SSIS ACCELERATOR WHITEPAPER SERIES

WORKING WITH SSIS EXPRESSIONS
Pragmatic Works BI Technical Article

Writer: Brian Knight
Technical Reviewer: Dustin Ryan

Published: 7/19/2009
Updated: 7/19/2009

Summary: This white paper covers how to use the SSIS expression language.
COPYRIGHT

The information contained in this document represents the current view of Pragmatic Works on the issues discussed as of the date of publication. Because Pragmatic Works must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Pragmatic Works, and Pragmatic Works cannot guarantee the accuracy of any information presented after the date of publication.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Pragmatic Works.
# TABLE OF CONTENTS

SSIS Accelerator Whitepaper Series ................................................................. 1
Working with SSIS Expressions ................................................................. 1
Introduction ........................................................................................................ 1
The Language ................................................................................................... 1
  Mathematical Expression Functions ............................................................... 1
  String Expression Functions ............................................................................ 2
  Date and Time Expression Functions .............................................................. 3
  Common Logic ................................................................................................. 4
Task Expressions ............................................................................................... 5
Precedence Constraint Expressions ................................................................. 6
Connection Manager Expressions .................................................................... 7
Variable Expressions ......................................................................................... 9
Data Flow Expressions ..................................................................................... 9
  Derived Columns Transforms .......................................................................... 10
  Conditional Split Transforms .......................................................................... 11
Enumerator Expressions ................................................................................... 13
  BI xPress and Expressions ............................................................................. 14
Conclusion ....................................................................................................... 15
INTRODUCTION

Whether you’ve been involved with SQL Server Integration Services (SSIS) for years or are just learning SSIS, I’m sure you’ve probably found the learning curve of the SSIS expression language quite challenging. The expression language is a core component of SSIS that helps you make any component in your SSIS packages dynamic and also helps your data flow make workflow decisions. This whitepaper walks you through a crash course in the SSIS expression language across all components of SSIS. After you complete this whitepaper, you’ll know how to use expressions in the control flow, data flow and in connections to make your package change at runtime and maximize the full power of SSIS.

THE LANGUAGE

The idea for this whitepaper came out of teaching countless SSIS classes where students gave us a very confused look when we showed expressions. Frankly, I can’t blame them for the look. The expression language feels like a secret society of confused developers ambushed C# in the back alley after a party, beat it senseless and out came the expression language after some reconstructive surgery. It has some constructs of C and feels like the expression language of Reporting Services.

Regardless of how we feel about the language itself, the expression language is quite powerful. The tiny language is responsible for making nearly any component in SSIS dynamic. Expressions tied to any object will cause the SSIS object to change every time that object is referenced. For example, you may have an expression on a connection that points to a variable called FileName that’s used in a loop. Every time the connection is referred by any object in the loop, the connection may point to a different file for its source. Because of this power, expressions are really the most important thing in SSIS to master since it will ultimately save you many headaches.

Without further ado, here’s the most boring table in this whitepaper, which shows a list of the common SSIS expression functions and a short description of them. Pin this up on your cube wall when you’re looking to take a nap.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Returns the absolute, positive value of a numeric expression. Example: ABS(-4.1) returns 4.1</td>
</tr>
<tr>
<td>CEILING</td>
<td>Returns the next largest whole number. Example: CEILING(123.01) returns 124</td>
</tr>
</tbody>
</table>
SSIS Accelerator Series: Working with SSIS Expressions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| FLOOR    | Returns the next smallest whole number.  
Example: FLOOR(32.8) returns 32 |
| POWER    | Returns the result of raising a numeric expression to a power.  
Example: POWER(3,2) returns 9 (3 to the power of 2) |
| ROUND    | Returns a numeric expression that is rounded to the specified length or precision.  
Example: ROUND(15.27881, 2) returns 15.28000 |
| SIGN     | Returns the positive (+), negative (-), or zero (0) sign of a numeric expression.  
Example: SIGN(-15.78) returns -1 |
| SQUARE   | Returns the square of a numeric expression.  
Example: SQUARE(4) returns 16 |
| SQRT     | Returns the square root of a numeric expression.  
Example: SQRT(16) returns 4 |

