Microsoft® BizTalk® Server 2010

BizTalk Server – How Maps Work

Maps or transformations are one of the most common components in the integration processes. They act as essential translators in the decoupling between the different systems to connect. This article aims to explain how maps are processed internally by the engine of the product as we explore the map editor BizTalk Server.

Sandro Pereira

September 2012
Version 1.0

Content

Introduction

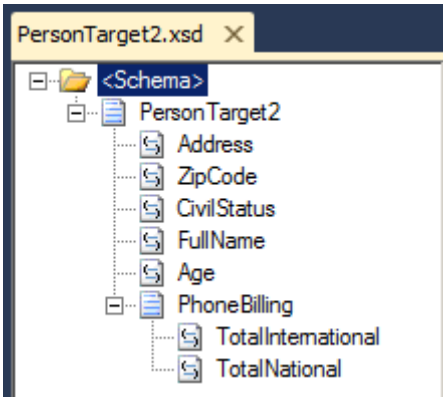
Maps or transformations are one of the most common components in the integration processes. They act as essential translators in the decoupling between the different systems to connect. This article aims to explain how maps are processed internally by the engine of the product as we explore the map editor BizTalk Server.

This series of post are based on the series "BizTalk Server – Basics principles of Maps" published earlier where it's explained in detail the basic functionalities of maps and how they can be implemented. This article intends to be an introductory note for whom is taking the first steps in this technology.

As explained in the previous article, when we perform a transformation in the message there are 5 basic functionalities that typically arise:

- Simple mapping of a given value (direct copy)
- Concatenation of values
- Conditional selection
- Custom scripts
- Add new values (data)

Based on these functionalities we will explain how the BizTalk Mapper Designer processes and translates these links internally. To better understand its functioning, we perform some changes to the structure of the destination schema: we add an optional element ("CivilStatus") and intentionally disorganized its structure.



Processing model of Maps

Although traditionally the information is extracted from the source as it is being processed, in reality in the models based on XSLT what happens is exactly the opposite: Each element in the target (destination schema) causes the search for the corresponding in the source. Here's a traditional example:

- The source is traveled (parsed) from beginning to end of file;
- The information is extracted from the source in the exact order that is found;
- The mapping rules are constructed as the source is traveled (parsed).

BizTalk also uses this technique in the conversions of Flat Files to XML format (Syntax Transformations, also explained in BizTalk Server – Basics principles of Maps), however the transformations inside maps use a different approach, as we will see later in this article.

One of the important factors when using integration tools is also having in mind the existing technology standards, and that's what the creators of the product made. BizTalk works internally, almost exclusively, with XML documents, so it made sense to use a standard technology to carry out this type of transformations. For that W3C defined the XSLT (Extensible Stylesheet Language Transformation) as the standard format to represent the transformations between XML documents

This way all links and functoids that are graphically visible in the Mapper Grid are not more than a graphical representation of an XSLT document that transforms the source document in a particular format specified by the destination schema, i.e., BizTalk maps are graphical representations of XSLT (*Extensible Stylesheet Language Transformation*) documents that allow us to perform, in a simple and visual manner, transformations between XML messages.

We can say that BizTalk maps always have its focus in the final document, thereby making sense that the mapping rules to be processed in the sequence required to create the final document. When the map is compiled, these rules are translated into XPath queries and XSLT functions in order to transform the required information, i.e., the mapping rules are built from the destination (target) and not the source as some traditional tools (or traditional models).

BizTalk maps follows the model below:

- The BizTalk mapping engine traverses the destination schema from beginning to end;
- The mapping rules are constructed and executed as links are encountered in the destination schema;
- The information is extracted from the source when a link is encountered in the destination schema.

