

mitsdiscover

MIT S Discover Expressions

Building MITS Discover Expressions

Expressions are used within MITS Discover for describing the computations that need to take place on raw data in order to prepare that data for use as an identifier description or to populate a column. As MITS Discover expressions are compiled, the expression syntax generally follows that of the underlying BASIC syntax, but there are additional MITS Discover-specific extensions that allows the expression writer to more easily gain access to the underlying data stored within their MITS Discover system. This processing that occurs during expression compilation follows a specific set of steps:

1. Locate and process references to any Hypercube identifier IDs or abbreviations
2. Locate and process any MITS Discover functions defined in the MITS.FUNCTION.TEMPLATES or MITS.FUNCTION.TEMPLATES_SITE file. (Some commonly used functions included with the MITS Discover system are defined below, although you can also create custom functions.)
3. Process any BASIC syntax and/or standard arithmetic functions

NOTE: Many common arithmetic functions have been defined as MITS Discover functions, such as ADD, SUBTRACT, MULTIPLY, DIVIDE, TREND, and POT (Percent of Total). These functions include additional handling that can help to prevent common data-related errors. For example, the DIVIDE function verifies that neither of the values being processed are zero before processing which helps to avoid “divide by zero” errors. It is recommended that you utilize these included functions instead of their corresponding raw arithmetic operators.

Some Common Functions Included with MITS Discover

ACCUM(accumulator,eon)

The ACCUM function is the foundation for describing accumulator-based columns. The result of this function is the total of the amount referred to by (accumulator) for the (eon) time span.

ADD(value1,value2)

The ADD function adds two values together. It is often used to add the values in two ACCUM-based columns together to create a third column which will display the results. It is recommended that you create the two base columns using the ACCUM function, then use a pair of REF functions in the ADD expression to “REferencE” the values in the two original columns. For example, if you wanted to create a column that displayed the results of adding this year’s sales (SALES.Y) to last year’s sales (SALES.Y-1), the column expression would be:

```
ADD(REF(SALES.Y),REF(SALES.Y-1))
```

CALLSUB('programName', argumentData)

The CALLSUB function is used to call a custom BASIC subroutine that has been compiled and cataloged on the host system within the MITS Discover operational database account.

The CALLSUB function requires two arguments. The first argument is the name of the program that will be called. Note that this program name needs to be surrounded by single quotes. The second argument is the data that will be passed into the subroutine.

To use this function with a given subroutine, that subroutine must accept only two arguments. The first argument is the data that is passed into the routine, and the second is the result to pass back to MITS.

NOTE: If you need to call a subroutine from the Identifier Description Expression box within the Data Elements screen of MITSMaker, use the IDENT.CALLSUB function instead of the CALLSUB function.

DIVIDE(value1,value2)

In the same way that the ADD function will add two values together, the DIVIDE function will divide one value by another. You can use REF statements to “REFerence” the values in two existing columns to create a new column that displays the product of that division.

MULTIPLY(value1,value2)

In the same way that the ADD function will add two values together, the MULTIPLY function will multiply two values. You can use REF statements to “REFerence” the values in two existing columns to create a new column that displays the product of that multiplication.

READV(itemId,'fileName',attribute)

The READV function is most often used to provide an identifier's description (for example, a customer's name). An identifier abbreviation can be used in place of the itemId, and the file name should be surrounded by single quotes.

REF(column)

The REF function is used to derive the value already defined by another column definition, and is used when computing results based on multiple columns.

SUBTRACT(value1,value2)

The SUBTRACT function will subtract value1 from value2, displaying the result. Use REF statements to “REFerence” the values in two existing columns.

TOTAL(column)

This function produces the next-higher-level total for the column you specify, and is used for computing a percentage-of-total.

TREND(value1,value2)

This function produces the difference between value1 and value2 displayed as a percentage.

Expression Examples

An expression can be as simple as $2+29$ or "Hello Joe", but here are some more realistic examples:

Example 1: Using ACCUM

In this example, we will use the ACCUM function to retrieve sales totals for the current month:

```
ACCUM(SALES, M)
```

In the above expression, SALES refers to one of the Hypercube's accumulators and the M refers to the "month-to-date" eon. So, in theory, the column generated by this expression would display sales dollars for the current month.

Example 2: Using REF and ADD

The following column expression would produce a column that displays the total of this month's sales and last month's sales:

```
REF(SALES. M) + REF(SALES. M-1)
```

However, take note that we also could have used the ADD function as in the following example:

```
ADD(REF(SALES. M), REF(SALES. M-1))
```

Example 3: Using READV to Read a Value from a File

This example will demonstrate how to create a custom column to read in some data from a file on your source database. For the purpose of this example, we will pretend that the CUSTOMER.MASTER file on your source database is keyed by customer number and holds the customer's phone number on attribute 5. The following column expression (using the customer identifier abbreviation of "C") will allow you to display each customer's phone number in any flash screen that includes the customer identifier in the drill-down path:

```
READV(C, 'CUSTOMER.MASTER', 5)
```

Example 4: Identifier Description Expression Using READV

Let's look at an identifier description expression example. Imagine that the descriptions for a Product identifier could be found on attribute 1 of a file named INSTOCK.PRODUCTS or on attribute 1 of a file named NONSTOCK.PRODUCTS, depending on whether the product in question is in stock or not in stock. If you point MITS Discover at the INSTOCK.PRODUCTS file for the Product identifier description, then your flash screen will only display descriptions for items that happen to be on the INSTOCK.PRODUCTS file at the moment the flash screen is run (since MITS Discover retrieves identifier descriptions from the source data at the time the flash screen is requested). Any products that happen to currently be in the NONSTOCK.PRODUCTS file will simply have a blank description. Since no product will ever appear in both files, one way to handle this situation would be to create a compound identifier description expression using READV and concatenation:

```
READV(P, 'INSTOCK.PRODUCTS', 1): READV(P, 'NONSTOCK.PRODUCTS', 1)
```

In this case, both READV statements will be processed for each Product identifier row, and the results will be concatenated together (thanks to the “:” concatenation symbol). If the product is on the INSTOCK.PRODUCTS file, the first READV statement will retrieve the description from attribute 1 of the INSTOCK.PRODUCTS file and the second READV statement will effectively fail, returning nothing. In the same way, if the item currently resides in the NONSTOCK.PRODUCTS file, the first READV statement will fail and the second will succeed.

Example 5: Using Regular Expression Syntax

For this next example, assume that we already have a “PROFIT.M-1” column which displays the profit dollars for last month. Consider the following valid column expression:

```
'profitable' [1, 10*(REF(PROFIT.M-1)>0)]: 'loser' [1, 6*(REF(PROFIT.M-1)<0)]
```

This column expression will display a column value of “profitable” for rows where the first portion of the expression is valid and a column value of “loser” for rows where the second portion of the expression is valid. (In the case of a break-even situation, the column cell for that row would be blank.)