JUNE',
                                'JULY',
                                'AUGUST',
                                'SEPTEMBER',
                                'OCTOBER',
                                'NOVEMBER',
                                'DECEMBER',
                                '**ERROR**') ;
  MONTHNUM = MTHNUM + 0.00001 ;
  IF MONTHNUM < 1 |  MONTHNUM > 12 THEN
           MONTHNUM = 13 ; FULLMTH = MONTH_TABLE(MONTHNUM) ;
RETURN;
END MTHNAM;
```

Compiling the PL/I Version of MTHNAM Under LE on z/OS

This example includes the following steps for compiling, linkediting, and calling the PL/I version of MTHNAM:

1. Compiling the COBOL stub:

   ```
   //* Step 1 - compile the COBOL stub
   //*
   //COBSTUB   EXEC  IGYWCL,
   //    PARM.COBOL='APOST,DYNAM,RENT',
   //    PARM.LKED='LIST,MAP,SIZE=2046K'
   //COBOL.SYSIN     DD  *
          IDENTIFICATION DIVISION.
          PROGRAM-ID. COBSTUB.
          ENVIRONMENT DIVISION.
          CONFIGURATION SECTION.
          SOURCE-COMPUTER. IBM-370.
          OBJECT-COMPUTER. IBM-370.
          DATA DIVISION.
          WORKING-STORAGE SECTION.
          LINKAGE SECTION.
          01  DUMMY-IO  PIC X(99).
   ```

```
       PROCEDURE DIVISION USING DUMMY-IO.
       MAIN SECTION.
           CALL 'MTHNAM' USING DUMMY-IO.
       MAIN-EXIT.   EXIT.
           GOBACK.
/*
//*
//LKED.SYSIN    DD  *
 MODE AMODE(31),RMODE(ANY)
 ENTRY COBSTUB
 NAME COBSTUB(R)
/*
//LKED.SYSLMOD   DD  DSN=prefix.TSO.LOAD,DISP=SHR
//*
```

2. Compiling the PL/I subroutine:

```
//* Step 2 - compile the PLI program
//*
//COMPILE   EXEC  IEL1CL,
//     PARM.PLI='OBJECT,NODECK',
//     PARM.LKED='XREF,LIST'
//*
//PLI.SYSIN DD  *
/*
/*      The subroutine source code goes here.
/*      Alternatively, your DD statement can point to the data set
/*      containing the source code.
/*
//*
//LKED.SYSLMOD DD  DSN=prefix.TSO.LOAD,DISP=SHR
//LKED.SYSIN   DD  *
   ENTRY  MTHNAM
   NAME   MTHNAM(R)
/*
//*
```

3. Executing FOCUS:

```
//FOCUS     EXEC  PGM=FOCUS,COND=(0,NE)
//STEPLIB   DD  DSN=CEE.SCEERUN,DISP=SHR
//          DD  DSN=prefix.TSO.LOAD,DISP=SHR
//          DD  DSN=prefix.FOCLIB.LOAD,DISP=SHR
//          DD  DSN=prefix.FUSELIB.LOAD,DISP=SHR
```

```
//USERLIB   DD  DSN=prefix.FOCLIB.LOAD,DISP=SHR
//ERRORS    DD  DSN=prefix.ERRORS.DATA,DISP=SHR
//MASTER    DD  DSN=prefix.MASTER.DATA,DISP=SHR
//FOCEXEC   DD  DSN=prefix.FOCEXEC.DATA,DISP=SHR
//EMPLOYEE  DD  DSN=prefix.EMPLOYEE.FOCUS,DISP=SHR
 //SYSOUT    DD  SYSOUT=*
 //SYSPRINT  DD  SYSOUT=*
 //FSTRACE   DD  SYSOUT=*
 //OUT       DD  SYSOUT=*,DCB=BLKSIZE=121
 //OFFLINE   DD  SYSOUT=*
 //*SYSUDUMP  DD  DUMMY
 //SYSIN     DD  *
 SET PRINT = OFFLINE
 DEFINE FILE EMPLOYEE
 MONTH_NUM/M = PAY_DATE;
 PAY_MONTH/A12 = MTHNAM (MONTH_NUM, PAY_MONTH);
 END
 TABLE FILE EMPLOYEE
 HEADING
 "          "
 "FOCUS RELEASE - &FOCREL  PUT LEVEL - " &PUTLEVEL
 "SUBROUTINE - MTHNAM "
 "          "
 PRINT PAY_MONTH GROSS
BY EMP_ID BY FIRST_NAME BY LAST_NAME
BY PAY_DATE
IF LAST_NAME IS STEVENS
END
FIN
/*
//
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