Note: We can use an XSLT created by an external application and include it on the map inside a Scripting functoid (XSLT custom script) or by importing a XSLT file that carrying out the complete transformation of the map (of course the editor (BizTalk Mapper Designer) does not represent the graphical representations (links and functoid) in this scenario)

Deconstructing a map

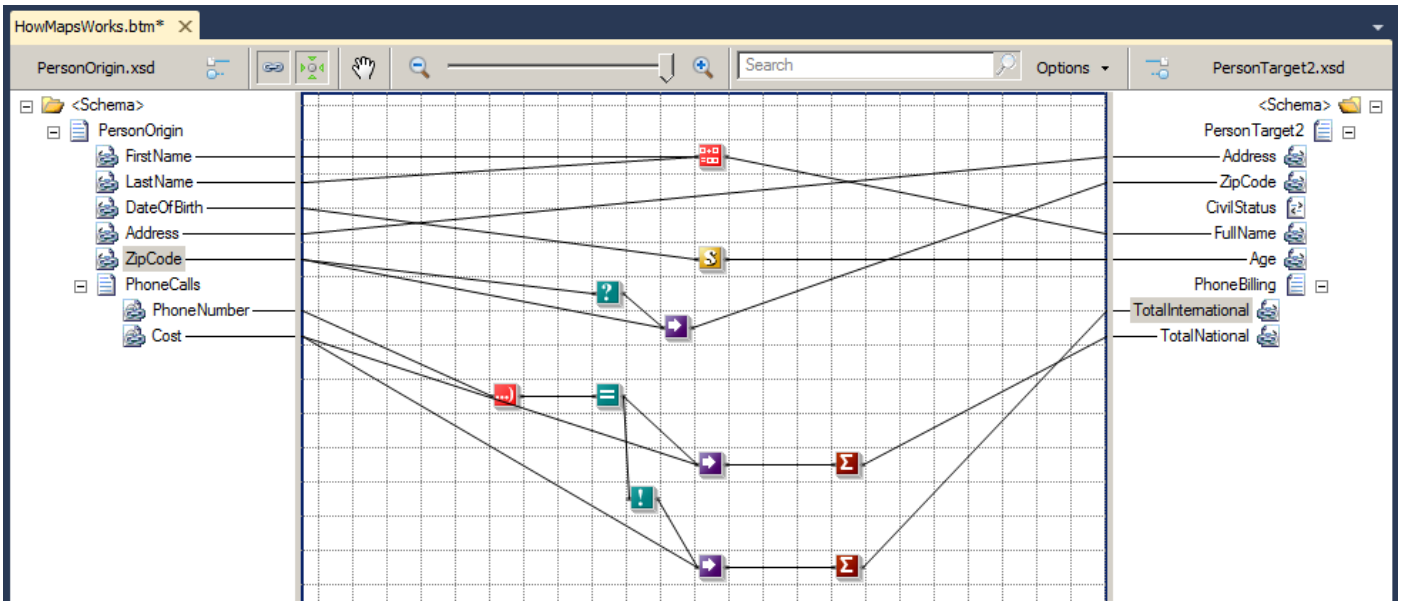
In this article we will use the basic mapping operations previously described, and analyze the decisions taken by the BizTalk mapping engine.

Basically on this mapping problem there are two similar schemes, which we intend to map the source elements in their proper destination and for which we implemented the following challenges:

- Concatenate the first and last name in order to give the full name (Concatenation of values);
- Map the Address in its destination element (Direct copy)
- Transform the date of birth in age (Custom scripts);
- The Zip Code should only be mapped if it has a valid string, otherwise it will not be mapped (Conditional selection);
- The element Civil Status is optional and as we have no evidence in the source to map it, it should be ignored (Add new values).
- Additionally, we perform a more advanced mapping logic to calculate the total of national and international calls using cycles, conditional selections and mathematical operations.

As you can see, we intentionally switched the order of elements in the destination schema in order to, more easily, understand and verify how BizTalk maps works.

This way we obtained the following final map:



Note: The order in which we perform the links between the elements from source to destination schema, in this scenario, is irrelevant to the compiler, however the same cannot be said for functoids. The functoids require certain input parameters that can vary according to the functoid that we are using, in this case the order with which we associate the link is extremely important

The previous image of the map is actually the following representation of this XSLT document:

- https://gist.github.com/2508405

Based on this document we will follow the behavior of the compiler. What he performs is to translate the links on the map, as he finds them, while he traverses the destination schema from beginning to end:

- The first element found is "Address"; Since there is link associated to this element, the link is translated by one XPath expression (”Address/text()”) that defines the element to be extracted from the source:

```
<Address>
   <xsl:value-of select="Address/text()" />
</Address>
```

