Db2 Web Query Functions

Release 2.3

Contents

Chapter 1

Using Functions

The following topics offer an introduction to functions and explain how to use them. Functions provide a convenient way to perform certain calculations and manipulations.

In order to use any of the specified functions described in this reference manual, you need to create a temporary field which is discussed later.

In this chapter:

Types of Functions

You can access any of the following types of functions:

- ❏ **Character functions.** Manipulate alphanumeric fields or character strings.
- ❏ **Simplifed character functions.** Provide streamlined parameter lists, similar to those used by SQL functions.
- ❏ **Data source and decoding functions.** Retrieve data source values and assign values based on the value of an input field.
- ❏ **Date and time functions.** Manipulate dates and times.
- ❏ **Simplified date and time functions.** Have streamlined parameter lists, similar to those used by SQL functions.

- ❑ **Format conversion functions.** Convert fields from one format to another.
- ❑ **Numeric functions.** Perform calculations on numeric constants and fields.
- ❑ **System functions.** Call the operating system to obtain information about the operating environment.

Character Functions

The following functions manipulate alphanumeric fields or character strings.

ARGLEN

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

BITSON

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

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 by converting every alphanumeric character to lowercase except the first letter of each new word.

LCWORD3

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.

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.

PTOA

Converts a 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.

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.

SPELLNUM

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

SUBSTR

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

UPCASE

Converts a character string to uppercase.

Simplified Character Functions

The following character functions have streamlined parameter lists, similar to those used by SQL functions.

CHAR_LENGTH

Returns the length in characters of a string.

DIGITS

Converts a number to a character string.

LOWER

Returns a sring with all letters lowercase.

LPAD

Left-pads a character string.

LTRIM

Removes blanks from the left end of a string.

POSITION

Returns the first position of a substring in a source string.

RTRIM

Removes blanks from the right end of a string.

RPAD

Right-pads a character string.

SUBSTRING

Extracts a substring from a source string.

TOKEN

Extracts a token from a string.

TRIM_

Removes leading characters, trailing characters, or both from a string.

UPPER

Returns a string with all letters uppercase.

Data Source and Decoding Functions

The following functions retrieve data source values and assign values.

DB_EXPR

Inserts a native SQL expression exactly as entered into the native SQL generated for a Web Query or SQL language request.

DECODE

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

LAST

Retrieves the preceding value for a field.

Date and Time Functions

The following functions manipulate dates and times.

Standard Date and Time 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

Converts 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.

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.

DATEPATTERN

Stores date values in alphanumeric format without any particular standard, with any combination of components such as year, quarter, and month, and with any delimiter.

DATETRAN

Formats dates in international formats.

DMY, MDY, and YMD

Calculates the difference between two dates.

DOWK and DOWKL

Finds 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.

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 give calendar date.

GREGDT

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

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.

HGETC

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

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.

JULDAT

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

TIMETOTS

Converts a time to a timestamp.

TODAY

Retrieves the current date from the system.

YM

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

Simplified Date and Date-Time Functions

The following functions have streamlined parameter lists, similar to those used by SQL functions.

DTADD

Increments a date or date-time component.

DTDIFF

Returns a number of component boundaries between date or date-time values.

DTPART

Reurns a date or date-time component in integer format.

DTRUNC

Returns the start of a date period for a given date.

Format Conversion Functions

The following functions convert fields from one format to another.

ATODBL

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

EDIT

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

FTOA

Converts a number in a numeric format to alphanumeric format.

HEXBYT

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

ITONUM

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

ITOPACK

Converts a large binary integer in a 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.

Numeric Functions

The following functions perform calculations on numeric constants or fields.

ABS

Returns the absolute value of a number.

BAR

Produces a horizontal bar chart.

CHKPCK

Validates the data in a field described as packed format.

DMOD, FMOD, and IMOD

Calculates the remainder from a division.

EXP

Raises the number "e" to a specified power.

INT

Returns the integer component of a number.

LOG

Returns the natural logarithm of a number.

MAX and MIN

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

PRDNOR and PRDUNI

Generates reproducible random numbers.

RDNORM and RDUNIF

Generates random numbers.

SQRT

Calculates the square root of a number.

System Functions

The following functions call the operating system to obtain information about the operating environment.

FGETENV

Retrieves the value of an environment variable and returns it as an alphanumeric string.

GETUSER

Retrieves the ID of the connected user.

HHMMSS

Retrieves the current time from the system.

TODAY

Retrieves the current date from the system.

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, or date 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
  ```
- Format of the output value enclosed in single quotation marks.
- Another function.

Increased Number of Function Arguments

The number of arguments supported for user-written subroutines has been increased from 28 to 200. All other rules regarding arguments for user-written subroutines remain the same.

Argument Formats

Depending on the function, an argument can be in alphanumeric, numeric, or date format. If you supply an argument in the wrong format, you will cause an error or the function will not return correct data. To obtain the valid formats, click the Format button on the tool for a list of possible types and lengths. 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.

- **Numeric argument.** A numeric argument is stored internally as a binary or packed number. A numeric argument includes integer (I), floating-point single-precision (F), floating-point double-precision (D), and packed-decimal (P) formats. A numeric argument can be a numeric constant, field, or expression, or the format of a numeric field. All numeric arguments are converted to floating-point double-precision format when used with a function, but results are returned in the format specified for the output field.
- **Date argument.** A date argument can be in either alphanumeric, numeric, or date format. The list of arguments for the individual function will specify what type of format the function accepts. A date argument can be a date in alphanumeric, numeric, or date format; a date field or expression; or the format of a date field. If you supply an argument with a two-digit year, the function assigns a century based on the DATEFNS, YRTHRESH, and DEFCENT parameter settings.

Chapter 2

Character Functions

Character functions manipulate alphanumeric fields and character strings.

In this chapter:

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.

Syntax: How to Measure the Length of a Character String

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

where:

inlength
: Integer

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

infield
: Alphanumeric

  Is the name of the field containing the character string.

outfield
: Integer

  Is the format of the output value enclosed in single quotation marks.

Example: Measuring the Length of a Character String

ARGLEN determines the length of the character string in LAST_NAME and stores the result in NAME_LEN:

```
COMPUTE NAME_LEN/I3 = ARGLEN(15, LAST_NAME, 'I3');
```

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.

Syntax: How to Determine If a Bit Is On or Off

```
BITSON(bitnumber, string, 'outfield')
```

where:

bitnumber
: Integer

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

string
: Alphanumeric

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

outfield
Integer

Is the format of the output value enclosed in single quotation marks.

Example: **Evaluating a Bit in a Field**

BITSON evaluates the 24th bit of LAST_NAME and stores the result in BIT_24:

```
COMPUTE BIT_24/I1 = BITSON(24, LAST_NAME, 'I1');
```

BYTVAL: Translating a Character to a Decimal Value

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

Syntax: **How to Translate a Character**

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

where:

character
Alphanumeric

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

outfield
Integer

Is the format of the output value enclosed in single quotation marks.

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

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

```
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3');
```

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 are returned as a no match with CHKFMT giving the first non-matching position as the result.

Syntax:

How to Check the Format of a Character String

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

where:

numchar

Integer

Is the number of characters being compared to the mask.

string

Alphanumeric

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

mask

Alphanumeric

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

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

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

9 is any digit between 0-9.

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

$ is any character.

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

outfield

Integer

Is the format of the output value enclosed in single quotation marks.

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

CHKFMT examines EMP_ID for nine numeric characters starting with 11 and stores the result in CHK_ID:

```
COMPUTE CHK_ID/I3 = CHKFMT(9, EMP_ID, '119999999', 'I3');
```

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.

To use CTRAN, you must know the decimal equivalent of the characters in internal machine representation.

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

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

Syntax: **How to Translate One Character to Another**

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

where:

charlen

Integer

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

string

Alphanumeric

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

decimal

Integer

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

decvalue

Integer

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Translating Spaces to Underscores on an EBCDIC Platform

CTRAN translates the spaces in ADDRESS_LN3 (EBCDIC decimal value 64) to underscores (EBCDIC decimal value 109) and stores the result in ALT_ADDR:

```
COMPUTE ALT_ADDR/A20 = CTRAN(20, ADDRESS_LN3, 64, 109, 'A20');
```

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.

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

```
CTRFLD(string, length, 'outfield')
```

where:

string
Alphanumeric

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

length
Integer

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

outfield
Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: **Centering a Field**

CTRFLD centers LAST_NAME and stores the result in CENTER_NAME:

```
COMPUTE CENTER_NAME/A12 = CTRFLD(LAST_NAME, 12, 'A12');
```

EDIT: Extracting or Adding Characters

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

EDIT works by comparing the characters in a mask to the characters in a source field. When it encounters a nine in the mask, EDIT copies the corresponding character from the source field to the new field. When it encounters a dollar sign in the mask, EDIT ignores the corresponding character in the source field. When it encounters any other character in the mask, EDIT copies that character to the corresponding position in the new field. EDIT does not require an outfield argument because the result is obviously alphanumeric and its size is determined from the mask value.

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

Syntax: How to Extract or Add Characters

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

where:

fieldname

Alphanumeric

Is the source field.

Is the string to extract characters from. It should be at least as long as the mask.

mask

Alphanumeric

Is a character string enclosed in single quotation marks. The length of the mask, excluding any characters other than nine and $, determines the length of the output field.

Example: Extracting and Adding a Character to a Field

EDIT extracts the first initial from the FIRST_NAME field and stores the result in FIRST_INIT. EDIT also adds dashes to the EMP_ID field and stores the result in EMPIDEDIT:

```
COMPUTE FIRST_INIT/A1 = EDIT(FIRST_NAME, '9$$$$$$$$$'); AND
COMPUTE EMPIDEDIT/A11 = EDIT(EMP_ID, '999-99-9999');
```

GETTOK: Extracting a Substring (Token)

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

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

Syntax: How to Extract a Substring (Token)

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

where:

infield

Alphanumeric

Is the field containing the parent character string.

inlen

Integer

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

token_number

Integer

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

delim

Alphanumeric

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

outlen

Integer

Is the maximum size of the token. If this argument is less than or equal to 0, the function returns spaces. If the token is longer than this argument, it is truncated; if it is shorter, it is padded with trailing spaces.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks. The delimiter is not included in the token.

Example: Extracting a Token From a Field

GETTOK extracts the last token from ADDRESS_LN3 and stores the result in LAST_TOKEN:

```
COMPUTE LAST_TOKEN/A10 = GETTOK(ADDRESS_LN3, 20, -1, ' ', 10, 'A10');
```

LCWORD: Converting a Character String to Mixed Case

The LCWORD function converts the letters in a character string to mixed case. It converts every alphanumeric character to lowercase except the first letter of each new word and the first letter after a single or double quotation mark. For example, O'CONNOR is converted to O'Connor and JACK'S to Jack'S.

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

Syntax: How to Convert a Character String to Mixed Case

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

where:

length
Integer

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

string
Alphanumeric

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

outfield
Alphanumeric

Is the format of the output value enclosed in single quotation marks. The length must be greater than or equal to the length of *length*.

Example: Converting a Character String to Mixed-Case

LCWORD converts the LAST_NAME field to mixed-case and stores the result in MIXED_CASE:

```
COMPUTE MIXED_CASE/A15 = LCWORD(15, LAST_NAME, 'A15');
```

LCWORD2: Converting a Character String to Mixed-Case

The LCWORD2 function converts the letters in a character string to mixed-case by converting every alphanumeric character to lowercase except the first letter of each new word. If LCWORD2 encounters a lone single quotation mark, the next letter is converted to lowercase. For example, 'SMITH' would be changed to 'Smith' and JACK'S would be changed to Jack's.

Syntax: How to Convert a Character String to Mixed-Case

```
LCWORD2(length, string, 'outfield')
```

where:

length

Integer

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

string

Alphanumeric

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

outfield

Alphanumeric

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

Example: Converting a Character String to Mixed-Case

Use the COMPUTE or DEFINED dialog in the report development tools or the Synonym Editor in Developer Workbench to perform a conversion where LCWORD2 converts the string O'CONNOR’s to mixed-case:

```
MYVAL1/A10='O'CONNOR'S';
LC2/A10 = LCWORD2(10, MYVAL1, 'A10');
```

The report output for these fields:

```
MYVAL1      LC2
------      ---
O'CONNOR'S  O'Connor's
```

LCWORD3: Converting a Character 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.

Syntax: How to 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 the length of *length*.

Example: Converting a Character String to Mixed-Case Using LCWORD3

Use the Compute or Define dialog box in the report development tools or the Synonym Editor in Developer Workbench to perform a conversion where LCWORD3 converts the strings O'CONNOR’s and o’connor’s to mixed-case:

```
MYVAL1/A10='O'CONNOR'S';
MYVAL2/A10='o'connor's';
LC1/A10 = LCWORD3(10, MYVAL1, 'A10');
LC2/A10 = LCWORD3(10, MYVAL2, 'A10');
```

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 Character String

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

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

Syntax: How to Left-Justify a Character String

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

where:

length
Integer

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

string
Alphanumeric

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

outfield
Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Left-Justifying a Field

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

```
COMPUTE YNAME/A25 = LJUST(15, XNAME, 'A25');
```

LOCASE: Converting Text to Lowercase

The LOCASE function converts alphanumeric text to lowercase.

Syntax: How to Convert Text to Lowercase

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

where:

length
Integer

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

string
Alphanumeric

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks. The field name can be the same as *string*.

Example: **Converting a Field to Lowercase**

LOCASE converts the LAST_NAME field to lowercase and stores the result in LOWER_NAME:

```
COMPUTE LOWER_NAME/A15 = LOCASE(15, LAST_NAME, 'A15');
```

OVRLAY: Overlaying a Character String

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

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

```
OVRLAY(string1, stringlen, string2, sublen, position, 'outfield')
```

where:

string1

Alphanumeric

Is the base character string.

stringlen

Integer

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

string2

Alphanumeric

Is the substring that will overlay string1.

sublen

Integer

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

position

Integer

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

outfield

Alphanumeric

Is the 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.

Example: Replacing Characters in a Character String

OVRLAY replaces the last three characters of EMP_ID with CURR_JOBCODE to create a new security identification code and stores the result in NEW_ID:

```
COMPUTE NEW_ID/A9 = OVRLAY(EMP_ID, 9, CURR_JOBCODE, 3, 7, 'A9');
```

PARAG: Dividing Text Into Smaller Lines

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

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

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

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

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

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

where:

length

Integer

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

string

Alphanumeric

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

delim

Alphanumeric

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

subsize

Integer

Is the maximum length of each sub-line.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: **Dividing Text Into Smaller Lines**

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

```
COMPUTE PARA_ADDR/A20 = PARAG(20, ADDRESS_LN2, ',', 10, 'A20');
```

POSIT: Finding the Beginning of a Substring

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

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

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

where:

parent

Alphanumeric

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

inlength

Integer

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

substring

Alphanumeric

Is the substring whose position you want to find. This can be the substring enclosed in single quotation marks, or the field that contains the string.

sublength

Integer

Is the length of *substring*. If this argument is less than or equal to 0, or if it is greater than *inlength*, the function returns a 0.

outfield

Integer

Is the format of the output value enclosed in single quotation marks.

Example: **Finding the Position of a Letter**

POSIT determines the position of the first capital letter I in LAST_NAME and stores the result in I_IN_NAME:

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

PTOA: Packed Decimal to Alphanumeric Conversion

The PTOA function converts a 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.

Reference: **Packed Decimal to Alphanumeric Conversion**

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

where:

`number`
: Numeric P (packed-decimal) or F or D (single and double precision floating-point)
Is the number to be converted.

`format`
: Alphanumeric
Is the format of the number enclosed in parenthesis.

`output`
: Alphanumeric

The length of this argument must be greater than the length of number and must account for edit options and a possible negative sign. For example, converting from packed to alphanumeric format converts PGROSS from packed-decimal to alphanumeric format.

```
PTOA(PGROSS, '(P12.2)', 'A14');
```

REVERSE: Reversing Characters in a Character String

The REVERSE function reverses the characters in a character string.

Syntax: **How to Reverse Characters in a Character String**

```
REVERSE(length, string, 'outfield')
```

where:

`length`
: Integer

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

`string`
: Alphanumeric

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Reversing the Characters in a String

REVERSE reverses the characters in PRODCAT and stores the result in REVERSE_NAME:

```
COMPUTE REVERSE_NAME/A15 = REVERSE(15, PRODCAT, 'A15');
```

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.

Syntax: How to Right-Justify a Character String

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

where:

length

Integer

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

string

Alphanumeric

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Right-Justifying a Field

RJUST right-justifies the LAST_NAME field and stores the result in RIGHT_NAME:

```
COMPUTE RIGHT_NAME/A15 = RJUST(15, LAST_NAME, 'A15');
```

SOUNDEX: Comparing Character Strings Phonetically

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

To conduct a phonetic search, do the following:

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

Syntax: How to Compare Character Strings Phonetically

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

where:

inlength

2-byte Alphanumeric

Is the length, in characters, of *string*, or a field that contains the length. It can be a number enclosed in single quotation marks, or a field containing the number. The number must be from 01 to 99, expressed with two digits (for example '01'); a number larger than 99 causes the function to return asterisks (*) as output.

string

Alphanumeric

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: **Comparing Character Strings Phonetically**

The following request creates three fields:

- ❑ PHON_NAME contains the phonetic code of employee last names.
- ❑ PHON_COY contains the phonetic code of your guess, MICOY.
- ❑ PHON_MATCH compares the guess with the phonetic code.

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

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.

Syntax: **How to Spell Out a Dollar Amount**

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

where:

outlength

Integer

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

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

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

If number is less than...	...outlength should be
$1,000,000	96

number

Alphanumeric or Decimal (9.2)

Is the number to be spelled out.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: **Spelling Out a Dollar Amount**

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

```
COMPUTE AMT_IN_WORDS/A82 = SPELLNM(82, CURR_SAL, 'A82');
```

SUBSTR: Extracting a Substring

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

Syntax: **How to Extract a Substring**

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

where:

inlength

Integer

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

parent

Alphanumeric

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

start

Integer

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

end

Integer

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

sublength

Integer

Is the length of the substring (normally end - start + 1). If *sublength* is longer than *end* - *start* +1, the substring is padded with trailing spaces. If it is shorter, the substring is truncated. This value should be the declared length of *outfield*. Only *sublength* characters will be processed.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Extracting a String

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

```
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,
'A3');
```

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.

Syntax: How to Convert Text to Uppercase

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

where:

length

Integer

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

input

Alphanumeric

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Converting a Mixed-Case Field to Uppercase

UPCASE converts the LAST_NAME_MIXED field to uppercase and stores the result in LAST_NAME_UPPER:

```
COMPUTE LAST_NAME_UPPER/A15 = UPCASE(15, LAST_NAME_MIXED, 'A15') ;
```

Chapter 3

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).

In this chapter:

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*.

Syntax: How to 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.

Example: 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
```

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.

Syntax: How to Convert a Number to a Character String

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

where:

number

Integer

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

length

Integer between 1 and 10

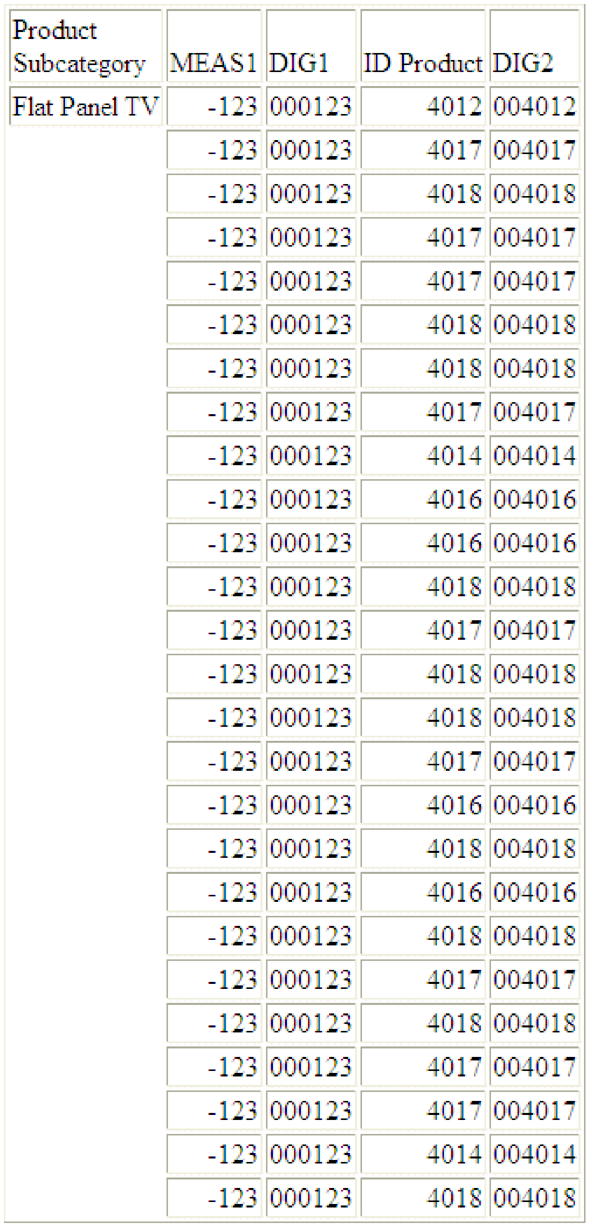
Is the length of the returned character string. If *length* is longer than the number of digits in the number being converted, the returned value is padded on the left with zeros. If *length* is shorter than the number of digits in the number being converted, the returned value is truncated on the left.

Example: Converting a Number to a Character String

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

```
DEFINE FILE WF_RETAIL_LITE
MEAS1/I8=-123.45;
DIG1/A6=DIGITS(MEAS1,6) ;
DIG2/A6=DIGITS(ID_PRODUCT,6) ;
END
TABLE FILE WF_RETAIL_LITE
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

Reference: 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.

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.

Syntax: How to 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.

Example: 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.

Syntax: How to 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

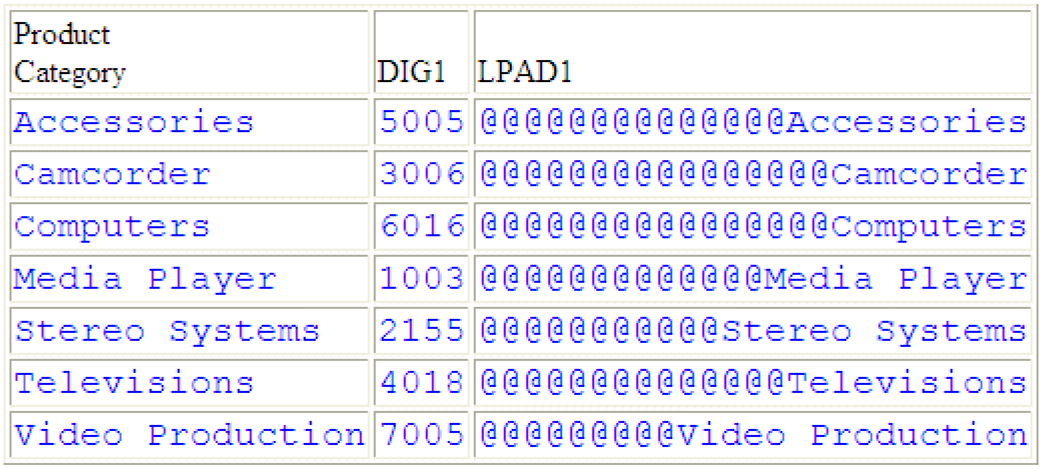
Is a single character to use for padding.

Example: 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 WF_RETAIL
LPAD1/A25 = LPAD(PRODUCT_CATEGORY,25,'@');
DIG1/A4 = DIGITS(ID_PRODUCT,4);
END
TABLE FILE WF_RETAIL
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

Reference: 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.

Syntax: How to 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.

Example: 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 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.
```

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.

Syntax: How to 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.

Example: 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
```

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.

Syntax: How to 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

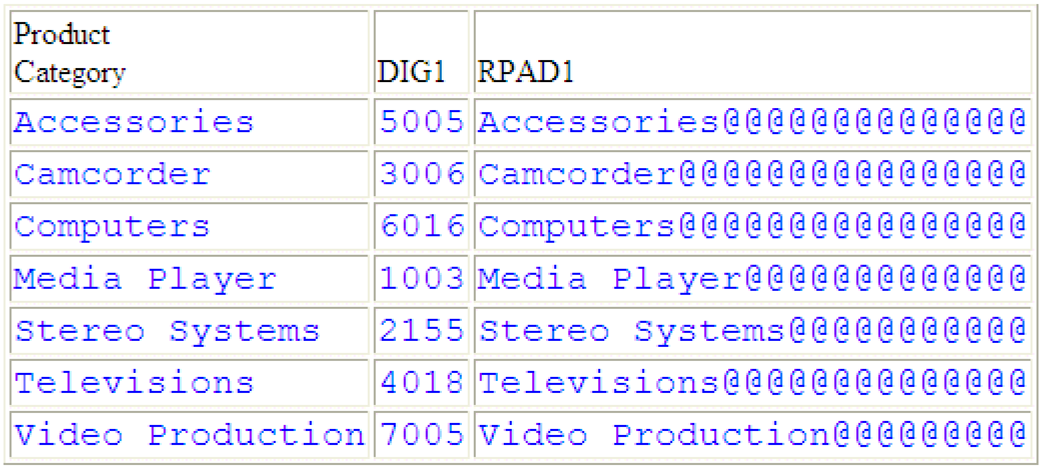
Is a single character to use for padding.

Example: 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 WF_RETAIL
RPAD1/A25 = RPAD(PRODUCT_CATEGORY,25,'@');
DIG1/A4 = DIGITS(ID_PRODUCT,4);
END
TABLE FILE WF_RETAIL
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@@@@@@@@@

Reference: 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.

Syntax: How to 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.

Example: 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./
```

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.

