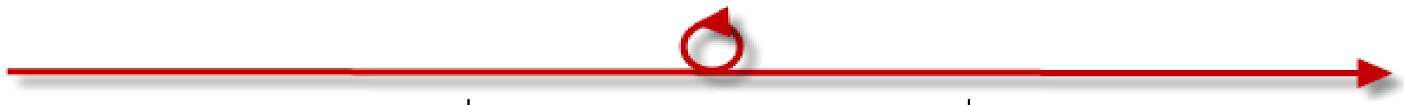
Tip Sheet

Topic: Expression Engine Language Programming **Date:** 06/15/2010 **Author name:** Jeff Bailey

Expression Node

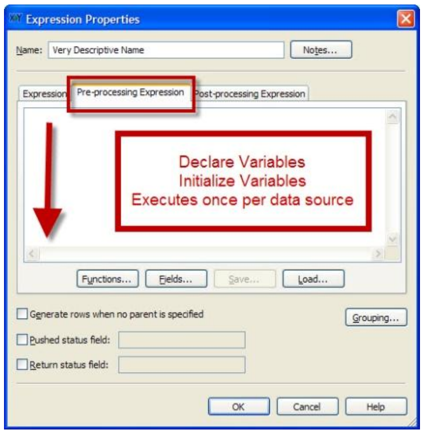
Create custom, user-written, DataFlux Expression Engine Language code for dfPower Architect jobs.

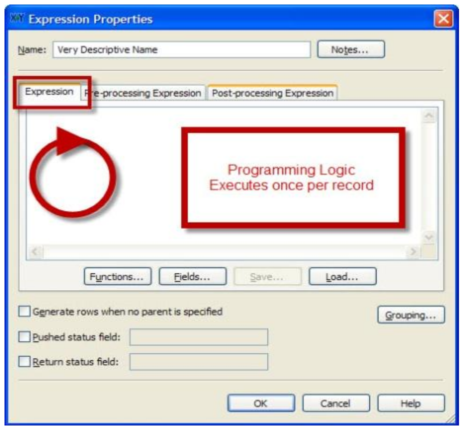


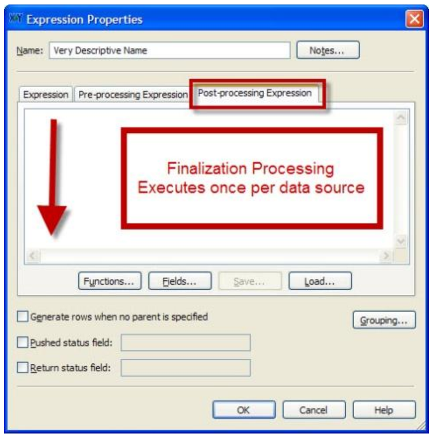
Pre-processing Expression Tab
Enter the code that should be executed before looping through the records in the input data source. You can use the Functions and Fields buttons to automatically include function calls and field names.

Expression Tab
Code written in the Expression Tab is executed for each record in the input data source. You can assign values to variable, call functions and use programming constructs (eg. loops
and IF/THEN/ELSE) here. For example: you may want to count values or keep a tally of error conditions. You can do that here.

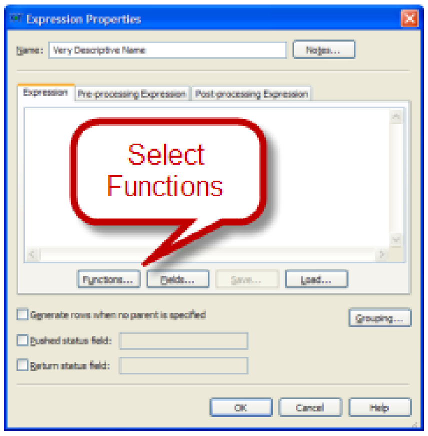
Post-processing Tab
The code included in the Post-processing Tab is executed last. Think of this as the clean up portion of your code. Data records are not being processed here. This is a good place to output any counters that you have been using and controlling output.

