SSIS for the hurried developer

Bill Fellows

August 20, 2016

Contents

Part I

How to use this book

Chapter 1

Getting started

This book is targeted to the novice SSIS developer. You aren't looking to spend months working on the great SSIS project that will have white papers written about it - you just want to get package finished so you can go on to the next fire.

1.1 Cookbook approach

TODO: determine which I like better

Is this a circle? (show square)? No? Well, is this a circle (show square overlaid with diamond)? Still no? Is this a circle yet (show square rotated 8 times)? We can at least agree this is a circle. The trick to development then, is to be able to decompose problems you canâĂŹt solve into a problem you can solve. The goal of this book is provide you with enough âĂIJsquaresâĂİ to tackle your SSIS problems.

When you look at a house, whether itâĂŹs a one-room shack or a palatial mansion, the same hammer, nails and 2x4's were used to create it-one simply has applied the pattern many more times over. ItâĂŹs the intention that after working through this, youâĂŹll be able to break your problem down into components and apply them to your project.

It should not be necessary to read this book from cover to cover. Instead, I attempt to put all the basic tools onto your toolbelt and then in the actual recipe section, we make liberal use of them.

1.2 Biml

The Business Intelligence Markup Language, Biml, describes the platform for business intelligence. Here, we're going to use it to describe the constructs we're going to create. I find that once you have built a few packages by clicking and dragging, and clicking through more and more panes and panel and context sensitive menus, SSIS development can really take a toll on your mouse and ability to get work done.

The solution, is Biml. It is a free add on for Visual Studio/BIDS/SSDT that is rolled into the BIDS Helper project on CodePlex.com. If you're doing SSIS, SSRS or SSAS development, BIDS Helper adds the 10% of needed functionality that didn't make it into the shipped product but really would have helped you, the developer.

TODO: include product plug for mist? If you think you love Biml via Visual Vtudio, then the functionality that is only available through Mist will blow your mind.

If you're in a tightly controlled environment, you can still use BIDS Helper. There is no-install version which should work with the most restrictive group policies.

All the solutions in this book will contain the Biml required to generate the same object. This allows you to reproduce the *exact* same SSIS package without the hassle of dealing with the nuances of each task or component in the designer.

The following is the Hello World equivalent in Biml. The engine translates that XML into an SSIS package that is identical to one created by selecting "New SSIS Package"

```
<Biml xmlns="http://schemas.varigence.com/biml.xsd">
    <Packages>
        <Package ConstraintMode="Linear" Name="HelloWorld">
        </Package>
    </Packages>
</Biml>
```

If you're unfamiliar with tag based languages, like XML or HTML, you define a tag with angle brackets and close it with the same tag but with a forward slash, thus <Entity>. . . </ Entity >If there is nothing between the tags, you can simply specify it as <Entity />

Part II

Expressions

Expressions are the difference between a hard coded package and one that gracefully responds to change. Most every object in the SSIS world supports the concept of expressions. An expression is just that, it is code that, when evaluated, expresses a new value. It can be important to note that expressions are evaluated each time they are used. This might be a good behaviour if you want to capture the exact moment a row is inserted into a table. It might not be desirable if your output file name changes every second.

Chapter 2

Date and time

We often need to generate a date string in various formats. A common challenge encountered is getting that format just right. Unless you have reasons to the contrary, using a format of year, month and date, all numeric, results in a sortable date which makes it easier to find specific files in large collections.

Getting a date There are different mechanisms for acquiring a date. A popular approach is to use `GETDATE()` GETDATE(SSIS Expression)

It's built into the SSIS Expression language but I find that the System scoped variable, `StartTime` provides a better reference point as it is evaluated only when the package begins. This is in contrast to GETDATE which is evaluated each time it is inspected. For a long running package, the drift in values may not be acceptable. It might also be *exactly* what is desired. Assume we have a Derived Column transformation in a package which adds a column named InsertDate into our buffer.

Consider the following two queries, which would more correctly describe the data loaded from today's load?