MTHNAM Subroutine Written in BAL Assembler

This is a BAL Assembler version of the MTHNAM subroutine:

```
* ======================================================================
*
*   A SIMPLE MAIN ASSEMBLE ROUTINE THAT CALLS THE LE CALLABLE SERVICES
*
* ======================================================================
MTHNAM   CEEENTRY PPA=MAINPPA,AUTO=WORKSIZE,MAIN=NO
         USING WORKAREA,13
*
         L        3,0(0,1)          LOAD ADDR OF FIRST ARG  INTO R3
         LD       4,=D'0.0'         CLEAR OUT FPR4 AND FPR5
         LE       6,0(0,3)          FP NUMBER IN FPR6
         LPER     4,6               ABS VALUE IN FPR4
         AW       4,=D'0.00001'     ADD ROUNDING CONSTANT
         AW       4,DZERO           SHIFT OUT FRACTION
         STD      4,FPNUM           MOVE TO MEMORY
         L        2,FPNUM+4         INTEGER PART IN R2
         TM       0(3),B'10000000'  CHECK SIGN OF ORIGINAL NO
          BNO      POS              BRANCH IF POSITIVE
          LCR      2,2              COMPLEMENT IF NEGATIVE
 *
 POS      LR       3,2              COPY MONTH NUMBER INTO R3
          C        2,=F'0'          IS IT ZERO OR LESS?
          BNP      INVALID          YES. SO INVALID
          C        2,=F'12'         IS IT GREATER THAN 12?
          BNP      VALID            NO. SO VALID
 INVALID  LA       3,13(0,0)        SET R3 TO POINT TO ITEM 13 (ERROR)
 *
 VALID    SR       2,2               CLEAR OUT R2
          M        2,=F'9'           MULTIPLY BY SHIFT IN TABLE
 *
          LA       6,MTH(3)          GET ADDR OF ITEM IN R6
```

```
          L        4,4(0,1)          GET ADDR OF SECOND ARG IN R4
          MVC      0(9,4),0(6)       MOVE IN TEXT
*
*   TERMINATE THE CEE ENVIRONMENT AND RETURN TO THE CALLER
*
         CEETERM  RC=0
* =====================================================================
*             CONSTANTS
* =====================================================================
         DS      0D                 ALIGNMENT
FPNUM    DS      D                  FLOATING POINT NUMBER
DZERO    DC      X'4E00000000000000' SHIFT CONSTANT
MTH      DC      CL9'DUMMYITEM'     MONTH TABLE
         DC      CL9'JANUARY'
```

```
         DC      CL9'FEBRUARY'
         DC      CL9'MARCH'
         DC      CL9'APRIL'
         DC      CL9'MAY'
         DC      CL9'JUNE'
         DC      CL9'JULY'
         DC      CL9'AUGUST'
         DC      CL9'SEPTEMBER'
         DC      CL9'OCTOBER'
         DC      CL9'NOVEMBER'
         DC      CL9'DECEMBER'
         DC      CL9'**ERROR**'
*
MAINPPA  CEEPPA                   CONSTANTS DESCRIBING THE CODE BLOCK
* ====================================================================
*        THE WORKAREA AND DSA
* ====================================================================
WORKAREA DSECT
         ORG     *+CEEDSASZ       LEAVE SPACE FOR THE DSA FIXED PART
PLIST    DS      0D
PARM1    DS      A
PARM2    DS      A
PARM3    DS      A
PARM4    DS      A
PARM5    DS      A
*
FOCPARM1 DS      F                SAVE FIRST PARAMETER PASSED
FOCPARM2 DS      F                SAVE SECOND PARAMETER PASSED
*
         DS      0D
WORKSIZE EQU     *-WORKAREA
         CEEDSA                   MAPPING OF THE DYNAMIC SAVE AREA
         CEECAA                   MAPPING OF THE COMMON ANCHOR AREA
*
          END    MTHNAM            NOMINATE MTHNAM AS THE ENTRY POINT
/*
```

Assembling the BAL Version of MTHNAM Under LE on z/OS

The following example assembles and linkedits the Assembler version of MTHNAM:

```
//ASSEMBLE  EXEC  PGM=ASMA90,
//          PARM='OBJECT,LIST,ESD,NODECK'
//SYSLIB    DD  DSN=CEE.SCEEMAC,DISP=SHR
//SYSUT1    DD  UNIT=SYSDA,SPACE=(TRK,(45,15))
//SYSUT2    DD  UNIT=SYSDA,SPACE=(TRK,(45,15))
//SYSUT3    DD  UNIT=SYSDA,SPACE=(TRK,(45,15))
//SYSPUNCH  DD  DUMMY
//SYSPRINT  DD  SYSOUT=*
//SYSLIN    DD  DSNAME=&&LINKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//          SPACE=(TRK,(3,3))
//SYSIN     DD  *
/*
/* The subroutine source code goes here.
/* Alternatively, the DD statement can point to a data set that contains
/* the source code.
/*
//*
//IEBGENER  EXEC  PGM=IEBGENER,
//          COND=(0,NE)
//SYSUT1    DD  DSN=&&LINKSET,DISP=(OLD,PASS)
//SYSUT2    DD  SYSOUT=*
//SYSPRINT  DD  DUMMY
//SYSIN     DD  DUMMY
//*
//LINKEDIT  EXEC  PGM=IEWL,
//          PARM='LIST,XREF,LET,REUS',
//          COND=(0,NE)
//SYSLIB    DD  DSN=CEE.SCEELKED,DISP=SHR
//SYSUT1    DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSLMOD   DD  DSN=prefix.TSO.LOAD(MTHNAM),DISP=SHR
//SYSPRINT  DD  SYSOUT=*
//SYSLIN    DD  DSNAME=&&LINKSET,DISP=(OLD,PASS)
//          DD  DDNAME=SYSIN
/*
//SYSIN     DD  *
 ENTRY MTHNAM
 NAME MTHNAM(R)
/*
//*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

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];
}
```