Syntax: How to 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.

Example: 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 character and specified by a token number reflecting the position of the token in the string.

Syntax: How to 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 single character delimiter.

number
: Integer

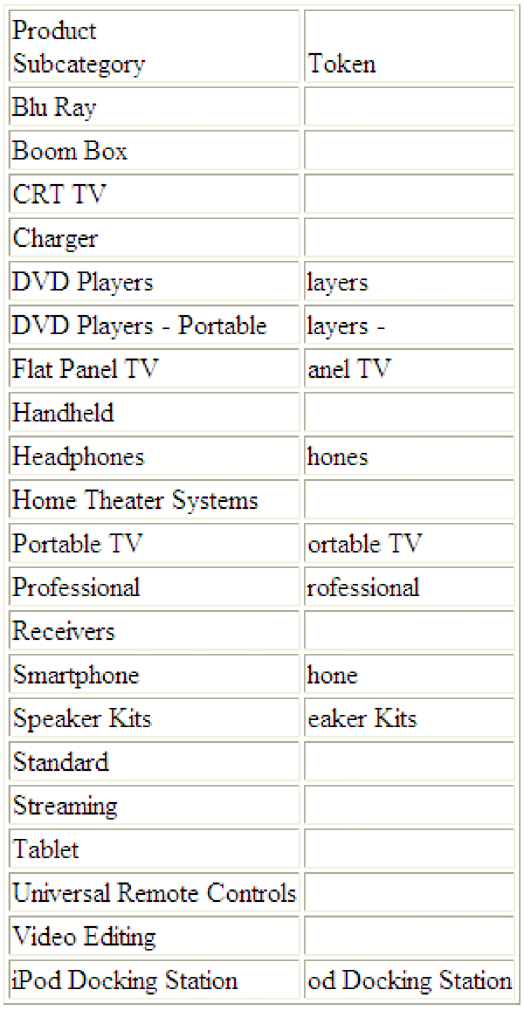
  Is the token number to extract.

Example: 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 WF_RETAIL_LITE
TOK1/A20 =TOKEN(PRODUCT_SUBCATEG,'P',2);
END
TABLE FILE WF_RETAIL_LITE
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.

Syntax: How to 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.

Example: 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.
```

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.

Syntax: How to 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.

Example: 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
```

Chapter 4

Data Source and Decoding Functions

Data source and decoding functions retrieve data source values and assign values based on the value of an input field.

In this chapter:

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 Web Query or SQL language request.

You can use the DB_EXPR function in a DEFINE command, a DEFINE in a Master File, a WHERE clause, 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.

Syntax: How to 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.

Reference: Usage Notes for the DB_EXPR Function

❑ Any request that includes one or more DB_EXPR functions must be for a synonym that has a relational SUFFIX.

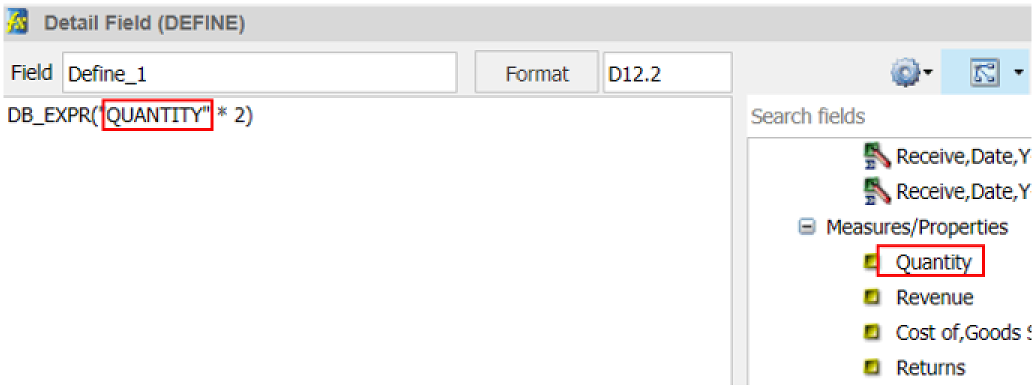
❑ Field references in the native SQL expression must be within the current synonym context.

❑ The native SQL expression must be coded inline. SQL read from a file is not supported.

- DB_EXPR requires using WITH or placing FIELDNAMEs within double quotation marks when you reference FIELDNAMEs within the function itself. It is recommended to use the double quotation marks to ensure the FIELDNAME reference in the SQL snippet expression is qualified appropriately when the translated SQL is created for the request.
- FIELDNAMEs specified in the SQL expression are case sensitive.

Example: Multiplying QUANTITY by Two

The below expression multiplies a field named QUANTITY by 2. The FIELDNAME is case sensitive and enclosed in double quotation marks ("). Note that the FIELDNAME is used in the expression, not the COLUMN HEADING that is displayed in the right panel of the DEFINE dialog box, as shown in the following image.



Field: Define_1
Format: D12.2

```
DB_EXPR("QUANTITY" * 2)
```

Example: Inserting the Db2 BIGINT and CHAR Functions into a Request

The following syntax uses the DB_EXPR function to call two Db2 functions. It calls the BIGINT function to convert the squared revenue field to a BIGINT data type, and then uses the CHAR function to convert that value to alphanumeric.

```
DB_EXPR(CHAR(BIGINT("REVENUE" * "REVENUE") ) )
```

The expression can be entered into a DEFINE or COMPUTE field and can be entered using InfoAssist+ or the Developer Workbench Synonym Editor.

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.

Syntax: How to Supply Values in the Function

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

where:

fieldname

Alphanumeric or Numeric

Is the name of the input field.

code

Alphanumeric or Numeric

Is the coded value for which DECODE searches. If the value has embedded blanks, commas, or other special characters, enclose it in single quotation marks. When DECODE finds the specified value, it assigns the corresponding result.

result

Alphanumeric or Numeric

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

default

Alphanumeric or Numeric

Is the value assigned if the code is not found. If you omit a default value, DECODE assigns a blank or zero to non-matching codes.

You can use up to 40 lines to define the code and result pairs for any given DECODE function, or 39 lines if you also use an ELSE phrase. Use either a comma or blank to separate the code from the result, or one pair from another.

Example: Supplying Values in the Function

EDIT extracts the first character of the CURR_JOBCODE field, then DECODE returns either ADMINISTRATIVE or DATA PROCESSING depending on the value extracted.

```
COMPUTE DEPX_CODE/A1 = EDIT(CURR_JOBCODE, '9$$'); AND
COMPUTE JOB_CATEGORY/A15 = DECODE DEPX_CODE(A 'ADMINISTRATIVE'
B 'DATA PROCESSING') ;
```

Syntax: How to Read Values From a File

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

where:

fieldname

Alphanumeric or Numeric

Is the name of the input field.

ddname

Alphanumeric

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

default

Alphanumeric or Numeric

Is the value assigned if the code is not found. If you omit a default, DECODE assigns a blank or zero to non-matching codes.

Reference: Guidelines for Reading Values From a File

- Each record in the file is expected to contain pairs of elements separated by a comma or blank.
- If each record in the file consists of only one element, this element is interpreted as the code, and the result becomes either a blank or zero, as needed.

  This makes it possible to use the file to hold screening literals referenced in the screening condition

  ```
  IF field IS (filename)
  ```

  and as a file of literals for an IF criteria specified in a computational expression. For example:

  ```
  TAKE = DECODE SELECT (filename ELSE 1);
  VALUE = IF TAKE IS 0 THEN... ELSE...;
  ```

TAKE is 0 for SELECT values found in the literal file and 1 in all other cases. The VALUE computation is carried out as if the expression had been:

```
IF SELECT (filename) THEN... ELSE...;
```

- The file can contain up to 32,767 characters in the file.
- Leading and trailing blanks are ignored.
- The remainder of each record is ignored and can be used for comments or other data. This convention applies in all cases, except when the file name is HOLD. In that case, the file is presumed to have been created by the HOLD command, which writes fields in the internal format, and the DECODE pairs are interpreted accordingly. In this case, extraneous data in the record is ignored.

Example: Reading Values From a File

DECODE assigns the value 0 to an employee whose EMP_ID appears in the HOLD file and 1 when EMP_ID does not appear in the file.

```
COMPUTE NOT_IN_LIST/I1 = DECODE EMP_ID(HOLD ELSE 1);
```

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.

Syntax: How to Retrieve the Preceding Value

```
LAST fieldname
```

where:

fieldname

Alphanumeric or Numeric

Is the field name.

Example: Retrieving the Preceding Value

LAST retrieves the previous value of the DEPARTMENT field to determine whether to restart the running total of salaries by department. If the previous value equals the current value, CURR_SAL is added to RUN_TOT to generate a running total of salaries within each department.

```
COMPUTE RUN_TOT/D12.2M = IF DEPARTMENT EQ LAST DEPARTMENT THEN
                (RUN_TOT + CURR_SAL) ELSE CURR_SAL ;
```

Chapter 5

Date and Time Functions

Date and time functions manipulate date and time values.

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

You can affect the behavior of date and time functions by 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.

AYM: Adding or Subtracting Months to or From Dates

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

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

```
AYM(indate, months, 'outfield')
```

where:

indate

Integer (I4, I4YM, I6, or I6YYM)

Is the original date in year-month format, the name of a field that contains the date, or an expression that returns the date. If the date is not valid, the function returns a 0.

months

Integer

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

outfield

Integer (I4YM or I6YYM)

Is 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.

Example: Adding Months to a Date

The COMPUTE command converts the dates in HIRE_DATE from year-month-day to year-month format and stores the result in HIRE_MONTH. AYM then adds six months to HIRE_MONTH and stores the result in AFTER6MONTHS.

```
COMPUTE HIRE_MONTH/I4YM = HIRE_DATE/100; AND
COMPUTE AFTER6MONTHS/I4YM = AYM(HIRE_MONTH, 6, 'I4YM');
```

AYMD: Adding or Subtracting Days to or From a Date

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

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

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

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

where:

indate

Integer (I6, I6YMD, I8, I8YYMD)

If the date is not valid, the function returns a 0.

days

Integer

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

outfield

Integer (I6, I6YMD, I8, or I8YYMD)

Is the format of the output value enclosed in single quotation marks. If *indate* is a field, *outfield* must have the same format.

Example: Adding Days to a Date

AYMD adds 35 days to each value in the HIRE_DATE field, and stores the result in AFTER35DAYS:

```
COMPUTE AFTER35DAYS/I6YMD = AYMD(HIRE_DATE, 35, 'I6YMD');
```

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, 97 or 1997). CHGDAT reads an input date character string and creates an output date character string that represents the same date in a different way.

Note: CHGDAT requires a date character string as input, not a date itself. Convert the input 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.

Character	Description
X	Displays the full name of the month.

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

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

Note: Display options are *not* date formats.

Reference: Short to Long Conversion

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

Portion of Date Missing	Portion Supplied by Function
Day (for example, from YM to YMD)	Last day of the month.
Month (for example, from Y to YM)	Last month of the year (December).
Year (for example, from MD to YMD)	The year 99.
Converting year from two-digit to four-digit (for example, from YMD to YYMD)	If DATEFNS=ON, the century will be determined by the 100-year window defined by DEFCENT and YRTHRESH. If DATEFNS=OFF, the year 19*xx* is supplied, where *xx* is the last two digits in the year.

Syntax: How to Change the Date Display String

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

where:

in_display_options

Alphanumeric (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

Alphanumeric (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

Alphanumeric (A2 to A8)

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

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

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

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.

Reference: Usage Notes for CHGDAT

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

Example: Converting the Date Display From YMD to MDYYX

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

```
COMPUTE ALPHA_HIRE/A17 = EDIT(HIRE_DATE); AND
COMPUTE HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
```

DA Functions: Converting a Date to an Integer

The DA functions convert a date to the number of days between December 31, 1899 and that date. By converting a date to the number of days, you can add and subtract dates and calculate the intervals between them.

There are six DA functions; each one accepts a date in a different format.

Syntax: How to Convert a Date to an Integer

```
function(indate, 'outfield')
```

where:

function

Is one of the following:

`DADMY` converts a date in day-month-year format.