**STRING EXPRESSION FUNCTIONS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| FINDSTRING | Returns the one-based index of the specified occurrence of a character string within an expression.  
Example: FINDSTRING("Jacksonville, FL, FL, FL", ",", 1) returns the value of 13, meaning that the comma is on the 13th character (1 based) |
| LEN | Returns the number of characters in a character expression.  
Example: LEN("Brian") returns 5 |
| LOWER | Returns a character expression after converting uppercase characters to lowercase characters.  
Example: LOWER(MyStringValue) returns mystringvalue |
| LTRIM | Returns a character expression after removing leading spaces.  
Example: LTRIM(" Brian ") returns "Brian " (without quotes) |
| REPLACE | Returns a character expression after replacing a string within the expression with either a different string or an empty string.  
Example: REPLACE("2004-01-02", ",","") returns 20040102 |
| REPlicate | Returns a character expression, replicated a specified number of times. |
### SSIS Accelerator Series: Working with SSIS Expressions

#### Example:
```plaintext
REPLICATE("Brian",2) returns BrianBrian
```

#### REVERSE
Returns a character expression in reverse order.
```plaintext
Example: REVERSE("Brian") returns nairB
```

#### RIGHT
Returns a given number of characters to the right of the string.
```plaintext
Example: RIGHT("1992",2) returns 92
```

#### RTRIM
Returns a character expression after removing trailing spaces.
```plaintext
Example: RTRIM("   Brian   ") returns "Brian" (without quotes)
```

#### SUBSTRING
Returns a part of a character expression. In the below example, you can retrieve part of the zip code starting at the first character and going for 5 characters.
```plaintext
Example: SUBSTRING("32043-8421",1,5) returns 32043
```

#### TRIM
Returns a character expression after removing leading and trailing spaces.
```plaintext
Example: TRIM("   Brian   ") returns "Brian" (without quotes)
```

#### UPPER
Returns a character expression after converting lowercase characters to uppercase characters.
```plaintext
Example: UPPER("Brian") returns BRIAN
```

### DATE AND TIME EXPRESSION FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATEADD</td>
<td>Returns a new DT_DBTIMESTAMP value by adding a date or time interval to a specified date.</td>
</tr>
<tr>
<td></td>
<td>Example: DATEADD(&quot;Month&quot;, 2, GETDATE())</td>
</tr>
<tr>
<td></td>
<td>returns 2009-09-11 18:58:36.043000000</td>
</tr>
<tr>
<td>DATEDIFF</td>
<td>Returns the number of date and time boundaries crossed between two specified dates.</td>
</tr>
<tr>
<td></td>
<td>Example: DATEDIFF(&quot;dd&quot;, (DT_DBTIMESTAMP)&quot;7/15/2009&quot;, (DT_DBTIMESTAMP)&quot;7/19/2009&quot;) returns 4</td>
</tr>
<tr>
<td>DATEPART</td>
<td>Returns an integer representing a datepart of a date. Pass in values such as yy, month, or dd to return years, months or dates out of a full date.</td>
</tr>
<tr>
<td></td>
<td>Example: DATEPART(&quot;yy&quot;, GETDATE()) returns 2009</td>
</tr>
<tr>
<td>DAY</td>
<td>Returns an integer that represents the day of the specified date.</td>
</tr>
<tr>
<td></td>
<td>Example: DAY(GETDATE()) returns 11 if it were July 11th.</td>
</tr>
</tbody>
</table>
SSIS Accelerator Series: Working with SSIS Expressions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETDATE</td>
<td>Returns the current date of the system.</td>
</tr>
<tr>
<td>GETUTCDATE</td>
<td>Returns the current date of the system in UTC time (Universal Time Coordinate or Greenwich Mean Time).</td>
</tr>
<tr>
<td>MONTH</td>
<td>Returns an integer that represents the month of the specified date.</td>
</tr>
<tr>
<td>YEAR</td>
<td>Returns an integer that represents the year of the specified date.</td>
</tr>
</tbody>
</table>