Compiling the C Version of MTHNAM Under LE on z/OS

The following example compiles and linkedits the C version of MTHNAM:

```
//CBG     EXEC PROC=EDCCL
//COMPILE.SYSPRINT DD SYSOUT=*,
//       DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//COMPILE.SYSIN     DD  *,DLM=XX
/*  #INCLUDE <STDIO.H> */
/*
/* The subroutine source code goes here.
```

```
/* Alternatively, the DD statement can point to a data set that contains
/* the source code.
/*
XX
//LKED.SYSPRINT  DD  SYSOUT=*
//LKED.SYSLMOD   DD  DSN=prefix.LOADLIB,DISP=SHR
//LKED.SYSIN  DD  *
  NAME MTHNAM(R)
//*
/*
```

where:

prefix

Is the high-level qualifier for your production FOCUS data sets.

Note:

- A STEPLIB and/or SYSLIB may be required to access C functions.
- The Language Environment has been enhanced to load the required libraries for LE-C upon entering FOCUS.

Calling the MTHNAM Subroutine From a Request

You can call the MTHNAM subroutine from a report request.

Calling the MTHNAM Subroutine

The DEFINE command extracts the month portion of the pay date. The MTHNAM subroutine then converts it into the full name of the month, and stores the name in the PAY_MONTH field. The report request prints the monthly pay of Alfred Stevens.

```
DEFINE FILE EMPLOYEE
MONTH_NUM/M = PAY_DATE;
PAY_MONTH/A12 = MTHNAM (MONTH_NUM, PAY_MONTH);
```

```
END
TABLE FILE EMPLOYEE
PRINT PAY_MONTH GROSS
BY EMP_ID BY FIRST NAME BY LAST_NAME
BY PAY_DATE
IF LN IS STEVENS
END
```

The output is:

```
EMP_ID     FIRST NAME   LAST_NAME   PAY_DATE   PAY_MONTH   GROSS
-------    ----------   ---------   --------   ---------   -------
071382660  ALFRED       STEVENS     81/11/30   NOVEMBER    $833.33
                                    81/12/31   DECEMBER    $833.33
                                    82/01/29   JANUARY     $916.67
                                    82/02/26   FEBRUARY    $916.67
                                    82/03/31   MARCH       $916.67
                                    82/04/30   APRIL       $916.67
                                    82/05/28   MAY         $916.67
                                    82/06/30   JUNE        $916.67
                                    82/07/30   JULY        $916.67
                                    82/08/31   AUGUST      $916.67
```

Subroutines Written in REXX

A request can call a subroutine coded in REXX. These subroutines, also called FUSREXX macros, provide a 4GL option to the languages supported for user-written subroutines.

REXX subroutines are supported in the z/OS environment. A REXX subroutine contains REXX source code. Compiled REXX code is not supported.

REXX subroutines are not necessarily the same in all operating environments. Therefore, some of the examples may use REXX functions that are not available in your environment.

Because of CPU requirements, the use of REXX subroutines in large production jobs should be monitored carefully.

For more information on REXX subroutines, see your REXX documentation.

Storing and Searching for a REXX Subroutine

To store a REXX subroutine, DDNAME FUSREXX must be allocated to a PDS. This library is searched before other z/OS libraries.

The search order for a REXX subroutine is:

1. FUSREXX.
2. Standard z/OS search order.

Call a REXX Subroutine

```
DEFINE FILE filename
fieldname/{An|In} = subname(inlen1, inparm1, ..., outlen, outparm);
END
```

or

```
{DEFINE|COMPUTE} fieldname/{An|In} = subname(inlen1, inparm1, ...,
outlen, outparm);
```

or

```
-SET &var = subname(inlen1, inparm1, ..., outlen, outparm);
```

where:

fieldname

Is the field that contains the result.

An, In

Is the format of the field that contains the result.

subname

Is the name of the REXX subroutine.

inlen1, inparm1 ...