`DADYM` converts a date in day-year-month format.

`DAMDY` converts a date in month-day-year format.

`DAMYD` converts a date in month-year-day format.

`DAYDM` converts a date in year-day-month format.

`DAYMD` converts a date in year-month-day format.

indate

Integer or Packed

I or P format with date display options.

Is the date to be converted, or the name of a field that contains the date. The date is truncated to an integer before conversion.

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

outfield

Integer

Is the format of the output value enclosed in single quotation marks. The format of the date returned depends on the function.

Example: Converting Dates and Calculating the Difference Between Them

DAYMD converts the DAT_INC and HIRE_DATE fields to the number of days since December 31, 1899, and the smaller number is then subtracted from the larger number:

```
COMPUTE DAYS_HIRED/I8 = DAYMD(DAT_INC, 'I8') - DAYMD(HIRE_DATE, 'I8');
```

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 date format. A unit is one of the following:

- **Year.**
- **Month.** If the calculation using the month unit creates an invalid date, DATEADD corrects it to the last day of the month. For example, adding one month to October 31 yields November 30, not November 31 since November has 30 days.
- **Day.**
- **Weekday.** When using the weekday unit, DATEADD does not count Saturday or Sunday. For example, if you add one day to Friday, the result is Monday.
- **Business day.** When using the business day unit, DATEADD uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. If Monday is not a working day, then one business day past Sunday is Tuesday.

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

DATEADD works only with full component dates.

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

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

where:

date
: Date

  Is a full component date.

unit
: Alphanumeric

Is one of the following enclosed in single quotation marks:

`Y` indicates a year unit.

`M` indicates a month unit.

`D` indicates a day unit.

`WD` indicates a weekday unit.

`BD` indicates a business day unit.

`#units`

Integer

Is the number of date units added to or subtracted from *date*. If this number is not a whole unit, it is rounded down to the next largest integer.

Example: **Adding Weekdays to a Date**

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.

```
COMPUTE NEW_DATE/YYMD = HIRE_DATE; AND
COMPUTE HIRE_DATE_PLUS_THREE/YYMD = DATEADD(NEW_DATE, 'WD', 3);
```

DATECVT: Converting the Format of a Date

The DATECVT function converts the format of a date in an application without requiring an intermediate calculation. If you supply an invalid format, DATECVT returns a zero or a blank.

The DATECVT function is optimized when using the following date formats: A8YYMD, A8MDYY, A8DMYY, I8YYMD, I8MDYY, I8DMYY, P8YYMD, P8MDYY, and P8DMYY. Optimized SQL is now passed to the DB2 Engine for processing when converting dates.

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

Syntax: **How to Convert a Date Format**

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

where:

`date`

Date

Is the date to be converted. If you supply an invalid date, DATECVT returns zero. When the conversion is performed, a legacy date obeys any DEFCENT and YRTHRESH parameter settings supplied for that field.

infmt

Alphanumeric

Is the format of the date enclosed in single quotation marks. It is one of the following:

- A non-legacy date format (for example, YYMD, YQ, M, DMY, JUL).
- A legacy date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in *infmt* functions as an offset from the base date of a YYMD field (12/31/1900).

outfmt

Alphanumeric

Is the output format enclosed in single quotation marks. It is one of the following:

- A non-legacy date format (for example, YYMD, YQ, M, DMY, JUL).
- A legacy date format (for example, I6YMD or A8MDYY).
- A non-date format (such as I8 or A6). A non-date format in *infmt* functions as an offset from the base date of a YYMD field (12/31/1900).

Example: Converting a YYMD Date to DMY

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

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

or

```
COMPUTE CONV_FIELD/DMY = DATECVT('19991231', 'A8YYMD', 'DMY');
```

Example: Converting a Legacy Date to Date Format

DATECVT converts HIRE_DATE from I6YMD legacy date format to YYMD date format:

```
COMPUTE NEW_HIRE_DATE/YYMD = DATECVT(HIRE_DATE, 'I6YMD', 'YYMD');
```

DATEDIF: Finding the Difference Between Two Dates

The DATEDIF function returns the difference between two dates in units. A unit is one of the following:

- **Year.** Using the year unit with DATEDIF yields the inverse of DATEADD. If subtracting one year from date X creates date Y, then the count of years between X and Y is one. Subtracting one year from February 29 produces the date February 28.

- **Month.** Using the month unit with DATEDIF yields the inverse of DATEADD. If subtracting one month from date X creates date Y, then the count of months between X and Y is one. If the to-date is the end-of-month, then the month difference may be rounded up (in absolute terms) to guarantee the inverse rule.

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

- **Day.**

- **Weekday.** With the weekday unit, DATEDIF does not count Saturday or Sunday when calculating days. This means that the difference between Friday and Monday is one day.

- **Business day.** With the business day unit, DATEDIF uses the BUSDAYS parameter setting and holiday file to determine which days are working days and disregards the rest. This means that if Monday is not a working day, the difference between Friday and Tuesday is one day.

DATEDIF returns a whole number. If the difference between two dates is not a whole number, DATEDIF truncates the value to the next largest integer. For example, the number of years between March 2, 2001, and March 1, 2002, is zero. If the end date is before the start date, DATEDIF returns a negative number.

You can find the difference between non-day based dates (for example YM or YQ) directly without using DATEDIF.

Syntax: How to Find the Difference Between Two Dates

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

where:

from_date

Date

Is the start date from which to calculate the difference. Is a full component date.

to_date

Date

Is the end date from which to calculate the difference.

unit

Alphanumeric

Is one of the following enclosed in single quotation marks:

`Y` indicates a year unit.

`M` indicates a month unit.

`D` indicates a day unit.

`WD` indicates a weekday unit.

`BD` indicates a business day unit.

Example: Finding the Number of Weekdays Between Two Dates

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:

```
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');
```

DATEMOV: Moving a Date to a Significant Point

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

DATEMOV works only with full component dates.

Syntax: How to Move a Date to a Significant Point

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

where:

`date`

Date

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

`move-point`

Alphanumeric

Is the significant point the date is moved to enclosed in single quotation marks. An invalid point results in a return code of zero. Valid values are:

`EOM` is the end of month.

`BOM` is the beginning of month.

`EOQ` is the end of quarter.

`BOQ` is the beginning of quarter.

`EOY` is the end of year.

`BOY` is the beginning of year.

`EOW` is the end of week.

`BOW` is the beginning of week.

`NWD` is the next weekday.

`NBD` is the next business day.

`PWD` is the prior weekday.

`PBD` is the prior business day.

`WD-` is a weekday or earlier.

`BD-` is a business day or earlier.

`WD+` is a weekday or later.

`BD+` is a business day or later.

A business day calculation is affected by the BUSDAYS and HDAY parameter settings.

Example: **Determining the End of the Week**

DATEMOV determines the end of the week for each date in NEW_DATE and stores the result in EOW:

```
COMPUTE NEW_DATE/YYMDWT = DATECVT(HIRE_DATE, 'I6YMD', 'YYMDWT'); AND
COMPUTE EOW/YYMDWT = DATEMOV(NEW_DATE, 'EOW');
```

Returning a Date Component as an Integer

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

Syntax: **How to Extract a Date Component and Return It in Integer Format**

```
DPART(datevalue, 'component', outfield)
```

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 the following:

For year: YEAR, YY

For month: MONTH, MM

For day: DAY, for day of month: DAY-OF-MONTH.

For quarter: QUARTER, QQ

outfield

Integer

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

Example: 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', 'I4');
 MONTH/I4 = DPART(TRANSDATE, 'MM', 'I4');
 DAY/I4 = DPART(TRANSDATE, 'DAY', 'I4');
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
```

DATETRAN: Formatting Dates in International Formats

The DATETRAN function formats dates in international formats.

Syntax: How to Format Dates in International Formats

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

where:

indate

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

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 single quotation marks and parentheses.

These are 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).

These are 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.

These are 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.
`'(MDY)'`	Month component followed by day followed by two-digit year.
`'(MD)'`	Month component followed by day (derived from three-component date by ignoring year component).
`'(DM)'`	Day component followed by month (derived from three-component date by ignoring year component).

formatops

Is a string of zeros 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 are as follows:

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

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

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

Valid month and day name translation options are:

Format Option	Description
T	Displays month as an abbreviated name with no punctuation, all uppercase.
TR	Displays month as a full name, all uppercase.
Tp	Displays month as an abbreviated name followed by a period, all uppercase.

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

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

Valid date delimiter options are:

Format Option	Description
B	Uses a blank as the component delimiter. This is the default if the month or day of week is translated or if comma is used.
.	Uses a period as the component delimiter.
-	Uses a minus sign as the component delimiter. This is the default when the conditions for a blank default delimiter are not satisfied.
/	Uses a slash as the component delimiter.
|	Omits component delimiters.
K	Uses appropriate Asian characters as component delimiters.

Format Option	Description
c	Places a comma after the month name (following T, Tp, TR, t, tp, or tr). Places a comma and blank after the day name (following W, Wp, WR, w, wp, or wr). Places a comma and blank before the day name (following X, XR, x, or xr).
e	Displays the Spanish or Portuguese word de or DE between the day and month and between the month and year. The case of the word de is determined by the case of the month name. If the month is displayed in uppercase, DE is displayed; otherwise de is displayed. Useful for formats DMY, DMYY, MY, and MYY.
D	Inserts a comma after the day number and before the general delimiter character specified.
Y	Inserts a comma after the year and before the general delimiter character specified.

lang

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

'AR' Arabic

'CS' Czech

'DA' Danish

'DE' German

'DU' Dutch

'EN' English

'ES' Spanish

'FI' Finnish

'FR' French

'EL' Greek

'IW' Hebrew

'IT' Italian

'JA' Japanese

'KO' Korean

'LT' Lithuanian

'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.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Reference: Usage Notes for the DATETRAN Function

❑ The type A output field may contain variable length information, since the lengths of month names and day names can vary. Also month and day numbers may be either one or two bytes long if a zero-suppression option is chosen. Unused bytes are filled with blanks.

❑ All invalid and inconsistent inputs result in all blank output strings. Missing data also results in blank output.

❑ The base dates (1900-12-31 and 1900-12 or 1901-01) are treated as though the DATEDISPLAY setting were ON (that is, not automatically shown as blanks). To suppress the printing of base dates, which have an internal integer value of 0, test for 0 before calling DATETRAN. For example:

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

- Valid translated date components are contained in files named DTLNG*lng* where *lng* is a three-character code that specifies the language. These files must be accessible for each language into which you want to translate dates.

Example: **Using the DATETRAN Function**

DATETRAN prints the day of the week in the default case for French:

```
COMPUTE OUT/A8=DATETRAN(DATEW, '(W)', '(wr)', 'FR', 8 , 'A8') ;
```

Precision for Date-Time Values

In prior releases, the seconds component of a date-time value could be displayed with zero, three, or six decimal digits. It can now be displayed with zero through nine decimal digits.

In order to control the precision of the seconds component in a date-time format, you can specify a digit from 1 to 9 in the format.

Syntax: **How to Specify Precision for Date-Time Values**

Special format components exist for three decimal digits (milliseconds, s), six decimal digits (microseconds, m), and nine decimal digits (nanoseconds, n). To display:

- **A Time-only value**, when the format includes more than one time display component:
  - The components must appear in the order hour, minute, second, millisecond, microsecond, or nanosecond.
  - The first component must be either hour, minute, or second.
  - No intermediate component can be skipped. If hour is specified, the next component must be minute; it cannot be second.
  - To display a specific number of digits, include the format for seconds (S) followed by the number of digits to display. Alternatively, you can specify one or more of the following components: milliseconds (*s*), microseconds (*m*), nanoseconds (*n*).
- **Both a date component and a time component**, the time component is represented by one character, which specifies the smallest unit of time to be displayed. This character can be either:
  - A number from 1 to 9, which specifies the number of decimal digits to display for the seconds component.

- One of the supported time components. In this case, all higher-order time components will also be included in the display.

Example: Specifying Precision for Date-Time Values

Use the COMPUTE or DEFINE dialog in the report development tools or the Synonym Editor in Developer Workbench to perform these transformations. Assume the date is February 5, 1999 and the time is 02:05:25.123456789 a.m. The following defines use of the nanosecond component or a precision in display formats and function calls:

```
TRANSDT/HYYMDn       = DT(19990205 02:05:25.123456789);
 O_HSsmn/HSsmn        = TRANSDT;
 O_HHIS2/HHIS2        = TRANSDT;
 O_HYYMDn/HYYMDn      = TRANSDT;
 O_HYYMD1/HYYMD1      = TRANSDT;
 O_HADD/HYYMD9        = HADD(TRANSDT, 'NS', 2, 12, 'HYYMD9');
 O_HCNVRT/A26         = HCNVRT(TRANSDT, '(H23)', 23, 'A26');
 O_HDIFF/D12.2        = HDIFF(O_HADD, TRANSDT, 'NS', 'D12.2');
 TRANSDATE_DATE/YYMD = HDATE(TRANSDT, 'YYMD');
 O_HDTTM/HYYMDn       = HDTTM(TRANSDATE_DATE,12, 'HYYMDn');
 O_HEXTR/HHIS9        = HEXTR(TRANSDT, 'n',12, 'HHIS9');
 O_HGETC/HYYMDn       = HGETC(12, 'HYYMDn');
 O_HINPUT/HYYMDn      = HINPUT(14, O_HCNVRT,12, 'HYYMDn');
 O_HMASK/HYYMDn       = HMASK(O_HEXTR, 'HISsmn', TRANSDT, 12, 'HYYMDn');
 O_HMIDNT/HYYMDn      = HMIDNT(TRANSDT,12, 'HYYMDn');
 O_HNAME/A10          = HNAME(TRANSDT, 'NANOSECOND', 'A10');
 O_HPART/I10          = HPART(TRANSDT, 'NANOSECOND', 'I10');
 O_HSETPT/HYYMDn      = HSETPT(TRANSDT, 'NS', 28, 12,'HYYMDn');
 O_HTIME/P20.2C       = HTIME(12,TRANSDT, 'D16');
```

Output of these Define fields in a report:

```
TRANSDT   1999/02/05 02:05:25.123456789
O_HSsmn   25.123456789
O_HHIS2   02:05:25.12
O_HYYMDn  1999/02/05 02:05:25.123456789
O_HYYMD1  1999/02/05 02:05:25.1
O_HADD    1999/02/05 02:05:25.123456791
O_HCNVRT  19990205020525123456789
O_HDIFF                               2.00
O_HDTTM   1999/02/05 00:00:00.000000000
O_HEXTR   00:00:00.000000789
O_HGETC   2008/06/25 10:26:52.343644000
O_HINPUT  1999/02/05 02:05:25.000000000
O_HMASK   1999/02/05 00:00:00.000456789
O_HMIDNT  1999/02/05 00:00:00.000000000
O_HNAME   123456789
O_HPART                          123456789
O_HSETPT  1999/02/05 02:05:25.000000028
O_HTIME            7,525,123,456,789.00
```

Note:

- ❏ Field O_HSsmn displays the seconds (S), milliseconds (s), microseconds (m), and nanoseconds (n) from TRANSDT.
- ❏ Field O_HHIS2 displays the hours (H), minutes (I), and seconds (S) from TRANSDT. The seconds are displayed with two decimal digits (2).
- ❏ Field O_HYYMDn displays TRANSDT with the date in YYMD format and the time down to nanoseconds (n).
- ❏ Field O_HYYMD1 displays TRANSDT with the date in YYMD format and the time down to seconds with one decimal digit (1).
- ❏ Field O_HADD is created by calling the HADD function to add 2 nanoseconds to the date-time value in TRANSDT.
- ❏ Field O_HCNVRT is created by calling the HCNVRT function to convert the date-time value in TRANSDT to alphanumeric format.
- ❏ Field O_HDIFF is created by calling the HDIFF function to subtract the date-time value in TRANSDT from the date-time version in O_HADD.
- ❏ Field O_HDTTM is created by calling the HDTTM function to create a date-time field by taking the date from TRANSDATE_DATE and setting the time components to zero.
- ❏ Field O_HEXTR is created by calling the HEXTR function to extract the nanoseconds (low order three digits of nine) from TRANSDT and set the remaining components to zero.
- ❏ Field HGETC is created by calling the HGETC function to retrieve the current date and time and display them with 9 decimal digits for the seconds component. Note that not all operating systems return all 9 decimal digits, in which case the digits not returned display as zeros.
- ❏ Field O_HINPUT is created by calling the HINPUT function to convert the alphanumeric date-time string stored in field O_HCNVRT to a date-time value and display it down to the nanosecond.
- ❏ Field O_HMASK is created by calling the HMASK function to extract the hours, minutes, seconds, milliseconds, microseconds, and nanoseconds from O_HEXTR and take the remaining components from TRANSDT.
- ❏ Field O_HMIDNT is created by calling the HMIDNT function to retrieve the date from TRANSDT and set the time to midnight.

- Field O_HNAME is created by calling the HNAME function to retrieve the nanosecond component from TRANSDT in alphanumeric format.
- Field O_HPART is created by calling the HPART function to retrieve the nanosecond component from TRANSDT in numeric format.
- Field O_HSETPT is created by calling the HSETPT function to set the nanoseconds component to the value 28.
- Field O_HTIME is created by calling the HTIME function to convert the time portion of TRANSDT to the number of nanoseconds.

Reference: Usage Notes for Nanosecond Date-Time Format Component

- ACTUAL formats for date-time fields can be up to H12. USAGE formats can be up to H23.
- The following date-time functions take a component name as an argument: HADD, HDIFF, HNAME, HPART, and HSETPT. The component name for nanoseconds for use in these date-time functions is *nanosecond*, which can also be abbreviated as *ns*. In addition, the HEXTR and HMASK functions have a new component, *n*, that represents the low order three digits of nine decimal digits.
- The following functions have arguments whose length depends on the precision of the date-time format:
  - HADD, HDIFF, HINPUT, HMIDNT, HSETPT, and HTIME have a length argument. The length can be 8, 10, or 12, where 12 is needed if the seconds value has more than six decimal digits.
  - HDTTM and HGETC have an argument for the length of the date-time value, which can be 8, 10, or 12, where 12 is needed if the seconds value has more than six decimal digits.
  - HCNVRT takes one argument that specifies the format of the date-time field to be converted to alphanumeric and one for the length of that field. The length of these arguments can be up to 23. The format for the output must be long enough to hold all of the characters returned.
  - HTIME converts the time portion of a date-time value to nanoseconds if the first argument is 12.

DATEPATTERN in the Master File

In some data sources, date values are stored in alphanumeric format without any particular standard, with any combination of components such as year, quarter, and month, and with any delimiter. In a sorted report, if such data is sorted alphabetically, the sequence does not make business sense. To ensure adequate sorting, aggregation, and reporting on date fields, DB2 Web Query can convert the alphanumeric dates into standard DB2 Web Query date format using a conversion pattern that you can specify in the Master File attribute called DATEPATTERN.

Each element in the pattern is either a constant character which must appear in the actual input or a variable that represents a date component. You must edit the USAGE attribute in the Master File so that it accounts for the date elements in the date pattern. The maximum length of the DATEPATTERN string is 64.

Specifying Variables in a Date Pattern

The valid date components (variables) are year, quarter, month, day, and day of week. In the date pattern, variables are enclosed in square brackets (these brackets are not part of the input or output). Note that if the data contains brackets, you must use an escape character in the date pattern to distinguish the brackets in the data from the brackets used for enclosing variables.

Syntax: **How to Specify Years in a Date Pattern**

`[YYYY]`
Specifies a 4-digit year.

`[YY]`
Specifies a 2-digit year.

`[yy]`
Specifies a zero-suppressed 2-digit year (for example, 8 for 2008).

`[by]`
Specifies a blank-padded 2-digit year.

Syntax: **How to Specify Month Numbers in a Date Pattern**

`[MM]`
Specifies a 2-digit month number.

`[mm]`
Specifies a zero-suppressed month number.

`[bm]`
Specifies a blank-padded month number.

Syntax: How to Specify Month Names in a Date Pattern

`[MON]`
: Specifies a 3-character month name in uppercase.

`[mon]`
: Specifies a 3-character month name in lowercase.

`[Mon]`
: Specifies a 3-character month name in mixed-case.

`[MONTH]`
: Specifies a full month name in uppercase.

`[month]`
: Specifies a full month name in lowercase.

`[Month]`
: Specifies a full month name in mixed-case.

Syntax: How to Specify Days of the Month in a Date Pattern

`[DD]`
: Specifies a 2-digit day of the month.

`[dd]`
: Specifies a zero-suppressed day of the month.

`[bd]`
: Specifies a blank-padded day of the month.

Syntax: How to Specify Julian Days in a Date Pattern

`[DDD]`
: Specifies a 3-digit day of the year.

`[ddd]`
: Specifies a zero-suppressed day of the year.

`[bdd]`
: Specifies a blank-padded day of the year.

Syntax: How to Specify Day of the Week in a Date Pattern

`[WD]`
: Specifies a 1-digit day of the week.

`[DAY]`
: Specifies a 3-character day name, uppercase.

`[day]`
: Specifies a 3-character day name, lowercase.

`[Day]`
: Specifies a 3-character day name, mixed-case.

`[WDAY]`
: Specifies a full day name, uppercase.

`[wday]`
: Specifies a full day name, lowercase.

`[Wday]`
: Specifies a full day name, mixed-case.

For the day of the week, the WEEKFIRST setting defines which day is day 1.

Syntax: How to Specify Quarters in a Date Pattern

`[Q]`
: Specifies a 1-digit quarter number (1, 2, 3, or 4).

  For a string like Q2 or Q02, use constants before [Q], for example, Q0[Q].

Specifying Constants in a Date Pattern

Between the variables, you can insert any constant values.

If you want to insert a character that would normally be interpreted as part of a variable, use the backslash character as an escape character. For example:

- ❑ Use \[to specify a left square bracket constant character.
- ❑ Use \\ to specify a backslash constant character.

For a single quotation mark, use two consecutive single quotation marks ('').

Example: Sample Date Patterns

If the date in the data source is of the form CY 2001 Q1, the DATEPATTERN attribute is:

```
DATEPATTERN = 'CY [YYYY] Q[Q]'
```

If the date in the data source is of the form Jan 31, 01, the DATEPATTERN attribute is:

```
DATEPATTERN = '[Mon] [DD], [YY]'
```

If the date in the data source is of the form APR-06, the DATEPATTERN attribute is:

```
DATEPATTERN = '[MON]-[YY]'
```

If the date in the data source is of the form APR - 06, the DATEPATTERN attribute is:

```
DATEPATTERN = '[MON] - [YY]'
```

If the date in the data source is of the form APR '06, the DATEPATTERN attribute is:

```
DATEPATTERN = '[MON] ''[YY]'
```

If the date in the data source is of the form APR [06], the DATEPATTERN attribute is:

```
DATEPATTERN = '[MON] \[[YY]\]' (or '[MON] \[[YY]]'
```

Note the right square bracket does not require an escape character.

Example: Sorting By an Alphanumeric Date

In the following example, date1.ftm is a sequential file containing the following data:

```
June 1, '02
June 2, '02
June 3, '02
June 10, '02
June 11, '02
June 12, '02
June 20, '02
June 21, '02
June 22, '02
June 1, '03
June 2, '03
June 3, '03
June 10, '03
June 11, '03
June 12, '03
June 20, '03
June 21, '03
June 22, '03
June 1, '04
June 2, '04
June 3, '04
June 4, '04
June 10, '04
June 11, '04
June 12, '04
June 20, '04
June 21, '04
June 22, '04
```

In the DATE1 Master File, the DATE1 field has alphanumeric USAGE and ACTUAL formats, each A18:

```
FILENAME=DATE1, SUFFIX=FIX,
  DATASET = c:\tst\date1.ftm, $
  SEGMENT=FILE1, SEGTYPE=S0, $
    FIELDNAME=DATE1, ALIAS=E01, USAGE=A18, ACTUAL=A18, $
```

The following request sorts by the DATE1 FIELD:

```
TABLE FILE DATE1
PRINT DATE1 NOPRINT
BY DATE1
ON TABLE SET PAGE NOPAGE
END
```

The output shows that the alphanumeric dates are sorted alphabetically, not chronologically:

```
DATE1
-----
June 1, '02
June 1, '03
June 1, '04
June 10, '02
June 10, '03
June 10, '04
June 11, '02
June 11, '03
June 11, '04
June 12, '02
June 12, '03
June 12, '04
June 2, '02
June 2, '03
June 2, '04
June 20, '02
June 20, '03
June 20, '04
June 21, '02
June 21, '03
June 21, '04
June 22, '02
June 22, '03
June 22, '04
June 3, '02
June 3, '03
June 3, '04
June 4, '04
```

In order to sort the data correctly, you can add a DATEPATTERN attribute to the Master File that enables DB2 Web Query to convert the date to a DB2 Web Query date field. You must also edit the USAGE format to make it a DB2 Web Query date format. To construct the appropriate pattern, you must account for all of the components in the stored date. The alphanumeric date has the following variables and constants:

- Variable: full month name in mixed-case, [Month].
- Constant: blank space.
- Variable: zero-suppressed day of the month number, [dd].

- ❏ Constant: comma followed by a blank space followed by an apostrophe (coded as two apostrophes in the pattern).
- ❏ Variable: two-digit year, [YY].

The edited Master File follows. Note the addition of the DEFCENT attribute to convert the two-digit year to a four-digit year:

```
FILENAME=DATE1, SUFFIX=FIX,
  DATASET = c:\tst\date1.ftm, $
  SEGMENT=FILE1, SEGTYPE=S0, $
    FIELDNAME=DATE1, ALIAS=E01, USAGE=A18, ACTUAL=A18, DEFCENT=20,
    DATEPATTERN = '[Month] [dd], ''[YY]', $
```

Now, issuing the same request produces the following output. Note that DATE1 has been converted to a DB2 Web Query date in MtrDYY format (as specified in the USAGE format):

```
DATE1
-----
June  1, 2002
June  2, 2002
June  3, 2002
June 10, 2002
June 11, 2002
June 12, 2002
June 20, 2002
June 21, 2002
June 22, 2002
June  1, 2003
June  2, 2003
June  3, 2003
June 10, 2003
June 11, 2003
June 12, 2003
June 20, 2003
June 21, 2003
June 22, 2003
June  1, 2004
June  2, 2004
June  3, 2004
June  4, 2004
June 10, 2004
June 11, 2004
June 12, 2004
June 20, 2004
June 21, 2004
June 22, 2004
```

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

The DMY, MDY, and YMD functions calculate the difference between two dates in integer, alphanumeric, or packed format.

Syntax: How to Calculate the Difference Between Two Dates

```
function(begin, end)
```

where:

`function`

Is one of the following:

`DMY` calculates the difference between two dates in day-month-year format.

`MDY` calculates the difference between two dates in month-day-year format.

`YMD` calculates the difference between two dates in year-month-day format.

`begin`

Integer, Packed, or Alphanumeric

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

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

`end`

Integer, Packed, or Alphanumeric

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

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

Example: Calculating the Number of Days Between Two Dates

YMD calculates the number of days between the dates in HIRE_DATE and DAT_INC:

```
COMPUTE DIFF/I4 = YMD(HIRE_DATE, FST.DAT_INC);
```

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.

Syntax: How to Find the Day of the Week

```
{DOWK|DOWKL}(indate, 'outfield')
```

where:

`indate`

Integer (I6YMD or I8 YMD)

Is the input date in year-month-day format. If the date is not valid, the function returns spaces. If the date specifies a two-digit year and DEFCENT and YRTHRESH values have not been set, the function assumes the 20th century.

outfield

DOWK: Alphanumeric

DOWKL: Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: **Finding the Day of the Week**

DOWK determines the day of the week that corresponds to the value in the HIRE_DATE field and stores the result in DATED:

```
COMPUTE DATED/A3 = DOWK(HIRE_DATE, 'A3');
```

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.

Syntax: **How to Convert an Integer to a Date**

```
function(number, 'outfield')
```

where:

function

Is one of the following:

DTDMY converts a number to a day-month-year date.

DTDYM converts a number to a day-year-month date.

DTMDY converts a number to a month-day-year date.

DTMYD converts a number to a month-year-day date.

DTYDM converts a number to a year-day-month date.

DTYMD converts a number to a year-month-day date.

number

Integer

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

outfield

Integer

I6*xxx*, where *xxx* corresponds to the function DT*xxx* in the above list.

Is the format of the output value enclosed in single quotation marks. The output format depends on the function being used.

Example: Converting an Integer to a Date

DTMDY converts the NEWF field (which was converted to the number of days by DAYMD) to the corresponding date and stores the result in NEW_HIRE_DATE:

```
COMPUTE NEWF/I8 WITH EMP_ID = DAYMD(HIRE_DATE, NEWF); AND
COMPUTE NEW_HIRE_DATE/I8MDYY WITH EMP_ID = DTMDY(NEWF, NEW_HIRE_DATE);
```

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.

Syntax: How to 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.

Example: Obtaining the Financial Year

The following 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.

```
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
```

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
```

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.

Syntax: How to 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.

Example: **Obtaining the Financial Quarter**

The following 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.

```
FISCALQ/Q=FIQTR(START_DATE,'D',10,1,'FYE',FISCALQ);
FISCALI/I1=FIQTR(START_DATE,'D',10,1,'FYE',FISCALI);
```

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
```

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.

Syntax: **How to 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.

Example: **Converting a Calendar Date to a Financial Date**

The following 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.

```
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);
```

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
```

GREGDT: Converting From Julian to Gregorian Format

The GREGDT function converts a date in Julian format to Gregorian format (year-month-day).

A date in Julian format is a five- or seven-digit number. The first two or four digits are the year; the last three digits are the number of the day, counting from January 1. For example, January 1, 1999 in Julian format is either 99001 or 1999001.

Reference: DATEFNS Settings for GREGDT

GREGDT converts a Julian date to either YMD or YYMD format using the DEFCENT and YRTHRESH parameter settings to determine the century, if required. GREGDT returns a date as follows:

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

Syntax: How to Convert From Julian to Gregorian Format

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

where:

indate

Integer (I5 or I7)

Is the Julian date, which is truncated to an integer before conversion. Each value must be a five- or seven-digit number after truncation. If the date is invalid, the function returns a 0.

outfield

Integer (I6, I8, I6YMD, or I8YYMD)

Is the format of the output value enclosed in single quotation marks.

Example: Converting From Julian to Gregorian Format

GREGDT converts the JULIAN field to YYMD (Gregorian) format.

```
COMPUTE GREG_DATE/I8 = GREGDT(JULIAN, 'I8');
```

HADD: Incrementing a Date-Time Value

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

Syntax: How to Increment a Date-Time Value

```
HADD(value, 'component', increment, length, 'outfield')
```

where:

value

Date-time

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

component

Alphanumeric

Is the name of the component to be incremented enclosed in single quotation marks.

Note: WEEKDAY is not a valid component for HADD.

increment

Integer

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

length

Integer

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

`8` indicates a time value that includes milliseconds.

`10` indicates a time value that includes microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

Example: Incrementing the Month Component of a Date-Time Field

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.

```
COMPUTE ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS');
```

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.

Syntax: How to Convert a Date-Time Value to Alphanumeric Format

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

where:

value

Date-time

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

fmt

Alphanumeric

Is the format of the date-time field enclosed in single quotation marks and parentheses.

length

Integer

Is the length of the alphanumeric field that is returned. You can supply the actual value, the name of a numeric field that contains the value, or an expression that returns the value. If *length* is smaller than the number of characters needed to display the alphanumeric field, the function returns a blank.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Converting a Date-Time Field to Alphanumeric Format

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.

```
COMPUTE ALPHA_DATE_TIME1/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20'); AND
COMPUTE ALPHA_DATE_TIME2/A20 = HCNVRT(TRANSDATE, '(HYYMDS)', 20, 'A20');
```

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.

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

```
HDATE(value, 'YYMD')
```

where:

value
: Date-time

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

YYMD
: Date

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

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

HDATE converts the date portion of the TRANSDATE field to the date format YYMD:

```
COMPUTE TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD');
```

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

The HDIFF function calculates the number of units between two date-time values.

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

```
HDIFF(value1, value2, 'component', 'outfield')
```

where:

value1
: Date-time

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

value2
: Date-time

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

component
: Alphanumeric

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

outfield

Floating point or Decimal

Is the format of the output value enclosed in single quotation marks.

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

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:

```
COMPUTE ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS'); AND
COMPUTE DIFF_DAYS/D12.2 = HDIFF(ADD_MONTH, TRANSDATE, 'DAY', 'D12.2');
```

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

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

Syntax: How to Convert a Date Value to a Date-Time Value

```
HDTTM(date, length, 'outfield')
```

where:

date

Date

Is the date value to be converted, the name of a date field that contains the value, or an expression that returns the value.

length

Integer

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

`8` indicates a time value that includes milliseconds.

`10` indicates a time value that includes microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

Example: Converting a Date Field to a Date-Time Field

HDTTM converts the date field TRANSDATE_DATE to a date-time field:

```
COMPUTE TRANSDATE_DATE/YYMD = HDATE(TRANSDATE, 'YYMD'); AND
COMPUTE DT2/HYYMDIA = HDTTM(TRANSDATE_DATE, 8, 'HYYMDIA');
```

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

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

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

```
HGETC(length, 'outfield')
```

where:

length

Integer

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

8 indicates a time value that includes milliseconds.

10 indicates a time value that includes microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

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

HGETC stores the current date and time in DT2:

```
COMPUTE DT2/HYYMDm = HGETC(10, 'HYYMDm');
```

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.

Syntax: How to Retrieve the Current Time

```
HHMMSS('outfield')
```

where:

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks. The field format must be at least A8.

Example: Retrieving the Current Time

HHMMSS retrieves the current time:

```
COMPUTE NOWTIME/A8 = HHMMSS('A8');
```

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

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

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

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

where:

inputlength

Integer

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

inputstring

Alphanumeric

Is the alphanumeric string to be converted enclosed in single quotation marks, the name of an alphanumeric field that contains the string, or an expression that returns the string. The string can consist of any valid date-time input value.

length

Integer

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

`8` indicates a time value that includes milliseconds.

`10` indicates a time value that includes microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

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

HCNVRT converts the TRANSDATE field to alphanumeric format, then HINPUT converts the alphanumeric string to a date-time value:

```
COMPUTE ALPHA_DATE_TIME/A20 = HCNVRT(TRANSDATE, '(H17)', 17, 'A20'); AND
COMPUTE DT_FROM_ALPHA/HYYMDS = HINPUT(14, ALPHA_DATE_TIME, 8, 'HYYMDS');
```

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.

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

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

where:

value

Date-time

Is the date-time value whose time is to be set to midnight, the name of a date-time field that contains the value, or an expression that returns the value.

length

Integer

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

`8` indicates a time value that includes milliseconds.

`10` indicates a time value that includes microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

Example: Setting the Time to Midnight

HMIDNT sets the time portion of the TRANSDATE field to midnight first in the 24-hour system and then in the 12-hour system:

```
COMPUTE TRANSDATE_MID_24/HYYMDS  = HMIDNT(TRANSDATE, 8, 'HYYMDS'); AND
COMPUTE TRANSDATE_MID_12/HYYMDSA = HMIDNT(TRANSDATE, 8, 'HYYMDSA');
```

HNAME: Retrieving a Date-Time Component in Alphanumeric Format

The HNAME function extracts a specified component from a date-time value in alphanumeric format.

Syntax: How to Retrieve a Date-Time Component in Alphanumeric Format

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

where:

value

Date-time

Is the date-time value from which a component is to be extracted, the name of a date-time field 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.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks. The field format must be at lease A2.

The function converts all other components to strings of digits only. The year is always four digits, and the hour assumes the 24-hour system.

Example: Retrieving the Week Component in Alphanumeric Format

HNAME returns the week in alphanumeric format from the TRANSDATE field. Changing the WEEKFIRST parameter setting changes the value of the component.

```
COMPUTE WEEK_COMPONENT/A10 = HNAME(TRANSDATE, 'WEEK', 'A10');
```

Example: Retrieving the Day Component in Alphanumeric Format

HNAME retrieves the day in alphanumeric format from the TRANSDATE field:

```
COMPUTE DAY_COMPONENT/A2 = HNAME(TRANSDATE, 'DAY', 'A2');
```

HPART: Retrieving a Date-Time Component in Numeric Format

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

Syntax: How to Retrieve a Date-Time Component in Numeric Format

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

where:

value

Date-time

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

component

Alphanumeric

Is the name of the component to be retrieved enclosed in single quotation marks.

outfield

Integer

Is the format of the output value enclosed in single quotation marks.

Example: Retrieving the Day Component in Numeric Format

HPART retrieves the day in integer format from the TRANSDATE field:

```
COMPUTE DAY_COMPONENT/I2 = HPART(TRANSDATE, 'DAY', 'I2');
```

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.

Syntax: How to Insert a Component Into a Date-Time Value

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

where:

dtfield

Date-time

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

component

Alphanumeric

Is the name of the component to be inserted enclosed in single quotation marks.

value

Integer

Is the numeric value to be inserted for the requested component, the name of a numeric field that contains the value, or an expression that returns the value.

length

Integer

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

`8` indicates a time value that includes milliseconds.

`10` indicates a time value that includes microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

Example: Inserting the Day Component Into a Date-Time Field

HSETPT inserts the day as 28 into the ADD_MONTH field and stores the result in INSERT_DAY:

```
COMPUTE ADD_MONTH/HYYMDS = HADD(TRANSDATE, 'MONTH', 2, 8, 'HYYMDS'); AND
COMPUTE INSERT_DAY/HYYMDS = HSETPT(ADD_MONTH, 'DAY', 28, 8, 'HYYMDS');
```

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 first argument is eight, or microseconds if the first argument is ten. To include microseconds, the input date-time value must be 10-bytes.