Do you look for all the rows loaded at a specific point in time?

```
SELECT
    T.*
FROM
    MyTable AS T
WHERE
    T.InsertDate = '2015-01-17T12:24:48.123';
```

Or do you look for all the rows loaded between an interval?

```
SELECT
    T.*
```

```
FROM
    MyTable AS T
WHERE
    T.InsertDate >= '2015-01-17T12:24:48.123'
    AND T.InsertDate < '2015-01-17T15:56:02.673';
```

The "right" answer is neither—we should instead track data loads through an alternate means that supports tracking of concurrent loads.

2.1 YYYY

This expression defines how to get a 4 digit year. This is sometimes referred to as CCYY (century, century, year, year). The expression simply applies the `YEAR()` function to a variable which returns an integer value.

```
(DT_WSTR, 4) YEAR(@[System::StartTime])
```

To make use of it, it's been my experience that we will be using this value for string concatenation instead of mathematical operations: addition, subtraction, etc. Therefore, we explicitly cast the result of the `YEAR()` as a unicode string, `DT_WSTR` of length 4

The following snipped expresses the Biml required to generate an SSIS variable which uses the above expression.

```
<Variable
    Name="YYYY"
    DataType="String"
    Namespace="Cookbook"
    EvaluateAsExpression="true">
    (DT_WSTR, 4) YEAR(@[System::StartTime])
    </Variable>
```

2.2 YY

This expression defines how to get a 2 digit year. We apply the `RIGHT()` function and get the last 2 digits. There is no need to explicitly cast to a string as it is handled by the function call.

```
RIGHT(YEAR(@[System::StartTime]), 2)
```

Biml

```
<Variable
    Name="YY"
    DataType="String"
    Namespace="Cookbook"
    EvaluateAsExpression="true">RIGHT(YEAR(@[System::StartTime]),2)</Variable>
```

2.3 MM

This expression defines how to get a 2 digit month such that March is 03 instead of 3. The leading zero is important for character based sorting. We will use the `MONTH()` function which returns an integer from 1 to 12. We convert that to a string, prepend a zero to it and then shear off the last two characters using the `RIGHT()` function. The trimming is only required for October, November, and December but the logic is cleaner to unconditionally apply `RIGHT()`.

```
RIGHT("0" + (DT_WSTR, 2) MONTH(@[System::StartTime]), 2)
```

Biml

```
<Variable
    Name="MM"
    DataType="String"
    Namespace="Cookbook"
    EvaluateAsExpression="true">RIGHT("0" + (DT_WSTR, 2) MONTH(@[System::StartTime])
```

2.4 DD

This expression defines how to get a 2 digit day such that instead of 9 we get 09. It is the same process used in MM except with a call to `DAY()`. We continue to prepend a zero to our value and shear off the last two digits.

```
RIGHT("0" + (DT_WSTR, 2) DAY(@[System::StartTime]), 2)
```

Biml

```
<Variable
    Name="DD"
    DataType="String"
    Namespace="Cookbook"
    EvaluateAsExpression="true">RIGHT("0" + (DT_WSTR, 2) DAY(@[System::StartTime]),
```

2.5 YYYY-MM-DD

The most common format is a year, month and day separated by a dash. This recipe uses what we've built in sections 2.1, 2.3 and 2.4.

verbatimimport

```
@[Cookbook::YYYY] + "-" + @[Cookbook::MM] + "-" + @[Cookbook::DD]
```

Biml

```
<Variable
    Name="YYYYMMDD"
    DataType="String"
    Namespace="Cookbook"
    EvaluateAsExpression="true">@[Cookbook::YYYY] + "-" + @[Cookbook::MM] + "
```

2.6 YYYYMMDD

A more compact version of the year, month, day string is represented below.

```
@[Cookbook::YYYY] + @[Cookbook::MM] + @[Cookbook::DD]
```

Biml

```
<Variable
    Name="YYYYMMDD"
    DataType="String"
    Namespace="Cookbook"
    EvaluateAsExpression="true">@[Cookbook::YYYY] + @[Cookbook::MM] + @[Cookb
```

Chapter 3

File names

File name operations are generally better served by using the .NET System.IO library as it offers robust error handling, file system translations, etc. For general use however, an expression can get the job done.