Are the input parameters. Each parameter consists of a length and an alphanumeric parameter value. You can supply the value, the name of an alphanumeric field that contains the value, or an expression that returns the value. Up to 13 input parameter pairs are supported. Each parameter value can be up to 256 bytes long.

Dialogue Manager converts numeric arguments to floating-point double-precision format. Therefore, you can only pass alphanumeric input parameters to a REXX subroutine using -SET.

outlen, outparm

Is the output parameter pair, consisting of a length and a result. In most cases, the result should be alphanumeric, but integer results are also supported. The result can be a field or a Dialogue Manager variable that contains the value, or the format of the value enclosed in single quotation marks. The return value can be a minimum of one byte long and a maximum (for an alphanumeric value) of 256 bytes.

Note: If the value returned is an integer, the *outlen* must be 4 because FOCUS reserves four bytes for integer fields.

&var

Is the name of the Dialogue Manager variable that contains the result.

Returning the Day of the Week

The REXX subroutine DOW returns the day of the week corresponding to the date an employee was hired. The routine contains one input parameter pair and one return field pair.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE) ;
2. DAY_OF_WEEK/A9 WITH AHDT = DOW(6, AHDT, 9, DAY_OF_WEEK);
   END
```

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAY_OF_WEEK
END
```

The procedure processes as follows:

1. The EDIT function converts HIRE_DATE to alphanumeric format and stores the result in a field with the format A6.
2. The result is stored in the DAY_OF_THE_WEEK field, and can be up to nine bytes long.

The output is:

```
LAST_NAME        HIRE_DATE  DAY_OF_WEEK
---------        ---------  -----------
STEVENS           80/06/02  Monday
SMITH             81/07/01  Wednesday
JONES             82/05/01  Saturday
SMITH             82/01/04  Monday
BANNING           82/08/01  Sunday
IRVING            82/01/04  Monday
ROMANS            82/07/01  Thursday
MCCOY             81/07/01  Wednesday
BLACKWOOD         82/04/01  Thursday
MCKNIGHT          82/02/02  Tuesday
GREENSPAN         82/04/01  Thursday
CROSS             81/11/02  Monday
```

The REXX subroutine appears below. It reads the input date, reformats it to MM/DD/YY format, and returns the day of the week using a REXX DATE call.

```
/* DOW routine. Return WEEKDAY from YYMMDD format date */
Arg ymd .
Return Date('W',Translate('34/56/12',ymd,'123456'),'U')
```

Passing Multiple Arguments to a REXX Subroutine

The REXX subroutine INTEREST has four input parameters.

```
DEFINE FILE EMPLOYEE
1. AHDT/A6     = EDIT(HIRE_DATE);
2. ACSAL/A12   = EDIT(CURR_SAL);
3. DCSAL/D12.2 = CURR_SAL;
4. PV/A12    = INTEREST(6, AHDT, 6, '&YMD', 3, '6.5', 12, ACSAL, 12,
PV);
```

```
   END

TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE DCSAL PV
END
```

The procedure processes as follows:

1. EDIT converts HIRE_DATE to alphanumeric format and stores the result in AHDT.
2. EDIT converts CURR_SAL to alphanumeric format and stores the result in ACSAL.
3. CURR_SAL is converted to a floating-point double-precision field that includes commas, and the result is stored in DCSAL.
4. The second input field is six bytes long. Data is passed as a character variable &YMD in YYMMDD format.

   The third input field is a character value of 6.5, which is three bytes long to account for the decimal point in the character string.

   The fourth input field is 12 bytes long. This passes the character field ACSAL.

   The return field is up to 12 bytes long and is named PV.

The output is:

```
LAST_NAME        FIRST_NAME  HIRE_DATE            DCSAL            PV
---------        ----------  ---------            -----            --
STEVENS          ALFRED       80/06/02        11,000.00      14055.14
SMITH            MARY         81/07/01        13,200.00      15939.99
JONES            DIANE        82/05/01        18,480.00      21315.54
SMITH            RICHARD      82/01/04         9,500.00      11155.60
BANNING          JOHN         82/08/01        29,700.00      33770.53
IRVING           JOAN         82/01/04        26,862.00      31543.35
ROMANS           ANTHONY      82/07/01        21,120.00      24131.19
MCCOY            JOHN         81/07/01        18,480.00      22315.99
BLACKWOOD        ROSEMARIE    82/04/01        21,780.00      25238.25
MCKNIGHT         ROGER        82/02/02        16,100.00      18822.66
GREENSPAN        MARY         82/04/01         9,000.00      10429.03
CROSS            BARBARA      81/11/02        27,062.00      32081.82
```

The REXX subroutine appears below. The REXX Format command is used to format the return value.

```
/* Simple INTEREST program. dates are yymmdd format  */
Arg start_date,now_date,percent,open_balance, .

begin = Date('B',Translate('34/56/12',start_date,'123456'),'U')
stop  = Date('B',Translate('34/56/12',now_date,'123456'),'U')
valnow = open_balance * (((stop - begin) * (percent / 100)) / 365)

Return Format(valnow,9,2)
```