- The second element found is “ZipCode” also with a link associated; It is a conditional selection that will be translated into a XSLT condition (xsl:if):

```
<xsl:variable name="var:v1" select="userCSharp:LogicalExistence(boolean(ZipCode))" />
<xsl:if test="string($var:v1)='true'">
  <xsl:variable name="var:v2" select="ZipCode/text()" />
  <ZipCode>
    <xsl:value-of select="$var:v2" />
  </ZipCode>
</xsl:if>
```

- The third element is "CivilStatus", since there are no links associated with this element, it is ignored in the mapping process.
- The fourth element to be processed is “FullName” and once again there is a link associated, which corresponds to the concatenation of two elements of the origin; If you check the generated code, is assigned to the element "FullName" the value of variable “$var:v3” that is generated from the execution of the function *userCSharp:StringConcat* that visually represents the ***String Concatenate Functoid:***

```
<xsl:variable name="var:v3" select="userCSharp:StringConcat(string(LastName/text()) , &quot;,
&quot; , string(FirstName/text()))" />
<FullName>
 <xsl:value-of select="$var:v3" />
</FullName>
```

- The fifth element is “Age”, a link is found which means that custom script found inside the ***CSharp Inline Scripting Functoid*** will be carried out:

```
<xsl:variable name="var:v4" select="userCSharp:CalculateMyAge(string(DateOfBirth/text()))" />
<Age>
 <xsl:value-of select="$var:v4" />
</Age>
```

- Finally we found the record “PhoneBilling” that contain two elements: “TotalInternational” and “TotalNational” both with links associated.
  - Note that although we have not defined any cycle through loop functoid, the compiler is smart enough to realize that we are dealing with a cycle and translate it correctly.
  - However, unfortunately, it will create an own cycle for each element. For a better optimization, it will require us to use a custom script.
  - Both elements are calculated using conditions and mathematical calculations as you will see in the generated code

```
<PhoneBilling>
        <xsl:variable name="var:v5" select="userCSharp:InitCumulativeSum(0)" />
        <xsl:for-each select="/s0:PersonOrigin/PhoneCalls">
          <xsl:variable name="var:v6" select="userCSharp:StringLeft(string(@PhoneNumber) ,
&quot;4&quot;)" />
          <xsl:variable name="var:v7" select="userCSharp:LogicalEq(string($var:v6) ,
&quot;+351&quot;)" />
          <xsl:variable name="var:v8" select="userCSharp:LogicalNot(string($var:v7))" />
          <xsl:if test="string($var:v8)='true'">
            <xsl:variable name="var:v9" select="@Cost" />
            <xsl:variable name="var:v10"
select="userCSharp:AddToCumulativeSum(0,string($var:v9),&quot;1000&quot;)" />
          </xsl:if>
        </xsl:for-each>
        <xsl:variable name="var:v11" select="userCSharp:GetCumulativeSum(0)" />
        <TotalInternational>
          <xsl:value-of select="$var:v11" />
        </TotalInternational>
        <xsl:variable name="var:v12" select="userCSharp:InitCumulativeSum(1)" />
        <xsl:for-each select="/s0:PersonOrigin/PhoneCalls">
          <xsl:variable name="var:v13" select="string(@PhoneNumber)" />
          <xsl:variable name="var:v14" select="userCSharp:StringLeft($var:v13 , &quot;4&quot;)" />
          <xsl:variable name="var:v15" select="userCSharp:LogicalEq(string($var:v14) ,
&quot;+351&quot;)" />
          <xsl:if test="string($var:v15)='true'">
            <xsl:variable name="var:v16" select="@Cost" />
            <xsl:variable name="var:v17"
select="userCSharp:AddToCumulativeSum(1,string($var:v16),&quot;1000&quot;)" />
```

```
      </xsl:if>
    </xsl:for-each>
    <xsl:variable name="var:v18" select="userCSharp:GetCumulativeSum(1)" />
    <TotalNational>
      <xsl:value-of select="$var:v18" />
    </TotalNational>
  </PhoneBilling>
```

The sequence of links

"The order in which we perform the links between the elements from source to destination has a huge impact in the final result". This statement is true and false at the same time!