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

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

where:

length

Integer

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

`8` indicates a time value that includes milliseconds.

`10` indicates a time value that includes microseconds.

value

Date-time

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

outfield

Floating point or Decimal

Is the format of the output value enclosed in single quotation marks.

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

HTIME converts the time portion of the TRANSDATE field to the number of milliseconds:

```
COMPUTE MILLISEC/D12.2 = HTIME(8, TRANSDATE, 'D12.2');
```

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.

Reference: DATEFNS Settings for JULDAT

JULDAT converts a Gregorian date to either YYNNN or YYYYNNN format, using the DEFCENT and YRTHRESH parameter settings to determine if the century is required.

JULDAT returns dates as follows:

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

Syntax: **How to Convert From Gregorian to Julian Format**

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

where:

indate
Integer (I6, I8, I6YMD, I8YYMD)

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

outfield
Integer (I5 or I7)

Is the format of the output value enclosed in single quotation marks.

Example: **Converting From Gregorian to Julian Format**

JULDAT converts the HIRE_DATE field to Julian format.

```
COMPUTE JULIAN/I7 = JULDAT(HIRE_DATE, 'I7');
```

TIMETOTS: Converting a Time to a Timestamp

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

Syntax: **How to Convert a Time to a Timestamp**

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

where:

time
Date-time

Is the time in a date-time format.

length

Integer

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

8 for time values including milliseconds.

10 for input time values including microseconds.

outfield

Date-time

Is the format of the output value enclosed in single quotation marks.

Example: Converting a Time to a Timestamp

TIMETOTS converts a time argument to a timestamp:

```
COMPUTE TSTMPSEC/HYYMDS = TIMETOTS(TMSEC, 8, 'HYYMDS'); AND
COMPUTE TSTMPMILLI/HYYMDm = TIMETOTS(TMMILLI, 10, 'HYYMDm');
```

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.

Syntax: How to Retrieve the Current Date

```
TODAY('outfield')
```

where:

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks. The field format must be at least A8. The following apply:

- If DATEFNS=ON and the format is A8 or A9, TODAY returns the 2-digit year.
- If DATEFNS=ON and the format is A10 or greater, TODAY returns the 4-digit year.
- If DATEFNS=OFF, TODAY returns the 2-digit year, regardless of the format of *outfield*.

Example: Retrieving the Current Date

TODAY retrieves the current date and stores it in the DATE field.

```
COMPUTE DATE/A10 = TODAY('A10');
```

YM: Calculating Elapsed Months

The YM function calculates the number of months that elapse between two dates. The dates must be in year-month format. You can convert a date to this format by using the CHGDAT or EDIT function.

Syntax: How to Calculate Elapsed Months

```
YM(fromdate, todate, 'outfield')
```

where:

fromdate

Integer (I4YM or I6YYM)

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

todate

Integer (I4YM or I6YYM)

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

outfield

Integer

Is the format of the output value enclosed in single quotation marks.

Note: If *fromdate* or *todate* is in integer year-month-day format (I6YMD or I8YYMD), simply divide by 100 to convert to year-month format and set the result to an integer. This drops the day portion of the date, which is now after the decimal point.

Example: Calculating Elapsed Months

The COMPUTE commands convert the dates from year-month-day to year-month format; then YM calculates the difference between the values in the HIRE_DATE/100 and DAT_INC/100 fields:

```
COMPUTE HIRE_MONTH/I4YM = HIRE_DATE/100; AND
COMPUTE MONTH_INC/I4YM = DAT_INC/100; AND
COMPUTE MONTHS_HIRED/I3 = YM(HIRE_MONTH, MONTH_INC, 'I3');
```

Chapter 6

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. Literal date-time values can be used with the DT function.

All arguments can be either literals, field names, or amper variables.

In this chapter:

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.

Syntax: **How to 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.

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

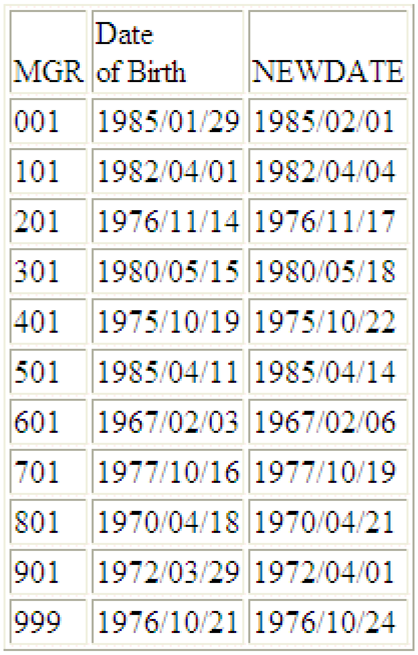
Is the value (positive or negative) to add to the component.

Example: **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 WF_RETAIL
NEWDATE/YYMD = DTADD(DATE_OF_BIRTH, DAY, 3);
MGR/A3 = DIGITS(ID_MANAGER, 3);
END
TABLE FILE WF_RETAIL
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

Reference: 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.

Syntax: How to Return the Number of Component Boundaries

```
DTDIFF(end_date, start_date, component)
```

where:

end_date

Date or date-time

Is the ending 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 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:

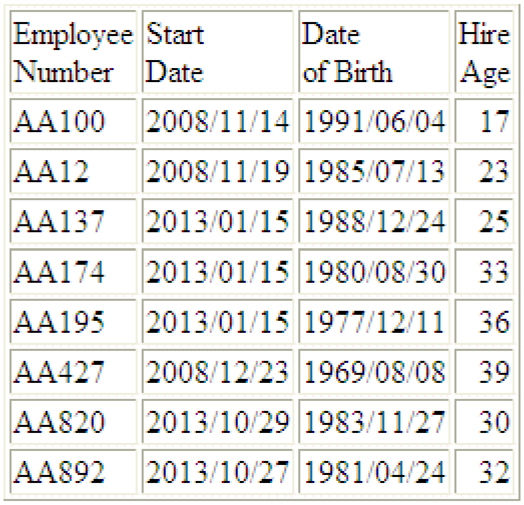
- ❏ YEAR (1-9999).
- ❏ QUARTER (1-4).
- ❏ MONTH (1-12).
- ❏ WEEK (1-53). This is affected by the WEEKFIRST setting.
- ❏ DAY (of the Month, 1-31).
- ❏ HOUR (0-23).
- ❏ MINUTE (0-59).
- ❏ SECOND (0-59).

Example: Returning the Number of Years Between Two Dates

The following request against the WF_RETAIL data source calculates employee age when hired:

```
DEFINE FILE WF_RETAIL
YEARS/I9 = DTDIFF(START_DATE, DATE_OF_BIRTH, YEAR);
END
TABLE FILE WF_RETAIL
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

DTPART: Returning a Date or Date-Time Component in Integer Format

Given a date in standard date or date-time format and a component, DTPART returns the component value in integer format.

Syntax: How to Return a Date or Date-Time Component in Integer Format

```
DTPART(date, component)
```

where:

`date`

Date or date-time

Is the date in standard date or date-time format.

component

Keyword

Is the component to extract in integer format. Valid components (and values) are:

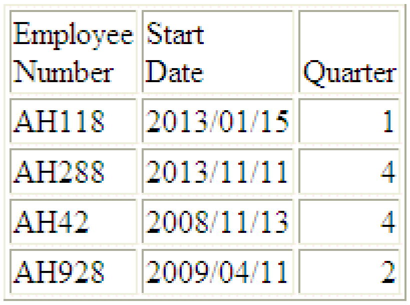
- YEAR (1-9999).
- QUARTER (1-4).
- MONTH (1-12).
- WEEK (of the year, 1-53). This is affected by the WEEKFIRST setting.
- DAY (of the Month, 1-31).
- DAY_OF_YEAR (1-366).
- WEEKDAY (day of the week, 1-7). This is affected by the WEEKFIRST setting.
- HOUR (0-23).
- MINUTE (0-59).
- SECOND (0-59).
- MILLISECOND (0-999).
- MICROSECOND (0-999999).

Example: 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 WF_RETAIL
QTR/I2 = DTPART(START_DATE, QUARTER);
END
TABLE FILE WF_RETAIL
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.

Syntax: How to Return the First or Last Date of a Date Period

```
DTRUNC(date_or_timestamp, date_period)
```

where:

date_or_timestamp

Date or date-time

Is the date or timestamp of interest.

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.

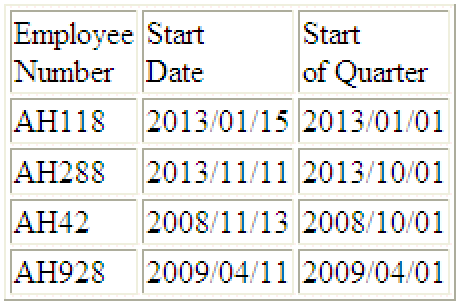
- YEAR_END, returns the last date of the year.
- QUARTER_END, returns the last date of the quarter.
- MONTH_END, returns the last date of the month.
- WEEK_END, returns the last date of the week.

Example: **Returning the First Date in a Date Period**

In the following request against the WF_RETAIL data source, DTRUNC returns the first date of the quarter given the start date of the employee:

```
DEFINE FILE WF_RETAIL
QTRSTART/YYMD = DTRUNC(START_DATE, QUARTER);
END
TABLE FILE WF_RETAIL
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

Chapter 7

Format Conversion Functions

Format conversion functions convert fields from one format to another.

In this chapter:

ATODBL: Converting an Alphanumeric String to Double-Precision Format

The ATODBL function converts a number in alphanumeric format to decimal (double-precision) format.

Syntax: How to Convert an Alphanumeric String to Double-Precision Format

```
ATODBL(string, length, 'outfield')
```

where:

string

Alphanumeric

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

length

Alphanumeric

Is the two-character length of *infield* in bytes. This can be a numeric constant, or a field that contains the value. If you specify a numeric constant, enclose it in single quotation marks. The maximum value is 15.

outfield

Decimal

Is the format of the output value enclosed in single quotation marks.

Example: **Converting an Alphanumeric Field to Double-Precision Format**

ATODBL converts the EMP_ID field into double-precision format and stores the result in D_EMP_ID:

```
COMPUTE D_EMP_ID/D12.2 = ATODBL(EMP_ID, '09', 'D12.2');
```

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. It is useful when you need to manipulate a field using a command that requires a particular format.

When EDIT assigns a converted value to a new field, the format of the new field must correspond to the format of the returned value. For example, if EDIT converts a numeric field to alphanumeric format, you must give the new field an alphanumeric format:

```
DEFINE ALPHAPRICE/A6 = EDIT(PRICE);
```

EDIT deals with a symbol in the following way:

- When an alphanumeric field is converted to numeric format, a sign or decimal point in the field is acceptable and is stored in the numeric field.
- When converting a floating-point or packed-decimal field to alphanumeric format, EDIT removes the sign, the decimal point, and any number to the right of the decimal point. It then right-justifies the remaining digits and adds leading zeros to achieve the specified field length. Converting a number with more than nine significant digits in floating-point or packed-decimal format may produce an incorrect result.

EDIT also extracts characters from or adds characters to an alphanumeric string. For more information, see *EDIT: Extracting or Adding Characters*.

Syntax: **How to Convert the Format of a Field**

```
EDIT(fieldname);
```

where:

fieldname

Alphanumeric or Numeric

Is the field name.

Example: **Converting From Numeric to Alphanumeric Format**

EDIT converts HIRE_DATE (a legacy date format) to alphanumeric format. CHGDAT is then able to use the field, which it expects in alphanumeric format:

```
COMPUTE ALPHA_HIRE/A17 = EDIT(HIRE_DATE); AND
COMPUTE HIRE_MDY/A17 = CHGDAT('YMD', 'MDYYX', ALPHA_HIRE, 'A17');
```

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 a number and right-justifies it with leading spaces. You can also add edit options to a number converted by FTOA.

When using FTOA to convert a number containing decimals to a character string, you must specify an alphanumeric format large enough to accommodate both the integer and decimal portions of the number. For example, a D12.2 format is converted to A14. If the output format is not large enough, decimals are truncated.

Syntax: **How to Convert a Number to Alphanumeric Format**

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

where:

number

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

Is the number to be converted, or the name of the field that contains the number.

format

Alphanumeric

Is the output format of the number enclosed in both single quotation marks and parentheses. Only floating point single-precision and double-precision formats are supported. Include any edit options that you want to appear in the output. The D (floating-point double-precision) format automatically supplies commas.

If you use a field name for this argument, specify the name without quotation marks or parentheses. If you specify a format, the format must be enclosed in parentheses.

outfield

Alphanumeric

Is the 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.