Accepting Multiple Tokens in a Parameter

A REXX subroutine can accept multiple tokens in a parameter. The following procedure passes employee information (PAY_DATE and MO_PAY) as separate tokens in the first parameter. It passes three input parameters and one return field.

```
DEFINE FILE EMPLOYEE
1.  COMPID/A256 = FN | ' ' | LN | ' ' |  DPT | ' ' | EID ;
2.  APD/A6 = EDIT(PAY_DATE);
3.  APAY/A12 = EDIT(MO_PAY);
4.  OK4RAISE/A1 = OK4RAISE(256, COMPID, 6, APD, 12, APAY, 1, OK4RAISE);
    END

TABLE FILE EMPLOYEE
PRINT EMP_ID FIRST_NAME LAST_NAME DEPARTMENT
IF OK4RAISE EQ '1'
END
```

The procedure processes as follows:

1. COMPID is the concatenation of several character fields passed as the first parameter and stored in a field with the format A256. Each of the other parameters is a single argument.
2. EDIT converts PAY_DATE to alphanumeric format.
3. EDIT converts MO_PAY to alphanumeric format.
4. OK4RAISE executes, and the result is stored in OK4RAISE.

The output is:

```
EMP_ID     FIRST_NAME       LAST_NAME     DEPARTMENT
------     ----------       ---------     ----------
071382660  ALFRED           STEVENS       PRODUCTION
```

The REXX subroutine appears below. Commas separate FUSREXX parameters. The ARG command specifies multiple variable names before the first comma and, therefore, separates the first FUSREXX parameter into separate REXX variables, using blanks as delimiters between the variables.

```
/* OK4RAISE routine. Parse separate tokens in the 1st parm, */
/* then more parms */

Arg fname lname dept empid,  pay_date,  gross_pay, .

If dept = 'PRODUCTION' & pay_date < '820000'
Then retvalue = '1'
Else retvalue = '0'

Return retvalue
```

REXX subroutines should use the REXX RETURN subroutine to return data. REXX EXIT is acceptable, but is generally used to end an EXEC, not a FUNCTION.

```
Correct
/* Some FUSREXX function */
Arg input
some rexx process ..
Return data_to_FOCUS
```

```
Not as Clear
/* Another FUSREXX function */
Arg input
some rexx process ...
Exit 0
```

Formats and REXX Subroutines

A REXX subroutine requires input data to be in alphanumeric format. Most output is returned in alphanumeric format. If the format of an input argument is numeric, use the EDIT or FTOA functions to convert the argument to alphanumeric. You can then use the EDIT or ATODBL functions to convert the output back to numeric.

The output length in the subroutine call must be four. Character variables cannot be more than 256 bytes. This limit also applies to REXX subroutines. FUSREXX routines return variable length data. For this reason, you must supply the length of the input arguments and the maximum length of the output data.

A REXX subroutine does not require any input parameters, but requires one return parameter, which must return at least one byte of data. It is possible for a REXX subroutine not to need input, such as a function that returns USERID.

A REXX subroutine does not support FOCUS date input arguments. When working with dates you can do one of the following:

- Pass an alphanumeric field with date display options and have the subroutine return a date value.

  Date fields contain the integer number of days since the base date 12/31/1900. REXX has a date function that can accept and return several types of date formats, including one called Base format ('B') that contains the number of days since the REXX base date 01/01/0001. You must account for the difference, in number of days, between the FOCUS base date and the REXX base date and convert the result to integer.

- Pass a date value converted to alphanumeric format. You must account for the difference in base dates for both the input and output arguments.

Returning a Result in Alphanumeric Format

The NUMCNT subroutine returns the number of copies of each classic movie in alphanumeric format. It passes one input parameter and one return field.

```
TABLE FILE MOVIES
   PRINT TITLE AND COMPUTE
1. ACOPIES/A3 = EDIT(COPIES); AS 'COPIES'
   AND COMPUTE
2. TXTCOPIES/A8 = NUMCNT(3, ACOPIES, 8, TXTCOPIES);
   WHERE CATEGORY EQ 'CLASSIC'
   END
```

The procedure processes as follows:

1. The EDIT field converts COPIES to alphanumeric format, and stores the result in ACOPIES.
2. The result is stored in an 8-byte alphanumeric field TXTCOPIES.