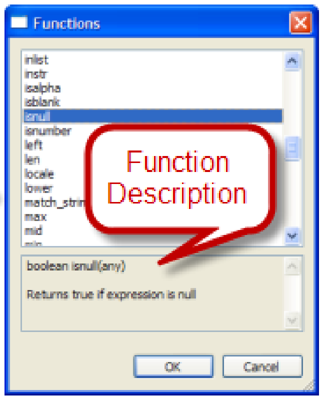

Name: Very Descriptive Name
Expression Properties
Expression
Pre-processing Expression
Post-processing Expression
Declare Variables
Initialize Variables
Executes once per data source
Functions... Fields... Save... Load...
Generate rows when no parent is specified
Pushed status field:
Return status field:
Grouping...
OK Cancel Help


Name: Very Descriptive Name
Expression Properties
Expression
Pre-processing Expression
Post-processing Expression
Programming Logic
Executes once per record
Functions... Fields... Save... Load...
OK Cancel Help

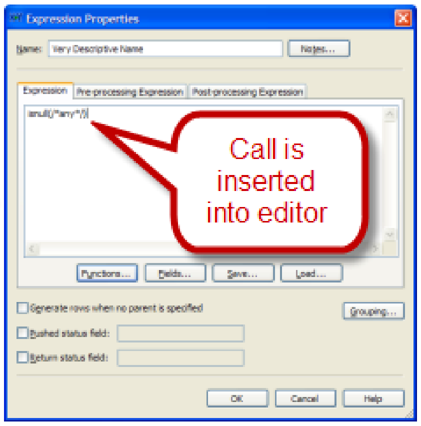

Name: Very Descriptive Name
Expression Properties
Expression
Pre-processing Expression
Post-processing Expression
Finalization Processing
Executes once per data source
Functions... Fields... Save... Load...
OK Cancel Help