Example: **Converting From Numeric to Alphanumeric Format**

FTOA converts the GROSS field from floating point double-precision to alphanumeric format and stores the result in ALPHA_GROSS:

```
COMPUTE ALPHA_GROSS/A15 = FTOA(GROSS, '(D12.2)', 'A15');
```

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. It returns a single alphanumeric character in the ASCII, EBCDIC, or Unicode character set. You can use this function to produce characters that are not on your keyboard, similar to the CTRAN function.

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

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

The display of special characters depends on your software and hardware; not all special characters may appear.

Syntax: **How to Convert a Decimal Integer to a Character**

```
HEXBYT(input, 'outfield')
```

where:

input

Integer

Is the decimal integer to be converted to a single character. In non-Unicode environments, a value greater than 255 is treated as the remainder of *input* divided by 256.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Converting a Decimal Integer to a Character

HEXBYT converts LAST_INIT_CODE to its character equivalent and stores the result in LAST_INIT:

```
COMPUTE LAST_INIT_CODE/I3 = BYTVAL(LAST_NAME, 'I3'); AND
COMPUTE LAST_INIT/A1 = HEXBYT(LAST_INIT_CODE, 'A1');
```

ITONUM: Converting a Large Binary Integer to Double-Precision Format

The ITONUM function converts a large binary integer in a data source to double-precision format. Some programming languages and some data storage systems use large binary integer formats. However, large binary integers (more than 4 bytes in length) are not supported in the Master File so they require conversion to double-precision format.

You must specify how many of the right-most bytes in the input field are significant. The result is an 8-byte double-precision field.

Syntax: How to Convert a Large Binary Integer to Double-Precision Format

```
ITONUM(maxbytes, infield, 'outfield')
```

where:

maxbytes

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign. Valid values are:

5ignores the left-most 3 bytes.

6ignores the left-most 2 bytes.

7ignores the left-most byte.

infield

Alphanumeric

Is the field that contains the binary number. Both the USAGE and ACTUAL formats of the field must be A8.

outfield

Decimal

Is the format of the output value enclosed in single quotation marks. The format must be D*n*.

Example: Converting a Large Binary Integer to Double-Precision Format

ITONUM converts the BINARYFLD field to double-precision format:

```
COMPUTE MYFLD/D14 = ITONUM(6, BINARYFLD, 'D14');
```

ITOPACK: Converting a Large Binary Integer to Packed-Decimal Format

The ITOPACK function converts a large binary integer in a data source to packed-decimal format. Some programming languages and some data storage systems use double-word binary integer formats. Large binary integers (more than 4 bytes in length) are not supported in the Master File so they require conversion to packed-decimal format.

You must specify how many of the right-most bytes in the input field are significant. The result is an 8-byte packed-decimal field of up to 15 significant numeric positions (for example, P15 or P16.2).

Limit: For a field defined as 'PIC 9(15) COMP' or the equivalent (15 significant digits), the maximum number that can be converted is 167,744,242,712,576.

Syntax: How to Convert a Large Binary Integer to Packed-Decimal Format

```
ITOPACK(maxbytes, infield, 'outfield')
```

where:

maxbytes

Numeric

Is the maximum number of bytes in the 8-byte binary input field that have significant numeric data, including the binary sign.

Valid values are:

5 ignores the left-most 3 bytes (up to 11 significant positions).

6 ignores the left-most 2 bytes (up to 14 significant positions).

7 ignores the left-most byte (up to 15 significant positions).

infield

Alphanumeric

Is the field that contains the binary number. Both the USAGE and ACTUAL formats of the field must be A8.

outfield

Packed

Is the format of the output value enclosed in single quotation marks. The format must be P*n* or P*n*.*d*.

Example: Converting a Large Binary Integer to Packed-Decimal Format

ITOPACK converts the BINARYFLD field to packed-decimal format:

```
COMPUTE PACKFLD/P14.4 = ITOPACK(6, BINARYFLD, 'P14.4');
```

ITOZ: Converting a Number to Zoned Format

The ITOZ function converts a number in numeric format to zoned format. Although a request cannot process zoned numbers, it can write zoned fields to an extract file for use by an external program.

Syntax: How to Convert to Zoned Format

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

where:

outlength

Integer

Is the length of *number* in bytes. The maximum number of bytes is 15. The last byte includes the sign.

number

Numeric

Is the number to be converted, or the field that contains the number. The number is truncated to an integer before it is converted.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: Converting a Number to Zoned Format

ITOZ converts the CURR_SAL field to a zoned format:

```
COMPUTE ZONE_SAL/A8 = ITOZ(8, CURR_SAL, 'A8');
```

PCKOUT: Writing a Packed Number of Variable Length

The PCKOUT function writes a packed number of variable length to an extract file. When a request saves a packed number to an extract file, it typically writes it as an 8- or 16-byte field regardless of its format specification. With PCKOUT, you can vary the field's length between 1 to 16 bytes.

Syntax: How to Write a Packed Number of Variable Length

```
PCKOUT(infield, outlength, 'outfield')
```

where:

infield

Numeric

Is the input field that contains the values. The field can be in packed, integer, floating-point, or double-precision format. If the field is not in integer format, its values are rounded to the nearest integer.

outlength

Numeric

Is the length of outfield from 1 to 16 bytes.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks. The function returns the field as alphanumeric although it contains packed data.

Example: Writing a Packed Number of Variable Length

PCKOUT converts the CURR_SAL field to a 5-byte packed field and stores the result in SHORT_SAL:

```
COMPUTE SHORT_SAL/A5 = PCKOUT(CURR_SAL, 5, 'A5');
```

Chapter 8

Numeric Functions

Numeric functions perform calculations on numeric constants and fields.

In this chapter:

ABS: Calculating Absolute Value

The ABS function returns the absolute value of a number.

Syntax: How to Calculate Absolute Value

```
ABS(argument)
```

where:

argument

Numeric

Is the value for which the absolute value is returned, the name of a field that contains the value, or an expression that returns the value. If you use an expression, use parentheses as needed to ensure the correct order of evaluation.

Example: Calculating Absolute Value

The first COMPUTE command creates the DIFF field, then ABS calculates the absolute value of DIFF:

```
COMPUTE DIFF/I5 = DELIVER_AMT - UNIT_SOLD; AND
COMPUTE ABS_DIFF/I5 = ABS(DIFF);
```

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.

Syntax: How to Produce a Bar Chart

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

where:

barlength
Numeric

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

infield
Numeric

Is the data field plotted as a bar chart.

maxvalue
Numeric

Is the maximum value of a bar. This value must be greater than the maximum value stored in *infield*. If *infield* is larger than *maxvalue*, the function uses *maxvalue* and returns a bar of maximum length.

char
Alphanumeric

Is the repeating character that creates the bars enclosed in single quotation marks. If you specify more than one character, only the first character is used.

outfield
Alphanumeric

Is the 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*.

Example: Producing a Bar Chart

BAR creates a bar chart for the CURR_SAL field, and stores the output in SAL_BAR. The bar created can be no longer than 30 characters long, and the value it represents can be no greater than 30,000.

```
COMPUTE SAL_BAR/A30 = BAR(30, CURR_SAL, 30000, '=', SAL_BAR);
```

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) defines the field as alphanumeric, not packed. This does *not* change the field data, which remains packed, but it enables the request to read the data without a data exception.
2. Call CHKPCK to examine the field. The function returns the output to a field defined as packed. If the value it examines is a valid packed number, the function returns the value; if the value is not packed, the function returns an error code.

Syntax: How to Validate a Packed Field

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

where:

inlength

Numeric

Is the length of the packed field. It can be between 1 and 16 bytes.

infield

Alphanumeric

Is the name of the packed field. The field is described as alphanumeric, not packed.

error

Numeric

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

`outfield`

Packed

Is the format of the output value enclosed in single quotation marks.

Example: **Validating Packed Data**

CHKPCK validates 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.

```
COMPUTE GOOD_PACK/P8CM = CHKPCK(8, PACK_SAL, -999, GOOD_PACK);
```

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.

For information on the INT function, see *INT: Finding the Greatest Integer*.

Syntax: **How to Calculate the Remainder From a Division**

```
function(dividend, divisor, 'outfield')
```

where:

`function`

Is one of the following:

`DMOD` returns the remainder as a decimal number.

`FMOD` returns the remainder as a floating point number.

`IMOD` returns the remainder as an integer.

dividend

Numeric

Is the number being divided.

divisor

Numeric

Is the number dividing the dividend.

outfield

Numeric

Is the format of the output value enclosed in single quotation marks. The format is determined by the result returned by the specific function.

Example: Calculating the Remainder From a Division

IMOD divides ACCTNUMBER by 1000 and returns the remainder to LAST3_ACCT:

```
COMPUTE LAST3_ACCT/I3L = IMOD(ACCTNUMBER, 1000, LAST3_ACCT);
```

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 an argument's logarithm.

EXP calculates the result by adding terms of an infinite series. If a term adds less than .000001 percent to the sum, the function ends the calculation and returns the result as a double-precision number.

Syntax: How to Raise *e* to the Nth Power

```
EXP(power, 'outfield')
```

where:

power

Numeric

Is the power to which "e" is raised.

outfield

Floating point or Decimal

Is the format of the output value enclosed in single quotation marks.

Example: Raising *e* to the Nth Power

EXP raises e to the 2nd power:

```
COMPUTE E2/D12.2 = EXP(2, 'D12.2');
```

INT: Finding the Greatest Integer

The INT function returns the integer component of a number.

Syntax: How to Find the Greatest Integer

```
INT(argument)
```

where:

argument

Numeric

Is the value for which the integer component is returned, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation.

Example: Finding the Greatest Integer

INT finds the greatest integer in the DED_AMT field and stores it in INT_DED_AMT:

```
COMPUTE INT_DED_AMT/I9 = INT(DED_AMT);
```

LOG: Calculating the Natural Logarithm

The LOG function returns the natural logarithm of a number.

Syntax: How to Calculate the Natural Logarithm

```
LOG(argument)
```

where:

argument

Numeric

Is the value for which the natural logarithm is calculated, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation. If *argument* is less than or equal to 0, LOG returns 0.

Example: Calculating the Natural Logarithm

LOG calculates the logarithm of the CURR_SAL field:

```
COMPUTE LOG_CURR_SAL/D12.2 = LOG(CURR_SAL);
```

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.

Syntax: How to Find the Maximum or Minimum Value

```
{MAX|MIN}(argument1, argument2, ...)
```

where:

`MAX`
: Returns the maximum value.

`MIN`
: Returns the minimum value.

`argument1, argument2`
: Numeric

  Are the values for which the maximum or minimum value is returned, the name of a field that contains the values, or an expression that returns the values. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation.

Example: Determining the Minimum Value

MIN returns either the value of the ED_HRS field or the constant 30, whichever is lower:

```
COMPUTE MIN_EDHRS_30/D12.2 = MIN(ED_HRS, 30);
```

SQRT: Calculating the Square Root

The SQRT function calculates the square root of a number.

Syntax: How to Calculate the Square Root

```
SQRT(argument)
```

where:

`argument`
: Numeric

Is the value for which the square root is calculated, the name of a field that contains the value, or an expression that returns the value. If you supply an expression, use parentheses as needed to ensure the correct order of evaluation. If you supply a negative number, the result is zero.

Example: Calculating the Square Root

SQRT calculates the square root of LISTPR:

```
COMPUTE SQRT_LISTPR/D12.2 = SQRT(LISTPR);
```

Chapter 9

System Functions

System functions call the operating system to obtain information about the operating environment.

In this chapter:

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.

Syntax: **How to Retrieve the Value of an Environment Variable**

```
FGETENV(varlength, 'varname', outfieldlen, 'outfield')
```

where:

varlength

Integer

Is the length of the environment variable name.

varname

Alphanumeric

Is the name of the environment variable.

outfieldlen

Integer

Is the length of the field in which the environment variable's value is stored.

outfield

Alphanumeric

Is the format of the output value enclosed in single quotation marks.

Example: **Retrieving the Language Locale**

Using the LANG environment variable, FGETENV retrieves the object location for the language locale:

```
COMPUTE LANG_LOCALE/A40 = FGETENV(4, 'LANG', 40, 'A40');
```

GETUSER: Retrieving a User ID

The GETUSER function retrieves the ID of the connected user.

Syntax: How to Retrieve a User ID

```
GETUSER('outfield')
```

where:

outfield

Alphanumeric

Is the format (at least A8) of the output value enclosed in single quotation marks. The 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.

Example: Retrieving a User ID

GETUSER retrieves the user ID of the person running the request:

```
COMPUTE USERID/A8 = GETUSER('A8');
```

Index