The output is:

```
TITLE                                      COPIES  TXTCOPIES
-----                                      ------  ---------
EAST OF EDEN                               001     One
CITIZEN KANE                               003     Three
CYRANO DE BERGERAC                         001     One
MARTY                                      001     One
MALTESE FALCON, THE                        002     Two
GONE WITH THE WIND                         003     Three
ON THE WATERFRONT                          002     Two
MUTINY ON THE BOUNTY                       002     Two
PHILADELPHIA STORY, THE                    002     Two
CAT ON A HOT TIN ROOF                      002     Two
CASABLANCA                                 002     Two
```

The subroutine is:

```
/* NUMCNT routine. */
/* Pass a number from 0 to 10 and return a character value */
Arg numbr .
data = 'Zero One Two Three Four Five Six Seven Eight Nine Ten'
numbr = numbr + 1           /* so 0 equals 1 element in array */
Return Word(data,numbr)
```

Returning a Result in Integer Format

In the following example, the NUMDAYS subroutine finds the number of days between HIRE_DATE and DAT_INC and returns the result in integer format.

```
   DEFINE FILE EMPLOYEE
1. AHDT/A6 = EDIT(HIRE_DATE);
2. ADI/A6 = EDIT(DAT_INC);
3. BETWEEN/I6 = NUMDAYS(6, AHDT, 6, ADI, 4, 'I6') ;
   END

TABLE FILE EMPLOYEE
PRINT LAST_NAME HIRE_DATE DAT_INC BETWEEN
IF BETWEEN NE 0
END
```

The procedure processes as follows:

1. EDIT converts HIRE_DATE to alphanumeric format and stores the result in AHDT.
2. EDIT converts DAT_INC to alphanumeric format and stores the result in ADI.
3. NUMDAYS finds the number of days between AHDT and ADI and stores the result in integer format.

The output is:

```
LAST_NAME       HIRE_DATE   DAT_INC  BETWEEN
---------       ---------   -------  -------
STEVENS         80/06/02    82/01/01     578
STEVENS         80/06/02    81/01/01     213
SMITH           81/07/01    82/01/01     184
JONES           82/05/01    82/06/01      31
SMITH           82/01/04    82/05/14     130
IRVING          82/01/04    82/05/14     130
MCCOY           81/07/01    82/01/01     184
MCKNIGHT        82/02/02    82/05/14     101
GREENSPAN       82/04/01    82/06/11      71
CROSS           81/11/02    82/04/09     158
```

The subroutine appears below. The return value is converted from REXX character to HEX and formatted to be four bytes long.

```
/* NUMDAYS routine.  */
/* Return number of days between 2 dates in yymmdd format */
/* The value returned will be in hex format               */

Arg first,second .

base1 = Date('B',Translate('34/56/12',first,'123456'),'U')
base2 = Date('B',Translate('34/56/12',second,'123456'),'U')

Return D2C(base2 - base1,4)
```

Passing a Date Value as an Alphanumeric Field With Date Options

In the following example, a date is used by passing an alphanumeric field with date options to the DATEREX1 subroutine. DATEREX1 takes two input arguments: an alphanumeric date in A8YYMD format and a number of days in character format. It returns a smart date in

YYMD format that represents the input date plus the number of days. The format A8YYMD corresponds to the REXX Standard format ('S').

The number 693959 represents the difference, in number of days, 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
```

Passing a Date as a Date Converted to Alphanumeric Format

In the following example, a date is passed to the subroutine as a smart date converted to alphanumeric format. The DATEREX2 subroutine takes two input arguments: an alphanumeric number of days that represents a smart date, and a number of days to add. It returns a smart date in YYMD format that represents the input date plus the number of days. Both the input date and output date are in REXX base date ('B') format.

The number 693959 represents the difference, in number of days, between the FOCUS base date and the REXX base date:

```
/* REXX DATEREX2 routine. Add indate (original format YYMD) to days */
Arg indate, days .
Return D2C(Date('B',indate+693959,'B') + days - 693959, 4)
```

The following request uses DATEREX2 to calculate the date that is 365 days from the hire date of each employee. The input arguments are the hire date and the number of days to add. Because HIRE_DATE is in I6YMD format, it must be converted to an alphanumeric number of days before being passed to the macro:

```
TABLE FILE EMPLOYEE
PRINT LAST_NAME FIRST_NAME HIRE_DATE
AND COMPUTE
  ADATE/YYMD =  HIRE_DATE; NOPRINT
AND COMPUTE
  INDATE/A8 = EDIT(ADATE); NOPRINT
AND COMPUTE
  NEXT_DATE/YYMD = DATEREX2(8,INDATE,3,'365',4,NEXT_DATE);
BY LAST_NAME NOPRINT
END
```

The output is:

```
LAST_NAME        FIRST_NAME  HIRE_DATE  NEXT_DATE
---------        ----------  ---------  ---------
BANNING          JOHN         82/08/01  1983/08/01
BLACKWOOD        ROSEMARIE    82/04/01  1983/04/01
CROSS            BARBARA      81/11/02  1982/11/02
```

