Pro Mapping in BizTalk Server 2009

Copyright © 2009 by Jim Dawson, John Wainwright

ISBN-13 (pbk): 978-1-4302-1857-9

ISBN-13 (electronic): 978-1-4302-1858-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com.

PART 1

Basic Concepts of BizTalk Mapping

CHAPTER 1

Creating a Simple Map

This chapter introduces the basics of mapping with BizTalk, including a high-level overview of the Visual Studio tools with which you will work when creating maps. You will create a simple mapping project in Visual Studio and add to that project a source schema and a target schema. You will then create a map that moves the data from a set of address records in the source schema to the more compact structure of the target document. Finally, you will test the map.

If you are an experienced BizTalk developer, you will already be familiar with the information discussed in this chapter. However, if you are transitioning from other mapping tools to BizTalk and have neither BizTalk nor general development experience, this chapter is for you.

Using the Development Studio

The Visual Studio will be your home for you mapping activities, so as a preface, we'll do a quick tour of the various Visual Studio windows that you will use most frequently. Remember that, in Visual Studio, you can set most windows to float. Alternatively, you can dock them in several optional locations. The positions of the windows in our figures reflect how we happen to like them set up. You should arrange them in the positions that work best for you.

The Mapper Grid

The mapper grid shown in Figure 1-1, which has the Source Schema on the left and the Destination Schema on the right, is the main working window of the map editor. Each map may have up to 20 mapper grids, and each grid has one tab at the bottom. In the figure, there are two tabs, Page 1 and Page 2, thus this map will be spread over two grids.

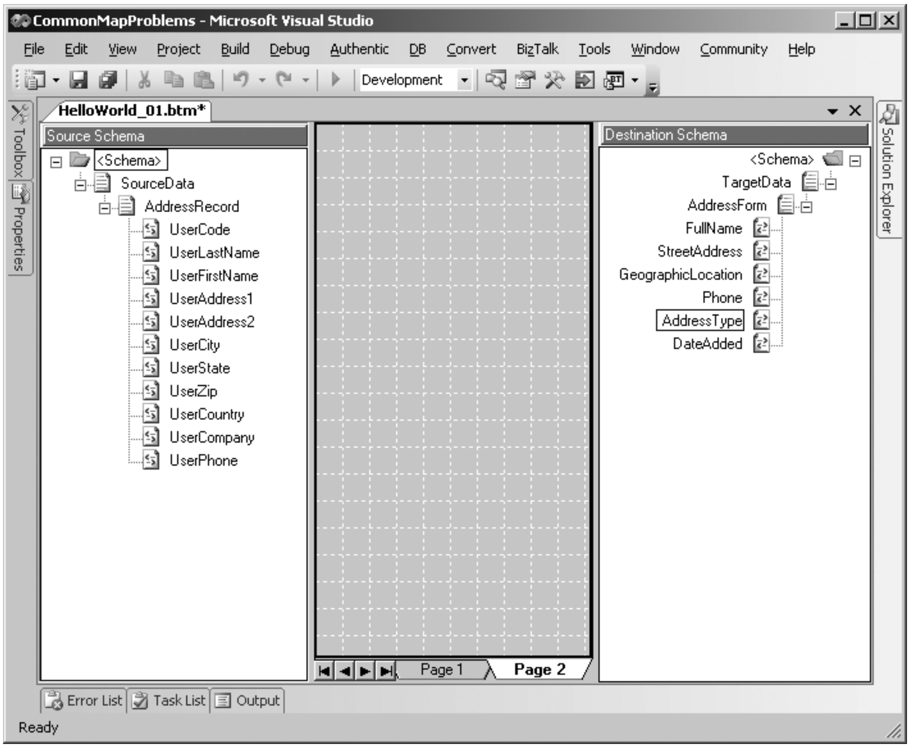


Figure 1-1. *The Visual Studio map editor*

Partitioning the map into grids is especially useful in reducing the clutter. You don't want the map to become so densely populated with functoids and links that you can't distinguish one item from another. By using multiple grids, you split the visual representation of the map onto more than one page. This technique allows you to focus on part of the map without having other parts of the map interfere with your view.

■ **Note** Mapper grids are self-contained when creating a map. For example, a link cannot begin on one grid page and end on another grid page. Thus you can work on only one grid at a time. The grids are invisible to the compiler, so organizing the map into many grids has no impact on how the map functions.