In fact the order with which we associate links (*Drag&Drop*) from the source to different destination elements is irrelevant, since the compiler, as previously explained, will process them in the correct order… Except if we have to associate several links to the same destination element or functoid. In these two last cases the order in which the link association takes place is extremely important and can lead to unexpected results.

Impact of the order of links in functoids

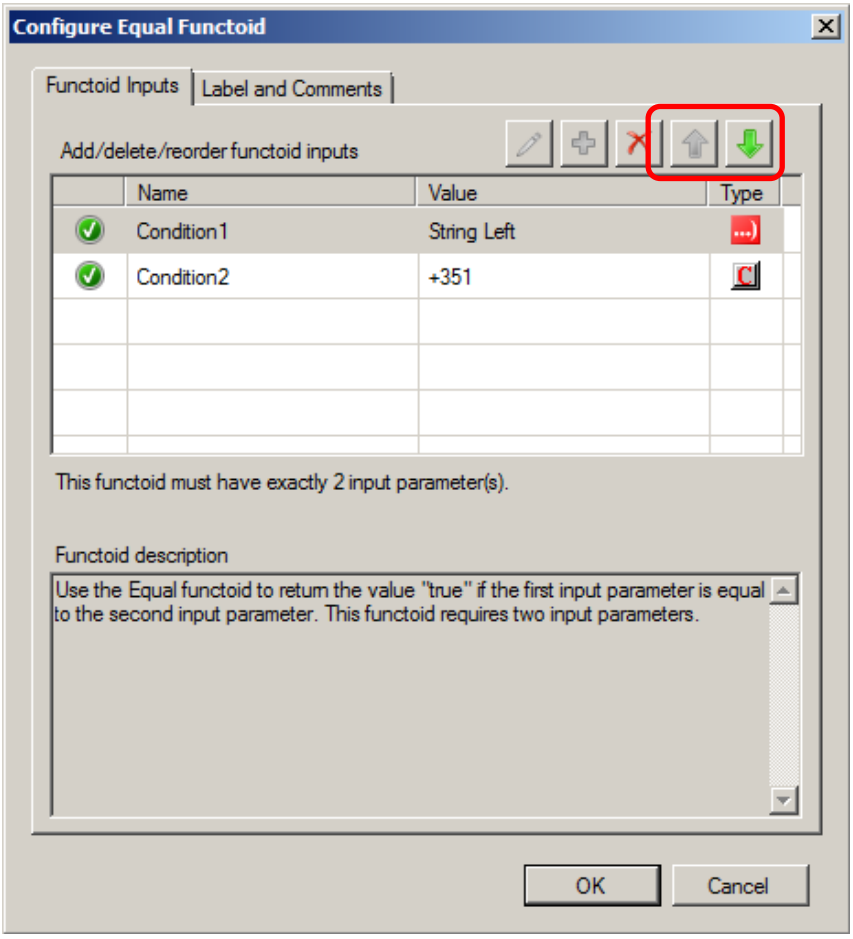
The functoids require certain input parameters that can vary according to the functoid that we are using, in this case the order with which we associate the link is extremely important, a practical example is the *Value Mapping Functoid*.

This functoid returns the value of the second parameter if the value of the first parameter is "true". If the value of the first parameter is not "true", the corresponding element or attribute in the output instance message is not created. Therefore, it is necessary to respect the order in which we associate the links:

- The first parameter must be a value "true" or "false", generally from the output of some other **Logical** functoid or from a variable Boolean field in the input instance message.
- The second is the value that is output if parameter 1 is "true". This value can be from a link from a node in the source schema that represents simple content, the output from another functoid, or a constant input parameter.

If we change the order in which the links are associated to the functoid, it will lead to mapping errors or unexpected results, according to the functoid used.

The link reorganization in functoids is very easy to accomplish, you just open the functoid detail (double click) and use the sort buttons.