```
GREENSPAN        MARY         82/04/01  1983/04/01
IRVING           JOAN         82/01/04  1983/01/04
JONES            DIANE        82/05/01  1983/05/01
MCCOY            JOHN         81/07/01  1982/07/01
MCKNIGHT         ROGER        82/02/02  1983/02/02
ROMANS           ANTHONY      82/07/01  1983/07/01
SMITH            MARY         81/07/01  1982/07/01
SMITH            RICHARD      82/01/04  1983/01/04
STEVENS          ALFRED       80/06/02  1981/06/02
```

ASCII and EBCDIC Codes

The table in this appendix lists ASCII and EBCDIC codes for printable and non-printable characters.

ASCII and EBCDIC Code Chart

This chart shows the standard ASCII characters in numerical order with the corresponding decimal and hexadecimal values.

Decimal	Hex	ASCII		EBCDIC	
0	00	NUL	null	NUL	null
1	01	SOH	start of heading	SOH	start of heading
2	02	STX	start of text	STX	start of text
3	03	ETX	end of text	ETX	end of text
4	04	EOT	end of transmission	SEL	select
5	05	ENQ	enquiry	HT	horizontal tab
6	06	ACK	acknowledge	RNL	required new-line
7	07	BEL	bell	DEL	delete
8	08	BS	backspace	GE	graphic escape
9	09	HT	horizontal tab	SPS	superscript
10	0A	LF	line feed	RPT	repeat

Decimal	Hex	ASCII		EBCDIC	
11	0B	VT	vertical tab	VT	vertical tab
12	0C	FF	form feed	FF	form feed
13	0D	CR	carriage return	CR	carriage return
14	0E	SO	shift out	SO	shift out
15	0F	SI	shift in	SI	shift in
16	10	DLE	data link escape	DLE	data link escape
17	11	DC1	device control 1	DC1	device control 1
18	12	DC2	device control 2	DC2	device control 2
19	13	DC3	device control 3	DC3	device control 3
20	14	DC4	device control 4	RES/ ENP	restore/enable presentation
21	15	NAK	negative acknowledge	NL	new-line
22	16	SYN	synchronous idle	BS	backspace
23	17	ETB	end of transmission block	POC	program-operator communications
24	18	CAN	cancel	CAN	cancel
25	19	EM	end of medium	EM	end of medium
26	1A	SUB	substitute	UBS	unit backspace
27	1B	ESC	escape	CU1	customer use 1
28	1C	FS	file separator	IFS	interchange file

Decimal	Hex	ASCII		EBCDIC	
					separator
29	1D	GS	group separator	IGS	interchange group separator
30	1E	RS	record separator	IRS	interchange record separator
31	1F	US	unit separator	IUS/ ITB	interchange unit separator / intermediate transmission block
32	20	SP	space	DS	digit select
33	21	!	exclamation point	SOS	start of significance
34	22	"	straight double quotation mark	FS	field separator
35	23	#	number sign	WUS	word underscore
36	24	$	dollar sign	BYP/ INP	bypass/inhibit presentation
37	25	%	percent sign	LF	line feed
38	26	&	ampersand	ETB	end of transmission block
39	27	'	apostrophe	ESC	escape
40	28	(	left parenthesis	SA	set attribute
41	29	)	right parenthesis		
42	2A	*	asterisk	SM/ SW	set model switch

Decimal	Hex	ASCII		EBCDIC	
43	2B	+	addition sign	CSP	control sequence prefix
44	2C	,	comma	MFA	modify field attribute
45	2D	-	subtraction sign	ENQ	enquiry
46	2E	.	period	ACK	acknowledge
47	2F	/	right slash	BEL	bell
48	30	0	0		
49	31	1	1		
50	32	2	2	SYN	synchronous idle
51	33	3	3	IR	index return
52	34	4	4	PP	presentation position
53	35	5	5	TRN	
54	36	6	6	NBS	numeric backspace
55	37	7	7	EOT	end of transmission
56	38	8	8	SBS	subscript
57	39	9	9	IT	indent tab
58	3A	:	colon	RFF	required form feed
59	3B	;	semicolon	CU3	customer use 3
60	3C	<	less-than	DC4	device control 4
61	3D	=	equal	NAK	negative acknowledge

Decimal	Hex	ASCII		EBCDIC	
62	3E	>	greater-than		
63	3F	?	question mark	SUB	substitute
64	40	@	at symbol	SP	space
65	41	A	A		
66	42	B	B		
67	43	C	C		
68	44	D	D		
69	45	E	E		
70	46	F	F		

Decimal	Hex	ASCII		EBCDIC	
71	47	G	G		
72	48	H	H		
73	49	I	I		
74	4A	J	J	¢	cent
75	4B	K	K	.	period
76	4C	L	L	<	less-than
77	4D	M	M	(	left parenthesis
78	4E	N	N	+	addition sign

Decimal	Hex	ASCII		EBCDIC	
79	4F	O	O	|	logical or