Figure 1-2 shows how to create a new grid. Right-click the existing tab, and click Add Page.

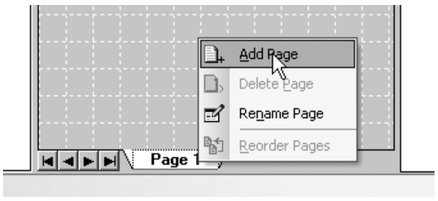


Figure 1-2. *Adding a grid page*

You can rename a new or existing grid tab by right-clicking the tab you wish to rename and replacing the existing text with your own text. Figure 1-3 shows the Rename Page selection.

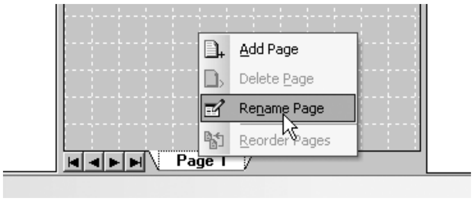


Figure 1-3. *Renaming a grid page*

Keep in mind that even when you have only one grid page, you still see only a very small percentage of the map. You can right-click the grid and select Grid Preview to see the entire grid for the current tab. Figure 1-4 shows one entire grid in the Grid Preview window.

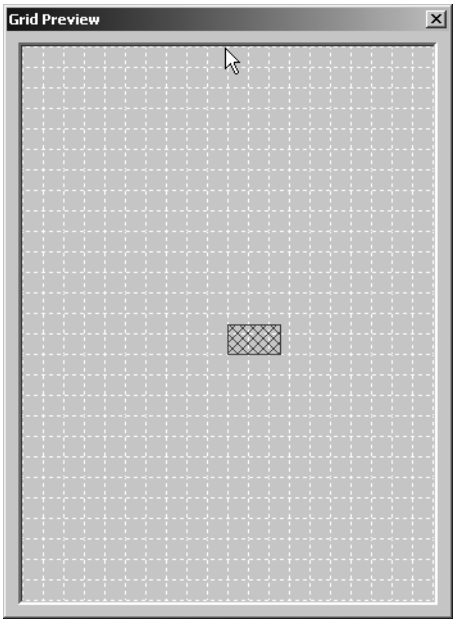


Figure 1-4. *The Grid Preview window*

The cross-hatched rectangle shows the area that will be displayed in the grid window when you close the grid preview. Links are not show in the preview window, but functoids are shown. To change the grid window view to show a particular functoid, drag the green box to the location of the functoid, and close the Grid Preview window. The functoid will now appear in the window.

■ **Tip** When you are looking at a mapper grid and the grid appears to be empty, you should check the Grid Preview window to see if there are functoids outside the current viewing area.

The Solution Explorer Window

The Solution Explorer shown in Figure 1-5 contains the current solution on which you are working along with all the projects and project files included in that solution. When working with maps you are mostly concerned with map files and schema files.

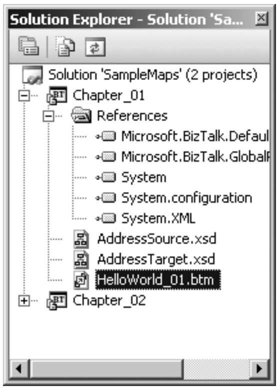


Figure 1-5. *The Solution Explorer window*

The Solution Explorer is a good window to keep handy. You use this window frequently during map development when you use incremental testing procedures. This is where you set parameters for map testing and start a map test. Testing is covered in detail in Chapter 3.

The Toolbox Window

The Toolbox window provides access to all of the functoids that you use in your map. The screen shot in Figure 1-6 shows you a standard display of the toolbar.

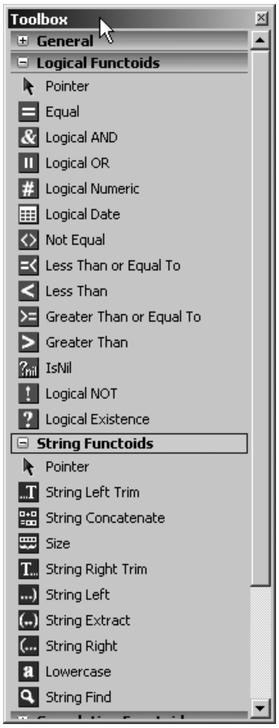


Figure 1-6. *The standard Toolbox window*

In BizTalk, you can customize the toolbox. By default, the General tab is empty. We use a subset of all the functoids more often than most and like to have the ones we use frequently in one location. We customize our toolbox by dragging and dropping selected icons into the General tab; our General tab is shown in Figure 1-7.