Insert Function Call Syntax into Expression dialog

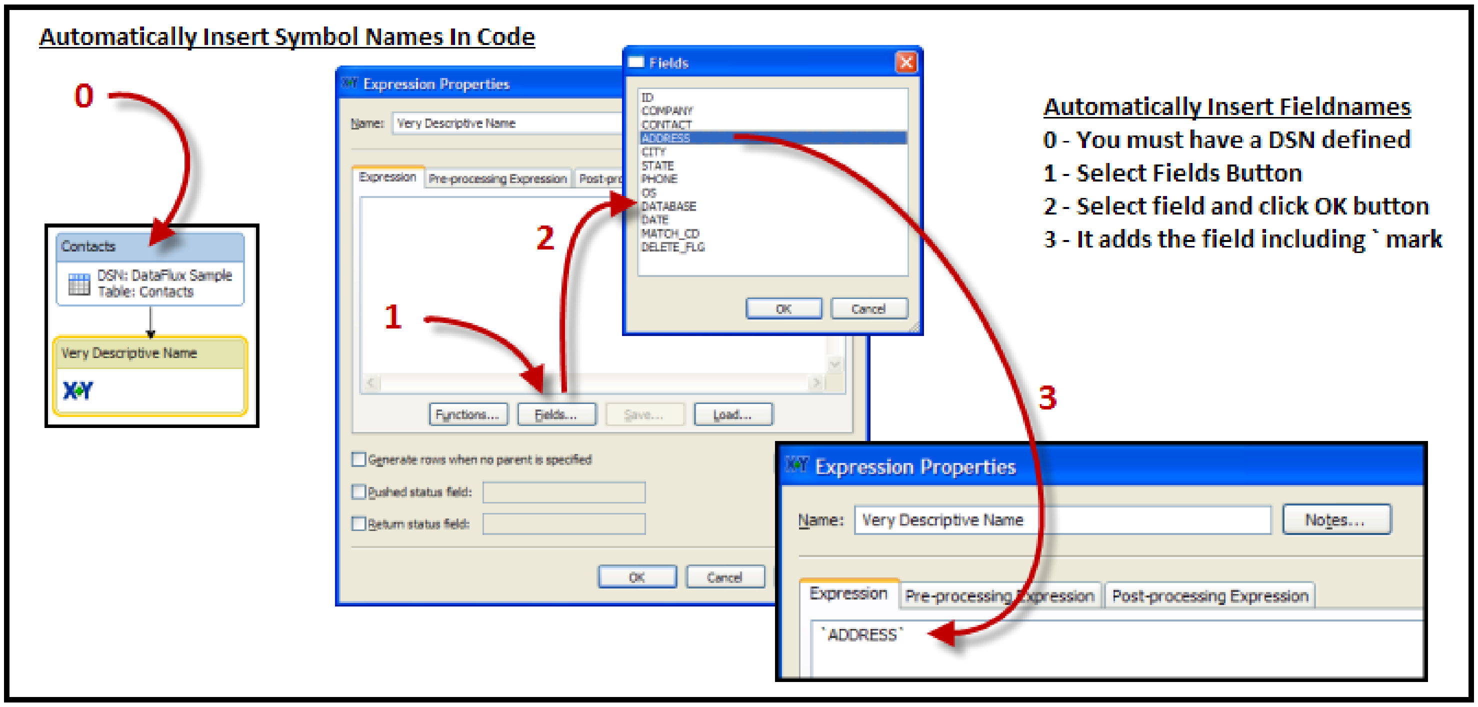



boolean isnull(any)

Returns true if expression is null


isnull("any")
Expression Properties
Call is inserted into editor
Functions... Fields... Save... Load...
OK Cancel Help

Note: many of the code samples and some of the text included in the Tip Sheet were directly copied from the **DataFlux Expression Language Reference Guide**.