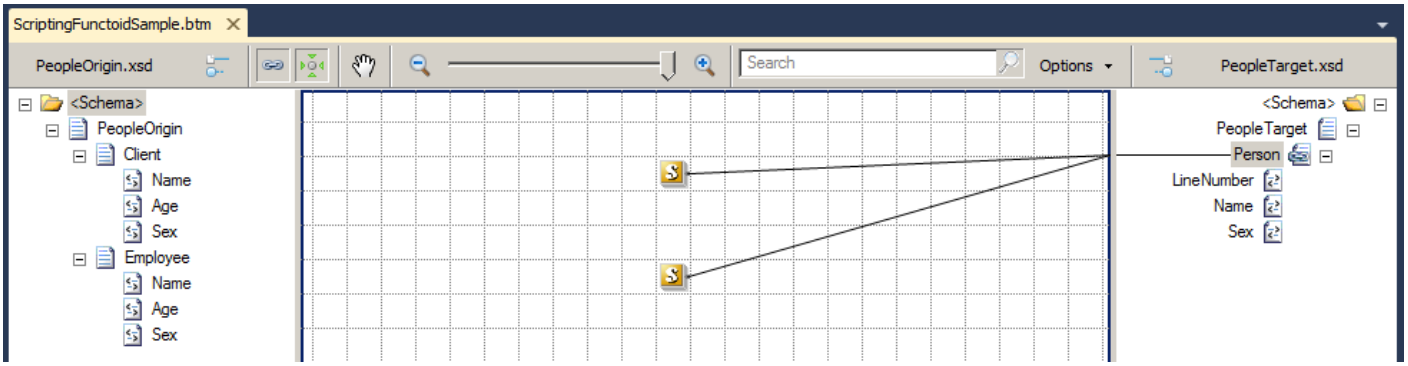


Impact of the order of links in elements of the destination schema

If we change the order in which we associate the links on the same element in the destination schema we can also have an impact on the desired final result.

Unfortunately, when we associated different links in the same element, there is no way or place in the graphical editor where you can check the order of the association, for example, similarly to what happens with the functoids. The only way to verify the order in these cases is to inspect the XSLT generated code or testing the map.

A good example of this scenario is when we associated two different Scripting functoid, both with custom inline XSLT scripts to the same destination element or record, once again, changing the order of the link association may have unexpected results



In this example the first Scripting functoid contain the following XSLT code:

```
<xsl:for-each select="Client">
     <Person>
          <Name>
               <xsl:value-of select="Name/text()" />
          </Name>
          <Sex>
               <xsl:value-of select="Sex/text()" />
          </Sex>
     </Person>
</xsl:for-each>
```

This code performs the mapping of all existing elements in the record "Client" from the source schema to the elements on the record "Person" in the destination schema.

The second Scripting functoid contain an identical XSLT code:

```
<xsl:for-each select="Employee">
     <Person>
          <Name>
               <xsl:value-of select="Name/text()" />
          </Name>
          <Sex>
               <xsl:value-of select="Sex/text()" />
          </Sex>
     </Person>
</xsl:for-each>
```

but this time it will map all existing elements in the record "Employee" from the source schema to the elements on the record "Person" in the destination schema.

The expected result is to appear in the final document all clients in the record "Person" and then all employees. If we change the order in which we association the links in the record “Person”, we will see that the result he also will be changed.

We can validate the result of this scenario in the following links:

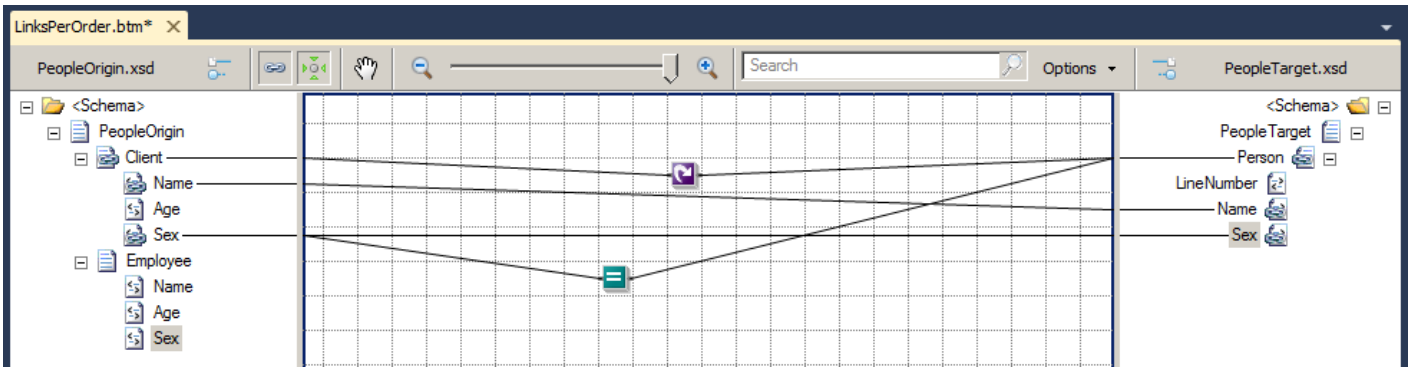
- Original Message:
  - https://gist.github.com/2624240
- Expected result:
  - https://gist.github.com/2624272
- Result if we change the order of the association:
  - https://gist.github.com/2624279