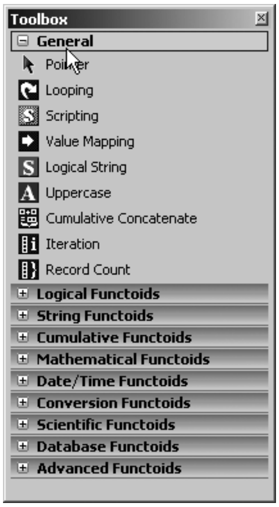


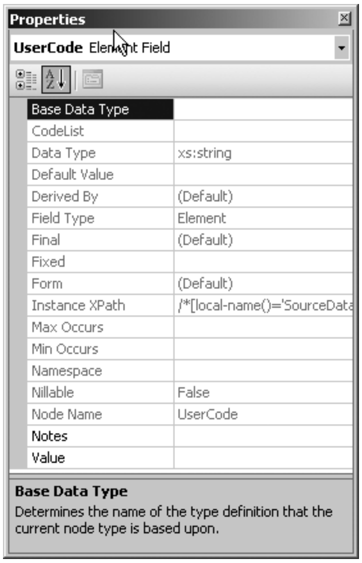
Figure 1-7. *Our Toolbox window's custom General tab*

You move an icon to a different tab using simple drag and drop. As you see, we selected functoids from the Advanced, Logical, String, and Cumulative Functoids tabs and moved them to the General tab. We also moved the Advanced Functoids menu from the top to the bottom, another drag-and-drop effort.

Tip The bulk of mapping is done with a small selection of the functoids. Moving the ones that you use most into one location, such as the General tab, will lessen the work you have to do to place functoids onto the mapper grid.

The Properties Window

The last menu you use extensively in mapping is the Properties window. Here, you can see the properties of whichever object you have currently highlighted in the mapper grid or in the source or destination schema windows. Figure 1-8 shows the Properties window as it appears when we highlight the root node of our target schema.



Properties

UserCode Element Field

Base Data Type	
CodeList	
Data Type	xs:string
Default Value	
Derived By	(Default)
Field Type	Element
Final	(Default)
Fixed	
Form	(Default)
Instance XPath	/*[local-name()='SourceData
Max Occurs	
Min Occurs	
Namespace	
Nillable	False
Node Name	UserCode
Notes	
Value	

Base Data Type
Determines the name of the type definition that the current node type is based upon.

Figure 1-8. *A sample Properties window*

The Working Environment

The working environment for mapping has lots of windows and takes up quite a bit of space on your monitor screen. Figure 1-9 is a screen shot of the way John arranges his windows for optimum use while he maps. This arrangement provides immediate access to all of the principal windows you use when creating maps.