**COMMON LOGIC**

Keep in mind that the above table is a partial list. There are also additional functions that handle casting to the various data types and resemble the logic of what is performed by the CONVERT T-SQL statement. These are important functions that will cast data types to nearly any other data type (there are some data types that are incompatible with each other). For example, if you want to convert today’s date to a string for later manipulation, you can use (DT_WSTR,length) like (DT_WSTR,30)(GETDATE()).

Commonly, you’re going to want to evaluate a value to make a decision. For example, if you want to read a variable, you have to use an @ sign as in @VariableName. This can be used in numerous examples throughout this whitepaper. Additionally, if you want to evaluate the value of the variable, you must use two equal signs as shown below:

@VariableName == "MonthEnd"

To determine if the variable is not equal, you would use the != combination. You can also evaluate multiple conditions by using && for a Logical AND or a || sign for a Logical OR as shown below:

@VariableName == "MonthEnd" && @ClientNuber == 405

You can also use a question mark and colon for conditional logic which is the equivalent of an IF..THEN..ELSE statement in many programming languages. The question mark represents the THEN statement and the colon represents the ELSE. The below example, sets the value of a column to the value of “Unknown” if the column is NULL. Otherwise, the present value is kept. This type of example would be used in a Derived Column Transform.

ISNULL(ColumnName) == True ? "Unknown" : ColumnName
**TASK EXPRESSIONS**

Nearly any property in a task can be made dynamic with expressions. For example, perhaps you want to set the value of a query to be dynamic in an Execute SQL Task or dynamically change the switches being based into an executable with the Execute Process Task. In most cases, task expressions can be set by going into the task’s configuration utility and going to the Expression tab of that task. Occasionally, tasks won’t have an Expression tab. In those cases, simply select the task in the Control Flow and click the ellipsis button next to the Expression property in the Properties pane.

In either case, the Property Expression Editor will open, where. You can select the property you wish to make dynamic. The below figure shows an Execute SQL Task. After you select the property you wish to make dynamic, click the ellipsis button next to that property to open the Expression Builder.

After the Expression Builder opens, you can drag variables or functions into the Expression pane as shown in the following screenshot. Notice in the screenshot that I had to cast the integer variable (MyVariable) to a 30 character unicode string with (DT_WSTR,30) so it could be concatenated into the string. As a general expression rule, data types of different types cannot be concatenated without being casted first. An example like the below could also be done by passing variables in with a question mark in the query itself. But, sometimes the provider won’t support a question mark.
If this was a string variable, then this type casting would not have been necessary but I would have had to add single quotes around the value like this:

```
"SELECT Count(*) as MyCount from MyTable where ColumnName = '" + @MyStringVariable + "'
```

Keep in mind that each time this task runs, the SQL Statement property would now be tied to the expression and evaluated each time it’s executed. So, if the task was inside a loop container and something was changing the MyVariable value in the loop, then each time the task was run it would return a different SELECT statement.

---

**PRECEDENCE CONSTRAINT EXPRESSIONS**

Precedence constraints control the workflow of the package, allowing tasks to execute if the predecessor was successful, completed or failed. In addition or instead of the success, failure or completion precedence constraints, you can also add expressions on
the constraint. This is useful if you want to only execute the next task under a given circumstance like during a month end processing period.

To create a precedence constraint expression, double-click on the precedence constraint in the Control Flow tab and change the Evaluation Operation drop down box to “Expression and Constraint”. Then, type the expression in the Expression box once it is available. For example, the below figure shows how to turn on the next set of tasks or containers only if the RowCount variable is greater than 0 and the preceding task was successful. This is useful if the RowCount variable was populated from a data flow’s Row Count Transform.