The following expressions assume we have a variable named @[User::CurrentFileName] define and populated with C:\SSISData\Client\Input\DailySales.csv

3.1 Get file extension

To get the file extension, we'll three functions: `RIGHT()`, `FINDSTRING()`, and `REVERSE()`. Working from the innermost parenthesis, we'll reverse the value for our string. We do this so that the first period we find *should* be the position where the file extension begins. This clearly doesn't work for cases where the file has no extension or where the extension is not the last portion of the filename *e.g. MyFile.txt.2015-02-01*.

Knowing the position where the file extension begins, we then pass that starting point for the call to `RIGHT()`.

```
RIGHT(@[User::CurrentFileName],
    FINDSTRING(REVERSE(@[User::CurrentFileName]), ".", 1))
```

This expression will return `.csv`. If you want only `csv`, then you will need to subtract one from the position passed to the `RIGHT()` call. Thus, the final three characters in the above expression `1))` become `1) -1)`.

3.2 Get file name

To get the file name without the path, `DailySales.csv` we'll apply the same concept as in section 3.1. However, instead of searching for *, this time we'll look for the \.

```
RIGHT(@[User::CurrentFileName],
    FINDSTRING(REVERSE(@[User::CurrentFileName]), "\\", 1) -1)
```

The double backslash, \\, in the above formula is not a typo. The backslash character is used to escape control codes in strings and in this case, it escapes itself.

3.3 Get file path

To get the path `C:\SSISData\Client\Input\`, we need to keep everything to the left of the final path separator, the \. The 2005 release of SSIS, did not include a `LEFT()` expression. We had `RIGHT()` and we had `SUBSTRING()`. It was argued that `LEFT` was really just a case of SUBSTRING with a starting position of 1, but the convenience and inconsistent behaviour when compared to other languages lead to the introduction of `LEFT` with 2008

TODO: verify release introduction

Our expression becomes

```
SUBSTRING(@[User::CurrentFileName],
    1,
    FINDSTRING(REVERSE(@[User::CurrentFileName]), "\\", 1))
```

The 2008 and forward releases of SSIS allow for the use of the slightly more compact expression

```
LEFT(@[User::CurrentFileName],
    FINDSTRING(REVERSE(@[User::CurrentFileName]), "\\", 1))
```

The similar caveat exists here as in 3.1 – this generates the file path with the trailing backslash intact, *i.e.* C:\SSISData\Client\Input\. If you need C:\SSISData\Client\Input, then you will need to subtract one from the final argument to SUBSTRING/LEFT *i.e.* `)-1)`

3.4 Get file name without extension

There are different recipes you can use to get the base file name, `DailySales`, but they generally take the form of eiter perform substring operations to trim the path and extension or you can use the `REPLACE()` function to strip out the known elements.

Chapter 4

Padding strings

A pad operation fills out a string with a given pattern. The reverse of padding a string would be trimming.

4.1 Left pad

In the U.S., a postal code is a five digit string. Tools like Excel interpret that data as numeric and thus 00532 becomes 532. Calling a left pad function, we could get our leading zeros back.

4.2 Right pad

A right pad operation may be needed if you were providing text input for printing a check. Assume the issued amount is \$100.30. The trailing zero is not needed from a mathematics perspective so the numeric value written would be \$100.3 That's going to look odd.

4.3 Trim to null

This expression only works in the context of a Data Flow. An SSIS Variable cannot be set to NULL. If you're dealing with fixed width files, that file transport mechanism does not have the ability to express that something lacks a value. There is no NULL in that column, it will simply be the space character. A Derived Column can be used with the following expression to ensure we make an string filled with whitespace is turned into a NULL for storage.

Part III

Looping

Chapter 5

Foreach Enumerator

5.1 Process all the files

Foreach File Enumerator

5.2 Shred a recordset

Foreach recordset

5.3 Iterate a resultset

Foreach ado.net rs

Part IV

Email

Chapter 6

Send mail task

6.1 Send an email

6.2 Send an attachment

6.3 Send the results of a query

6.4 Send an HTML formatted message

6.5 Send dynamic text

Chapter 7

sp send dbmail

7.1 Send an email

7.2 Send an attachment

7.3 Send the results of a query

7.4 Send an HTML formatted message

7.5 Send dynamic text

Chapter 8

Dislcaimers

The code represented in this book is good but not guaranteed to work. Any failure resulting from this code is yours, author not liable for damages resulting from outages, etc.

TODO: consult a lawyer for this