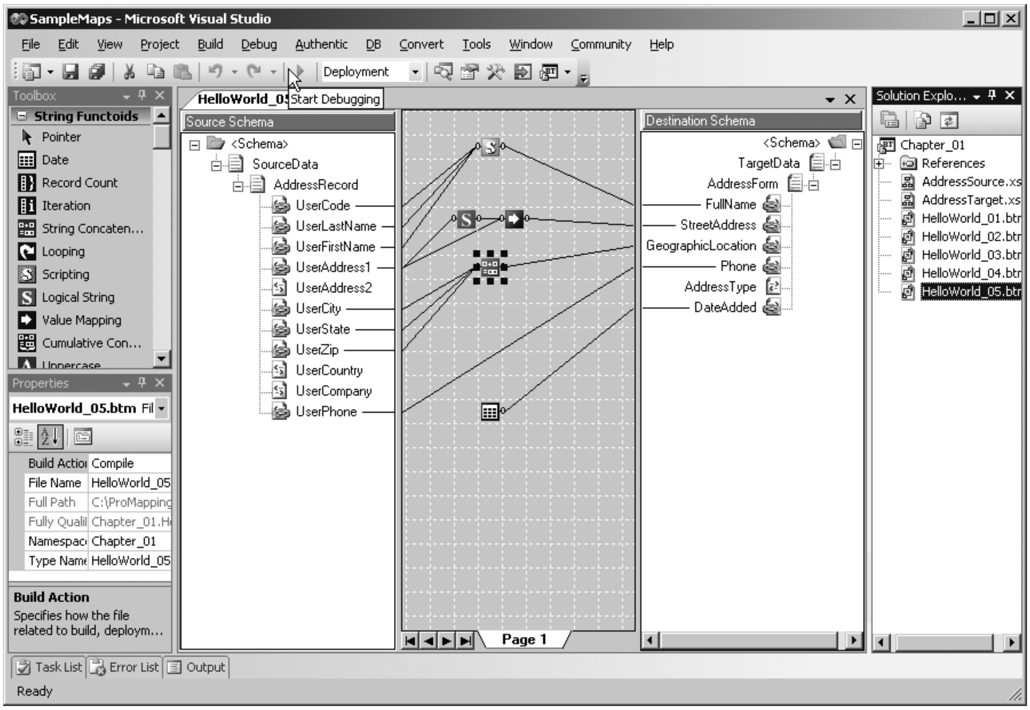


Figure 1-9. *One way to arrange your working screen*

Where John likes to keep his windows visible all the time, Jim uses a different arrangement and prefers to hide windows when the cursor is not over them. You should arrange your screen in the manner that fits your working methods.

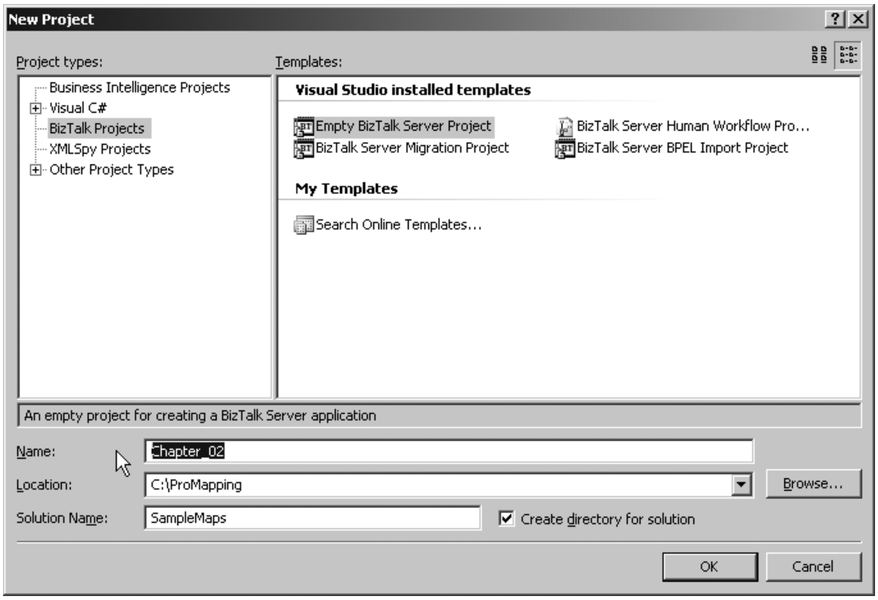
Creating the Map

Next, we will build a map from scratch, beginning with the creation of the project and solution. We'll start with a blank slate and go step by step—creating a new project in the development studio, importing schemas into the project, building a map, and testing the map.

Creating a BizTalk Project

Your first step is to open the development studio so that you can create a new BizTalk project. If you want your project to be placed under any particular directory structure, you should create that structure first, but do not create the project directory. The development studio will do that for you when you create your project.

In the development studio, go to the File drop-down, and select New ➤ Project. Fill in the window as shown in Figure 1-10.


New Project
Project types:
Templates:
Business Intelligence Projects
Visual C#
BizTalk Projects
XMLSpy Projects
Other Project Types
Visual Studio installed templates
Empty BizTalk Server Project
BizTalk Server Migration Project
BizTalk Server Human Workflow Pro...
BizTalk Server BPEL Import Project
My Templates
Search Online Templates...
Browse...
OK
Cancel

An empty project for creating a BizTalk Server application

Name: Chapter_02

Location: C:\ProMapping

Solution Name: SampleMaps

☑ Create directory for solution

Figure 1-10. *Creating a new project and solution*

When you click OK, the development studio creates the directory tree C:\ProMapping\SampleMaps\Chapter_01. SampleMaps is the solution directory; Chapter_01 is the project directory. Over the course of the rest of this book, we will build a project for each chapter, and those projects will all fall under this solution.