The rule of Link Sequence

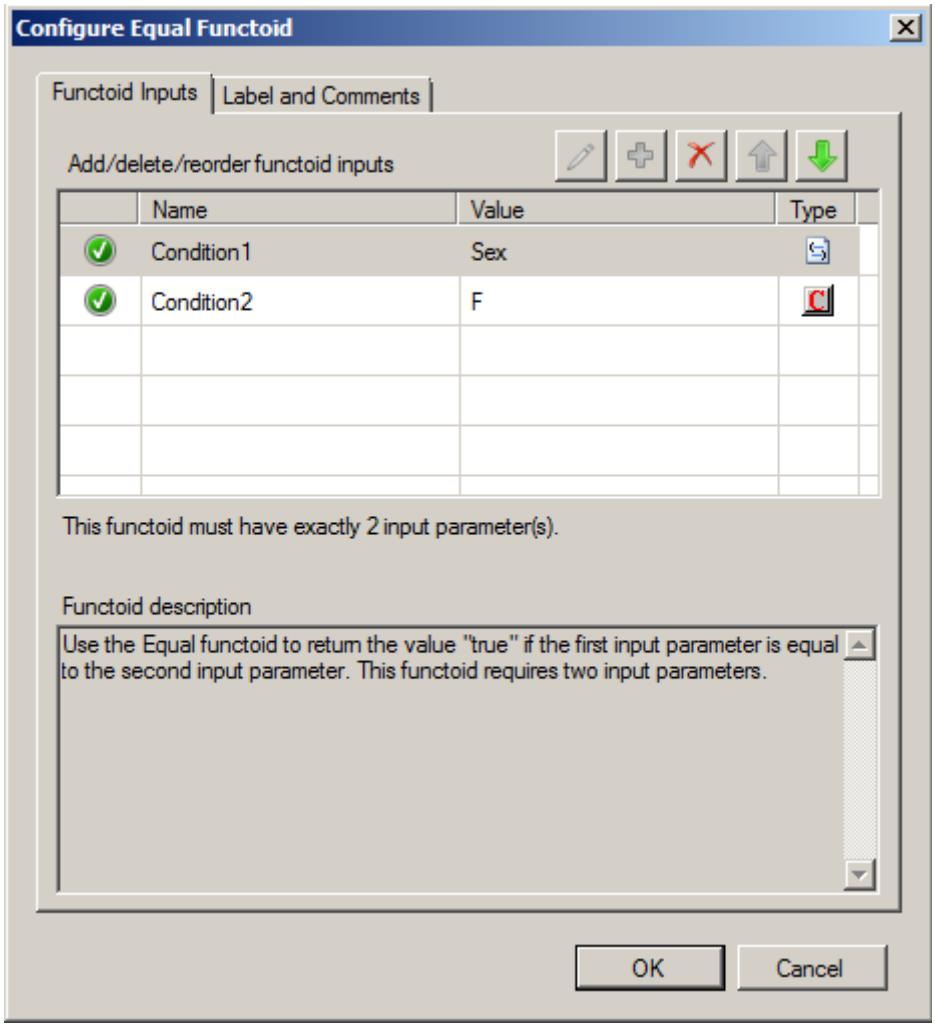
In brief, the mapping engine, process the rules by going through the destination schema from the beginning to the end, processing the links in the order that he finds and in case of multiple links in a particular element or functoid, they are processed in the order in which they were associated.