Automatically Insert Symbol Names In Code

Automatically Insert Fieldnames

0 - You must have a DSN defined

1 - Select Fields Button

2 - Select field and click OK button

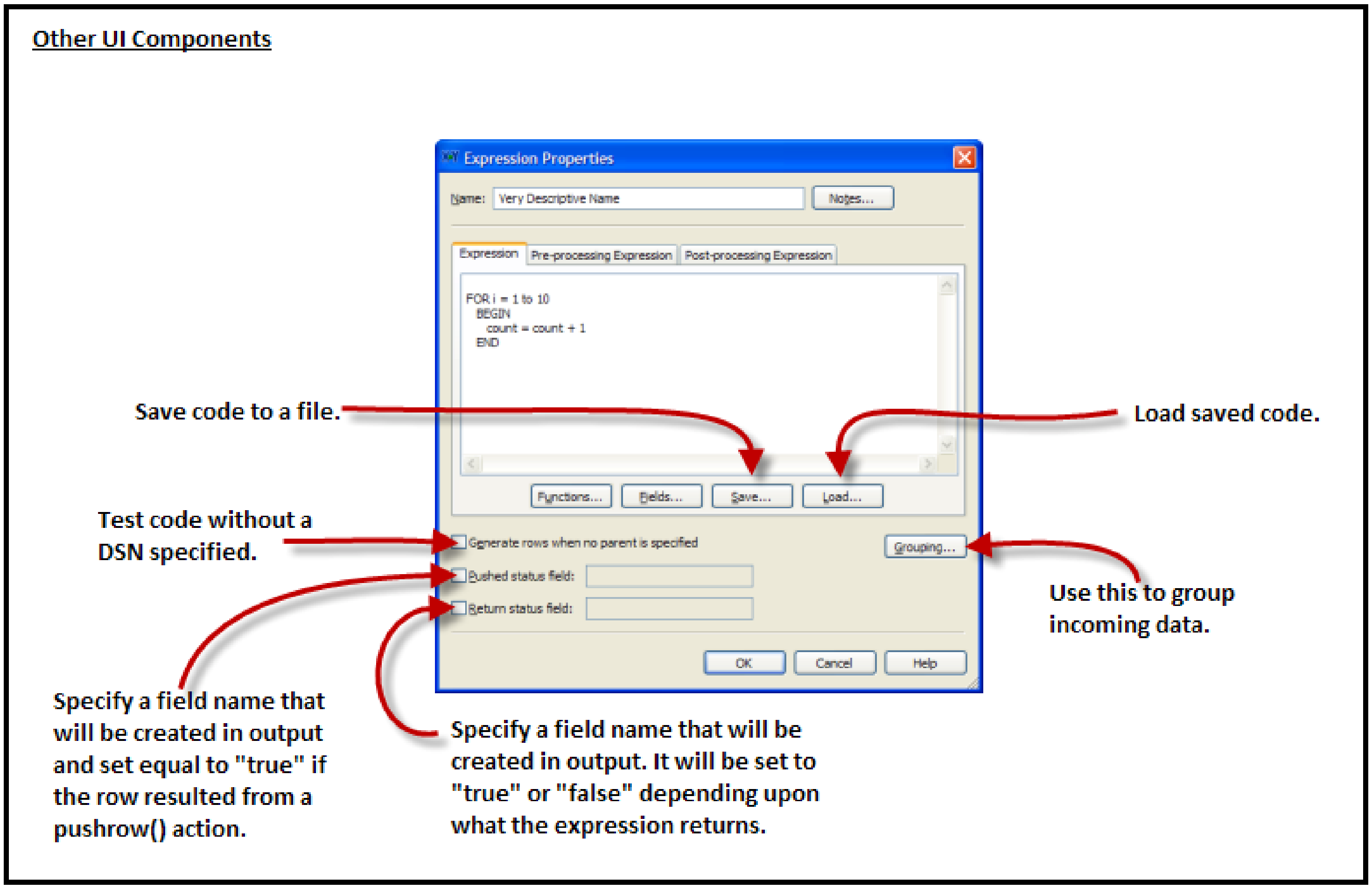
3 - It adds the field including ` mark

Name: Very Descriptive Name

```
`ADDRESS`
```

Other UI Components

Here are some of the other tasks that can be accomplished by the Expression node. Please see the help for more info.



Other UI Components

Name: Very Descriptive Name

```
FOR i = 1 to 10
  BEGIN
    count = count + 1
  END
```

Save code to a file.

Load saved code.

Test code without a DSN specified.

Use this to group incoming data.

Specify a field name that will be created in output and set equal to "true" if the row resulted from a pushrow() action.

Specify a field name that will be created in output. It will be set to "true" or "false" depending upon what the expression returns.

Declarations

Variables (DataFlux calls these SYMBOLS) – Symbols can contain spaces but must be surrounded by grave accent marks – -backwards quote on the upper-left side of the keyboard – if they do. Symbols can be declared anywhere except within programming constructs (eg. Loops and IF/THEN/ELSE).

Example: **`` `First Name` ``**

PUBLIC – is the default, are visible outside the code block in which they are defined.
PRIVATE – is only visible within the code block in which they are defined.
STATIC – the symbol's value is not reset between rows read in dfPower Architect.
VISIBLE – is the default. The symbol is output from the expression step.
HIDDEN – the symbol's value is not output from the expression step.

INTEGER – a number with no fractional part. 234 is an example.
STRING – by default it is 255 characters. Can be explicitly set "**`STRING(20) myStr`**" and can be set to bytes "**`STRING(20 BYTES) myStr`**".
REAL – number. 2.134 is an example of a real number.
BOOLEAN – "true" or "false". Useful for flags.
POINTER – is an available data type although it is not documented.

Example Declarations:

```
public string myStr                     // 255 characters by default
static  integer myInt
private string(20) myPrivStr            // explicitly set to 20 characters
string (20 bytes) `My 20 Byte String`  // string set to 20 bytes
hidden  boolean updataFlag
visible date `Todays Date`              /* C style comments work too */
```

Statements

DataFlux EEL statements can are separated by semi-colons or. This applies to declarations as well as programming statements.

Syntax Samples

// Assignment
```
`some symbol` = expression
```

// IF/THEN/ELSE
```
IF (x>y) THEN statement ELSE statement

IF (x==y) THEN
   BEGIN
      statement
      statement
   END
ELSE
   BEGIN
      statement
      statement
   END
```

// FOR Loop
```
FOR i = 1 TO 10 STEP 1
   statement
```

```
FOR i = 1 TO 10
   BEGIN
      statement
      statement
   END
```

// Return exits the block + returns value
```
RETURN expression
```

// Goto and Label
```
INTEGER x
x = 0
// continue is a label used for looping
continue:
   x = x + 1
   IF x < 10 goto continue
```

Arrays

Arrays of integers, strings, reals, dates and Booleans can be created. You cannot create arrays of objects: dbCursor, dbconnection, regex, file or any other type of object.