![Precedence Constraint Editor](image)

**CONNECTION MANAGER EXPRESSIONS**

One of the most vital ways you can use expressions is to make connections dynamic. Picture a package that looks like the following screenshot. In the screenshot, you can see a ForEach Loop Container which loops over a series of files. The tasks inside the loop loads the files and then archives them after an audit. The ForEach Loop Container captures the file name with the path and stores it into a variable. Once we know the file name, the PetitionData connection manager must be changed to point to the new file.
SSIS Accelerator Series: Working with SSIS Expressions

Then, every time we refer to the connection manager from any task, it will be changed to point to the new file.

To set a connection on an expression, select the connection manager and then in the Properties Window to the right, click the ellipsis button next to Expressions. For Flat File or File connection managers, you’ll want to set the ConnectionString property. For databases connections, set the ServerName (instance name) and the InitialCatalog (database name) properties inside the Property Expression Editor. You can also set the ConnectionString property for a database but it will require that you build the connection string, which may be more complex.
VARIABLE EXPRESSIONS

Variables can also be made dynamic by using expressions. This can be used to set a default value for a variable. For example, if you have a variable called FileName that holds the file name to be processed like C:\Projects\MyFile.csv. You could create a new variable called ArchiveFileName and that variable can be derived from the FileName variable, ultimately pointing to the file to C:\Projects\Archive\MyFile.csv.

To do this select the ArchiveFileName variable in the Variable Window and the in the Properties Window, change EvaluateAsExpression to True. Right below this, you can select the ellipsis button for the Expression property to set the variable. In our earlier example, you can replace the root path with the archive folder path with the below code. Note that the double-backslashes is required to use a single backslash and avoids an escape character like \n for a new line.

REPLACE(@FileName, "C:\Projects", "C:\Projects\Archive")

DATA FLOW EXPRESSIONS

Data flow expressions can be used to make data decisions in the Data Flow Task. At its highest level, you can use expressions on the Data Flow Task to control some of the common sources or tie transforms and destinations and make them tied to a variable. You would have to do it at the task-level because Microsoft did not add much dynamic control in the data flow components themselves. At its more granular level, expressions are used in common transforms like the Derived Column and Conditional Split Transform to change the data entering into the tasks dynamically or control the flow of data.

One common scenario in the Data Flow Task is to make an ADO.NET Source query (Data Reader Source in SQL Server 2005) dynamic. In the OLE DB source, you can tie a query to a variable or pass in variables with question marks. In an ADO.NET source though, there’s no way to parameterize the query. To get around this, you can select the Data Flow Task that holds the ADO.NET source and in the Expressions area of the Properties Window set the [ADO NET Source].[SqlCommand] property to your expression. The name [ADO NET Source] will in actuality be the name of your ADO.NET source. The downfall of this method is you cannot rename your source once you tie the expression to it since the expression is tied to the name of the source.

<table>
<thead>
<tr>
<th>Misc</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ADO NET Source].[SqlCommand]</td>
</tr>
<tr>
<td>[ADO NET Source].[TableOrViewName]</td>
</tr>
<tr>
<td>[Find City and State].[SqlCommanc] select * from [dbo].[ZipCodeLoc]</td>
</tr>
<tr>
<td>[Find City and State].[SqlCommanc] select * from (select * from [dbo] [Find NULL City and State].[Bad Dal] LTRIM(City) == &quot;&quot;</td>
</tr>
<tr>
<td>[Standardize Zip Code].[Derived Co SUBSTRING(ZipCode,1,5)]</td>
</tr>
</tbody>
</table>
DERIVED COLUMNS TRANSFORMS

The Derived Column Transform is the single most important transform you can learn in the data flow and it acts much like your SSIS Swiss Army Knife. There’s more power in the transform than most of the other tasks combined. This transform allows you to apply many business rules against your data while the data is in the SSIS memory buffer. For example, you can do basic IF..THEN..ELSE clauses, upper casing of data, or use a Derived Column Transform to put your data in bands.