80	50	P	P	&	ampersand
81	51	Q	Q		
82	52	R	R		
83	53	S	S		
84	54	T	T		
85	55	U	U		
86	56	V	V		
87	57	W	W		
88	58	X	X		
89	59	Y	Y		
90	5A	Z	Z	!	exclamation mark
91	5B	[	left bracket	$	dollar sign
92	5C	\	left slant	*	asterisk
93	5D	]	right bracket	)	right parenthesis
94	5E	^	hat, circumflex	;	semicolon
95	5F	_	underscore	¬	logical not
96	60	`	grave	-	subtraction sign
97	61	a	a	/	right slash

Decimal	Hex	ASCII		EBCDIC	
98	62	b	b		
99	63	c	c		
100	64	d	d		
101	65	e	e		
102	66	f	f		
103	67	g	g		
104	68	h	h		
105	69	i	i		
106	6A	j	j	|	split vertical bar
107	6B	k	k	,	comma
108	6C	l	l	%	percent sign
109	6D	m	m	_	underscore
110	6E	n	n	>	greater-than
111	6F	o	o	?	question mark
112	70	p	p		
113	71	q	q		
114	72	r	r		
115	73	s	s		
116	74	t	t		

Decimal	Hex	ASCII		EBCDIC	
117	75	u	u		
118	76	v	v		
119	77	w	w		
120	78	x	x		
121	79	y	y	`	grave
122	7A	z	z	:	colon
123	7B	{	opening brace	#	number sign
124	7C	|	vertical line	@	at symbol
125	7D	}	closing brace	'	apostrophe
126	7E	~	tilde	=	equal
127	7F			"	straight double quotation mark
128	80				
129	81			a	a
130	82			b	b
131	83			c	c
132	84			d	d
133	85			e	e
134	86			f	f

Decimal	Hex	ASCII		EBCDIC	
135	87			g	g
136	88			h	h
137	89			i	i
138	8A				
139	8B				
140	8C				
141	8D				
142	8E				
143	8F				
144	90				
145	91			j	j
146	92			k	k
147	93			l	l
148	94			m	m
149	95			n	n
150	96			o	o
151	97			p	p
152	98			q	q
153	99			r	r

Decimal	Hex	ASCII		EBCDIC	
154	9A				
155	9B				
156	9C				
157	9D				
158	9E				
159	9F				
160	A0				
161	A1			~	similar, tilde
162	A2			s	s
163	30			t	t
164	A4			u	u
165	A5			v	v
166	A6			w	w
167	A7			x	x
168	A8			y	y
169	A9			z	z
170	AA				
171	AB				
172	AC				

Decimal	Hex	ASCII		EBCDIC	
173	AD				
174	AE				
175	AF				
176	B0				
177	B1				
178	B2				
179	B3				
180	B4				
181	B5				
182	B6				
183	B7				
184	B8				
185	B9				
186	BA				
187	BB				
188	BC				
189	BD				

Decimal	Hex	ASCII		EBCDIC	
190	BE				
191	BF				
192	C0			{	left brace
193	C1			A	A
194	C2			B	B
195	C3			C	C
196	C4			D	D
197	C5			E	E
198	C6			F	F
199	C7			G	G
200	C8			H	H
201	C9			I	I
202	CA				
203	CB				
204	CC				
205	CD				
206	CE				
207	CF				
208	D0			}	right brace

Decimal	Hex	ASCII		EBCDIC	
209	D1			J	J
210	D2			K	K
211	D3			L	L
212	D4			M	M
213	D5			N	N
214	D6			O	O
215	D7			P	P
216	D8			Q	Q
217	D9			R	R
218	DA				
219	DB				
220	DC				
221	DD				
222	DE				
223	DF				
224	E0			\	left slash
225	E1				
226	E2			S	S
227	E3			T	T

Decimal	Hex	ASCII		EBCDIC	
228	E4			U	U
229	E5			V	V
230	E6			W	W
231	E7			X	X
232	E8			Y	Y
233	E9			Z	Z
234	EA				
235	EB				
236	EC				
237	ED				
238	EE				
239	EF				
240	F0			0	0
241	F1			1	1
242	F2			2	2
243	F3			3	3
244	F4			4	4
245	F5			5	5
246	F6			6	6

Decimal	Hex	ASCII		EBCDIC	
247	F7			7	7
248	F8			8	8
249	F9			9	9
250	FA			|	vertical line
251	FB				
252	FC				
253	FD				
254	FE				
255	FF			EO	eight ones

ibi Documentation and Support Services

For information about this product, you can read the documentation, contact Support, and join Community.

How to Access ibi Documentation

Documentation for ibi products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the ibi™ FOCUS® Documentation page.

How to Contact Support for ibi Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join ibi Community

ibi Community is the official channel for ibi customers, partners, and employee subject matter experts to share and access their collective experience. ibi Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from ibi products. For a free registration, go to ibi Community.

Legal and Third-Party Notices

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the "readme" file for the availability of a specific version of Cloud SG software on a specific operating system platform.