The dim() function is used to size or resize and array. It will also tell you how large an array is.

```
//declare an array
string array str_list

// size it so that it will hold 5 strings
Str_list.dim(5)

// resize it to hold 10 strings
Str_list.dim(10)

// determine  the array size
string array str_list // define array
integer array_size    // hold size
array_size = 0        // initialize to 0
str_list.dim(33)      // array size is 33


// determine array size
array_size = str_list.dim()
```

string_array, integer_array, data_array, Boolean_array and date_array are reserved words.

Set the value of an array item using the SET function.

```
Str_list.set(1, "EEL is awesome")
```

Get the value of an array item using the GET function.

```
String this_str
This_str = str_list.get(1)
```

FOR Loop

Has a start, end and step value. These values can be any expression.

The STEP value can be positive or negative. If it is not specified the default (1) is used.

If start value > end value you must specify a negative value for STEP.

```
// Example FOR loop code…
INTEGER i

FOR i = 1 to 10 STEP 1
   CALL PRINT("I = " & i)
```

```
FOR i = 100 to 1 STEP -1
   PRINT("i = " & i)
```

WHILE Loop

Loop while a condition is remains true.

```
INTEGER i
i = 10

WHILE i > 10
   BEGIN
      PRINT("Condition is met" )
      i = i + 1
   END

// Another example…
BOOLEAN continue_flag
continue_flag = "true"
INTEGER x
x = 0

WHILE continue_flag
   BEGIN
      x = x + 1
      IF x = 10 THEN
         continue_flag = "false"
   END
```

Expressions

An expression always results in a value. That value can be null.

Operators in order of precedence

Operators	Description
(,)	Parenthesis can be nested
* / %	Multiply Divide Modulo
+ -	Add Subtract
!= <> == > < >= <=	Equals – “!=” and “<>” are the same
and	Boolean and
or	Boolean or

```
String Expressions

string Str1
Str1 = "Hello"
String Str2
Str2 = "World"
String answerStr
answerStr = ""

// Concatenation
answerStr = str1 & " " & str2

// now answerStr contains
// "Hello World"
```

Integer and Real Expressions

There is automatic conversion between numeric values and strings.

```
real rNum
string myStr
myStr="3.14"

rNum = myStr // rNum = 3.14
```

Date Expressions

Dates are stored as a real number. The whole portion is the number of days since Jan 1, 1900. The fractional portion represents the fraction of a day.

Dates are denoted by **#**.

If you add a whole number to a date it adds that many days to the value.

```
date mydate

mydate = #10/06/09#   // October 6, 2009
mydate = #06 October 2009#
mydate = #Oct 06 2009 10:59:00#

mydate = mydate + 1 // now Oct 7
```

Boolean Expressions

A boolean expression can be: true or false. The results of comparisons are always either true of false.

Using AND or OR in an expression results in a boolean value.

EEL understands the following values: yes, no, true, false, y, n, t and f.

Numeric values of 0 are false. All others are true.

```
boolean t
boolean f
boolean answer

t = true
f = false

answer = t or f    // answer is true
answer = t and f   // answer is false
answer = 10<20     // answer is true
answer = 10==20    // answer is false
answer = "yes"     // answer is true
answer = "no"      // answer is false
```

Nulls

Null means "I don't know what the value is."
Empty string is a string that contains nothing.

```
null != empty string
```

Coercion (forcing an expression to a different data type)

You cannot convert an expression of type POINTER to any other data type.

You cannot convert an expression of type BOOLEAN to DATE. Also, you cannot convert a type of DATE to BOOLEAN.

Functions

Takes arguments (of specific data types) and returns a value (of a specific data type).

If you need to call a function but don't want the return value you can use the CALL statement. This is commonly done with the PRINT() function.