Once you connect the Derived Column Transform to the source or another transform, double-click on it to open the editor. You can then drag expression functions from the top right pane into the Expression line below. Then, for the Derived Column property, select the column name you’d like to replace with this function or if you’d like to add a new column from the results of the expression to be created. Lastly, if you’re creating a new column, type the column name in the Derived Column Name property (only required for new columns). If this is a new column, you can also set the data type and length. For an existing column, you cannot change the metadata in flight.

For example, in the below screenshot, you can see that the old ZipCode column is being replaced with the result of the SUBSTRING(ZipCode,1,5) expression. This expression will take the first 5 digits of the ZipCode string column and replace the old value with this value. You could add new business rules for new columns under this ZipCode line but you can only replace a given column once in a single transform. If you need to apply several rules to the same column, you’ll need several Derived Column Transforms.
If you wish to do an IF...THEN..ELSE statement you could use code like this the question mark (?) to represent the THEN statement and a colon (:) to represent the ELSE statement. For example, if you would like to replace a column that had NULL values with the word “Unknown”, you could use the following statement:

\[
\text{ISNULL(ColumnName)} == \text{True} \ ? \ "\text{Unknown}" : \text{ColumnName}
\]

In that statement, it would replace the value with “Unknown” and the colon represents the ELSE statement, saying keep the existing value. If you’d like convert a column to NULL if it hold a blank value, code like the below will work:

\[
\text{TRIM(ColumnName)} == "" \ ? \ (\text{DT_STR},4,1252)\text{NULL(DT_STR},4,1252) : \text{ColumnName}
\]

You may be a bit baffled by the above code. I too couldn’t believe I had to structure it this way. As you can see in the above code, you must cast a NULL value to a string twice to make this work. The (DT_STR,4,1252) cast the value into a four character string that has a code page of 1252 (Latin case insensitive).

**CONDITIONAL SPLIT TRANSFORMS**

The Conditional Split Transform moves data down different paths in the data flow based on a given condition. It can be used to throw data away if a condition is met or not met. The below screenshot shows an example of a Lookup Transform that tries to find a matched record based on a primary key in the target table. If it can find the record, the Conditional Split Transform will send the row down the Update path from the conditional split. If the row can’t be matched, it’s directed to the New Record path.
To do this, you can open the Conditional Split Transform (shown in the below screenshot) and add three conditions. One condition handles new records when there is no match with a NULL column. The next condition is for records that are update candidates when one column is different than another column. The last condition at the bottom is if neither of these conditions evaluate to true. Out of the Conditional Split transform, you’ll now see three possible lines for each of the conditions. If you don’t use one of the green lines, the row will be thrown away.
ENUMERATOR EXPRESSIONS

Enumerator expressions can be used to make enumerators in the ForEach Loop Container dynamic. The most common example of this is to make the directory name that you loop over dynamic so you don’t hard-code paths into your container. To set an expression here, go to the Collection page as is shown in the following screenshot, set the Enumerator option to whatever option you wish. Expand the Expressions section and click the ellipsis button. You will then be taken through the same steps that I’ve spoken about throughout this whitepaper.
BI xPress and Expressions

Pragmatic Works (http://www.pragmaticworks.com) specializes in the creation of software in order to make easier the lives of developers who write BI solutions using the Microsoft stack. Pragmatic Works has developed a product called BI xPress. BI xPress utilizes an enhanced expression editor that features built-in expression snippets, among dozens of other features. These expression snippets are pieces of reusable code that act as templates for your code. BI xPress also has an enhanced editor with more advanced parsing to get you past the SSIS learning curve. With BI xPress, SSIS components are color coded when associated with an expression. You can download a free evaluation of BI xPress at http://www.pragmaticworks.com.
CONCLUSION
This whitepaper discussed how to use the expression language to make various components in SSIS dynamic. You can use expressions to make nearly any property in the control flow, connections or variables dynamic. You can also use expressions in the data flow from transforms like the Conditional Split Transform or Derived Column Transform.

For more information:
http://www.pragmaticworks.com
Brian, Devin and Mike's new book: http://tinyurl.com/mcypmv