This means that the links associated with the parent nodes (records) are processed before links associated with the children (elements inside the record).

A good example of this scenario is the use of conditions in the parent node (record) when we want to influence the outcome according with a particular condition. So…let's find all the names of female clients. To do this we will create a map with the following settings:



- Open the Toolbox window and drag the Looping Functoid onto the grid;
  - Drag a link from the record “Client” from the source schema to the Looping Functoid;
  - Drag a link from the Looping Functoid to the record “Person” in the destination schema;
- Drag the Equal Functoid from the Toolbox window onto the grid;
  - Drag a link from the element “Sex” from the source schema to the Equal Functoid;
  - Drag a link from the Equal Functoid to the record “Person” in the destination schema;
  - Configured the Equal Functoid, by double click in the functoid, and edit the second condition with the value “F” to build a condition equivalent to Sex = "F";



The Equal Functoid will return the value “True” if the first input, in this case, the element “Sex” is equal to the second input parameter, ie, “F”. Otherwise it will return the value “False”. What will lead to: the record “Client” is only mapped if the condition returns the value "True".

If you look the code generated:

```
…
<ns0:PeopleTarget>
  <xsl:for-each select="Client">
     <xsl:variable name="var:v1" select="userCSharp:LogicalEq(string(Sex/text()) , "F")" />
     <xsl:if test="$var:v1">
        <Person>
           <Name>
              <xsl:value-of select="Name/text()" />
           </Name>
           <Sex>
              <xsl:value-of select="Sex/text()" />
           </Sex>
        </Person>
     </xsl:if>
  </xsl:for-each>
</ns0:PeopleTarget>
```

We can check that the first action of the map, after the cycle that travels the various elements of the record is: get the value generated in the Equal Functoid, which is represented in the variable "v1";

And the second action is to validate the condition (IF) with the value of the first operation (v1), ie, will test whether the value "v1" is "True" or “False”. If the condition is true the code within the condition is executed, otherwise it will move to the next element without doing anything. Therefore thus we obtain the desired result:

```
<ns0:PeopleTarget xmlns:ns0="http://HowMapsWorks.PeopleTarget">
  <Person>
    <Name>Elsa Ligia</Name>
    <Sex>F</Sex>
  </Person>
</ns0:PeopleTarget>
```

The exception to the rule of Link Sequence: Process links out of order

However there is one important exception to this rule of Link Sequence, especially when using custom scripts in recursive records or elements.

Once again, a good example of this scenario is the use of custom scripts to increment counters. We can illustrate this scenario by adding two Scripting Functoids to the map:

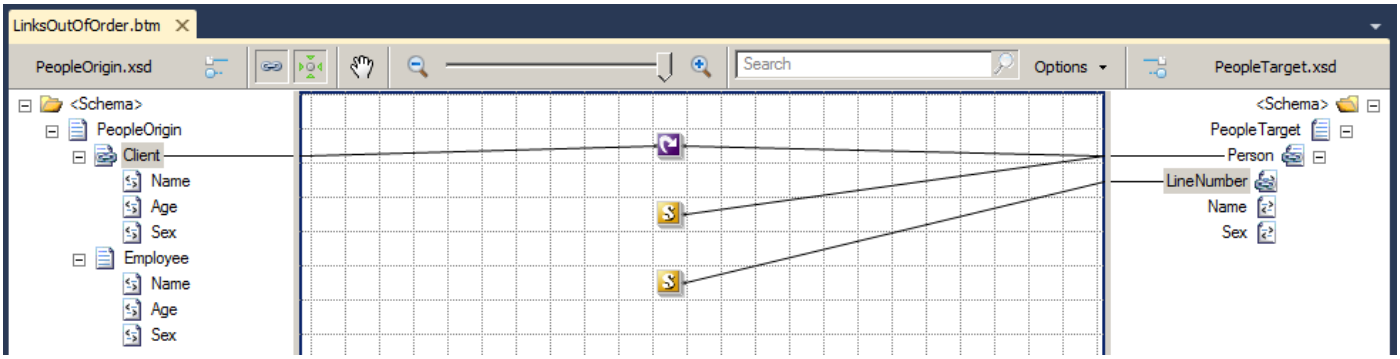
- The first containing the initialization and the function to increment the counter;

```
int myCounter = 0;
public void IncrementCounter()
{
    myCounter += 1;
}
```

- The second getting the value of the counter

```
public int ReturnCounter()
{
   return myCounter;
}
```

Note: This example will be associated with a loop, or with a recursive element.