Some functions alter the values of the parameters that are passed to them (call by reference). Check the documentation for the function that you are calling.

Function calls can be nested inside one another.

```
string mytext
mytext = "lower case"

finalText = print(upper(mytext))
```

"LOWER CASE" is written to the Step Log.

Objects

An object combines data structures with executable code (functions) that is designed to operate on the data structure.

Blue Fusion Objects
Expressions and Functions

This object allows you to dynamically alter match definitions, read them from another object or setting different match definitions.

For more information go to the **DataFlux Expression Language Reference Guide** page 87.

Database Objects
Database Connectivity (DBConnect Object)

The EEL DBConnect object allows you to work with databases. You can use this object to connect to data sources, return a list of data sources and evaluate data that is being input from parent nodes.

For more information go to the **DataFlux Expression Language Reference Guide** page 80.

File Object
Text file reading and writing (File Object)

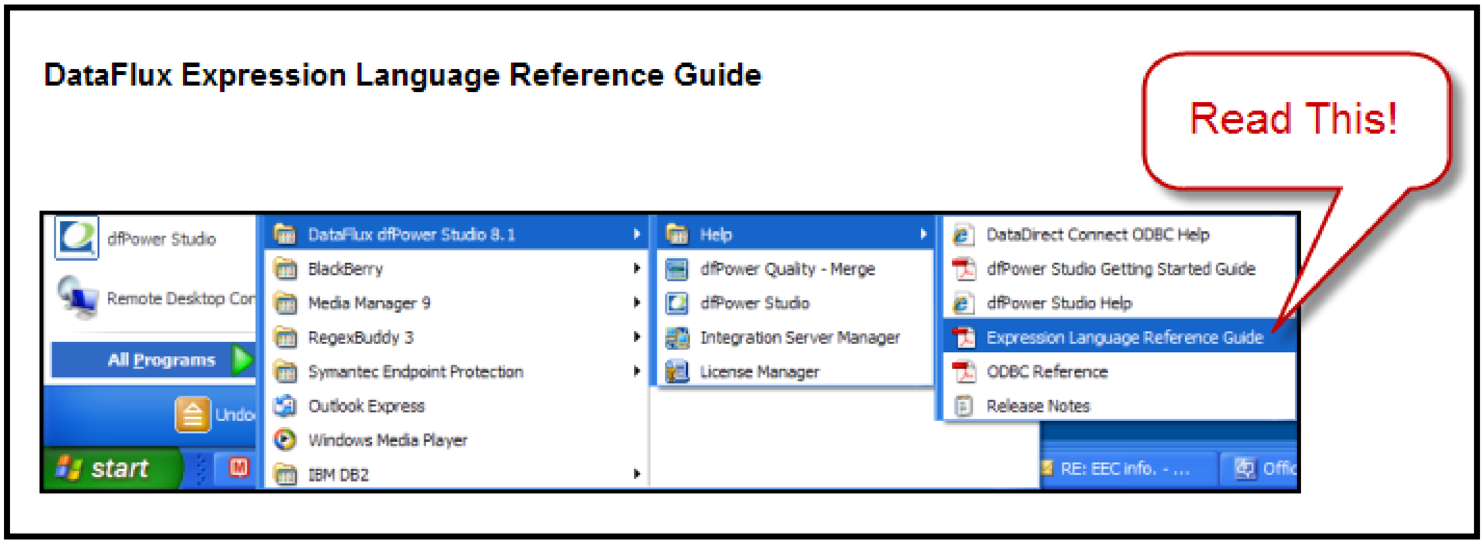
The file object can be used to read and write files, copy files, read information about the file (exists), execute programs and run batch files.

For more information go to the **DataFlux Expression Language Reference Guide** page 65.

Regular Expressions Object
Regular expression searches (Regex Object)

The regex object allows you to do regular expression searches of strings and text replacement operations.

For more information go to the **DataFlux Expression Language Reference Guide** page 84.



Note: many of the code samples and some of the text included in the Tip Sheet were directly copied from the **DataFlux Expression Language Reference Guide**.