We would expect that in the first cycle the result of the second script was the value "1", in the second cycle we obtained the value "2" and so on. However, if we test the map, we will see that the reality is different:

```
<ns0:PeopleTarget xmlns:ns0="http://HowMapsWorks.PeopleTarget">
   <Person><LineNumber>0</LineNumber></Person>
   <Person><LineNumber>1</LineNumber></Person>
   <Person><LineNumber>2</LineNumber></Person>
</ns0:PeopleTarget>
```

As we can see in the result above, the sequence in which the links are executed is:

- Create the record "PeopleTarget";
- Creating child elements, and carry out the link rules associated with them:
    - Execute the function "ReturnCounter" that will return the value "0" in the first iteration.
- Execute the links associated with the parent node:
    - Execution of the function “IncrementCounter”

As we can validate by checking code produced by the map:

```
…
<ns0:PeopleTarget>
  <xsl:for-each select="Client">
    <Person>
      <xsl:variable name="var:v1" select="userCSharp:ReturnCounter()" />
      <LineNumber>
        <xsl:value-of select="$var:v1" />
      </LineNumber>
      <xsl:variable name="var:v2" select="userCSharp:IncrementCounter()" />
      <xsl:value-of select="$var:v2" />
    </Person>
  </xsl:for-each>
</ns0:PeopleTarget>
```

Of course we can change the existing code in the Scripting Functoids so we can get around to this behavior and thus obtain the desired result. However, this example serves to warn that in some scenarios, especially in the use of custom scripts in recursive records or elements, it is necessary to verify and validate the sequence in which the rules (links) are executed.

Conclusion

With this article, as we explore some of the common mappings scenarios, trying to dismantle the options that the BizTalk Map engine has taken to fulfill with the original intent of the visual map.

When you begin to explore the world of maps, there are two questions that should evaluate carefully

- **What is the best way to solve a problem**: guaranteed there are several approaches to solve a common problem. Often deciding which the best way turns out to be the most difficult. Compile and analyze the code generated can be a good start to begin to understand the impact of certain options.
- **Incremental Testing**: very often we are tempted to try to solve a mapping problem from start to finish and only then we test solution. Leave it to the end can make it extremely difficult to detect problems in complex mappings. Limit the scope of testing should be a continuous and incremental process during the creation of maps, tests must be carried out as soon as a significant block is completed.

I hope this kind of hacking can help you to understand the behavior and debugging techniques for this type of elementary problems.

Author

Write By Sandro Pereira [MVP & MCTS BizTalk Server 2010]

Currently working as a BizTalk consultant at DevScope (www.devscope.net). In the last few years has been working implementing integration scenarios and Cloud Provisioning at a major telecommunications service provider in Portugal. His main focus is on Integration Technologies where is been using .NET, BizTalk and SOAP/XML/XSLT since 2002 and Windows Azure. Sandro is very active in the BizTalk community as blogger (http://sandroaspbiztalkblog.wordpress.com/), member and moderator on the MSDN BizTalk Server Forums, TechNet Wiki author, Code Gallery and CodePlex contributor, member of BizTalk Brazil community (http://www.biztalkbrasil.com.br/), NetPonto community (http://netponto.org/), BiztalkAdminsBlogging community (http://www.biztalkadminsblogging.com), editor of the magazine "Programar" (http://www.revista-programar.info/?action=editions), public speaker and technical reviewer of "BizTalk 2010 Cookbook", Packt Publishing book.

He has been awarded the Microsoft Most Valuable Professional (MVP) since January 2011 for his contributions to the world-wide BizTalk Server community (https://mvp.support.microsoft.com/profile/Sandro.Pereira) and is a certified MCTS: BizTalk Server BizTalk Server 2006 and BizTalk Server 2010.

You can contact Sandro at: sandro-pereira@live.com.pt (Twitter: @sandro_asp)

Sandro Pereira | DevScope | MVP & MCTS BizTalk Server 2010