Caution Don't consider the solution and project structure we use in this book a recommended method. Our standard practices for this are much different. For example, normally, we create a separate project for each map. We've cheerfully violated our normal practices in this book to make loading projects for a specific chapter easier for you.

Once you have created the solution, click the Save All icon to save your work.

Adding a Project File Structure

Your basic mapping project and the default directories are built during the saving process. Inside your Chapter_01 directory, you will find some other default directories created by the development studio. There are two, bin and obj, in the Chapter_01 directory. Each of those has several subdirectories as well, all used by the development studio for building and deploying your project.

We want you to add some folders to the Chapter_01 directory. We use this setup because it works for us in this book. You will develop your own preferences as to how best to organize your folders. Let's add some subdirectories to Chapter_01. You can see them in Figure 1-11.

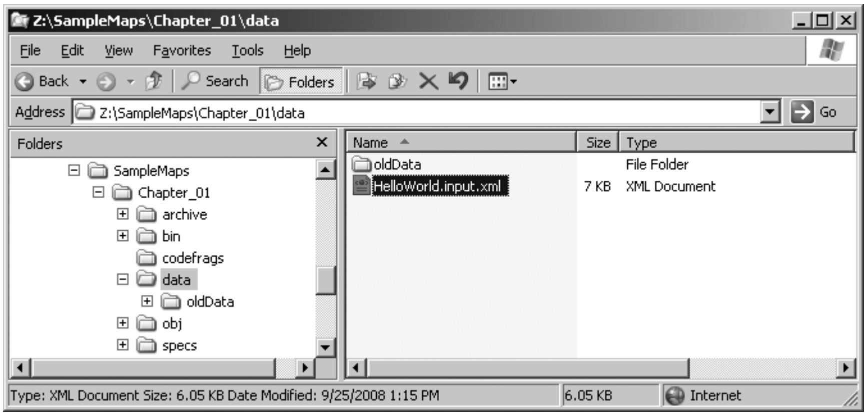

Z:\SampleMaps\Chapter_01\data
File Edit View Favorites Tools Help
Back Search Folders
Address Z:\SampleMaps\Chapter_01\data Go
Folders
SampleMaps
Chapter_01
archive
bin
codefrags
data
oldData
obj
specs
Name Size Type
oldData File Folder
HelloWorld.input.xml 7 KB XML Document
Type: XML Document Size: 6.05 KB Date Modified: 9/25/2008 1:15 PM 6.05 KB Internet

Figure 1-11. *Subdirectories for the project directory*

Create these subdirectories under the `Chapter_01` directory:

- `archive`: We save stages of our maps as we create them. We also go to Windows Explorer and create copies of the saved maps, renaming them to include a sequential number and/or a date time. As we accumulate backup copies, we find it helpful to move them to an archive file, just to reduce the clutter in the project directory.
- `codefrags`: This is where we keep copies of all the scripts we use in the map. We put details about the scripts in the name of the script. When the map is complete, we copy the scripts into a master library of code fragments. We have found our code fragment library to be a very valuable resource, since we use the same scripts in many maps.
- `data` *and* `oldData`: Data files in BizTalk mapping proliferate more rapidly than rabbits. We keep the inputs and associated outputs data files in these directories simply to make finding the correct files easier. Even if there are common data directories, isolating your map test data like this gives you a place where you can modify the data specific to the map if necessary. The `oldData` directory is where we put old data that has become outdated, perhaps due to a schema change, just in case we need to revert to it.
- `specs`: This holds any documentation related to the map, such as document specifications, mapping specifications, and notes from discussions about the map.

Creating the Input and Output Schemas

Each map requires a source and a target schema. Sometimes, you will have to create these schemas from scratch, but we have already built them for you for these exercises. For this map, you'll want to add `AddressSource.xsd` and `AddressTarget.xsd`. You will find these schemas in the support files that you downloaded for these exercises. Right-click the project name, and select Add ➤ Existing Item, as shown in Figure 1-12.

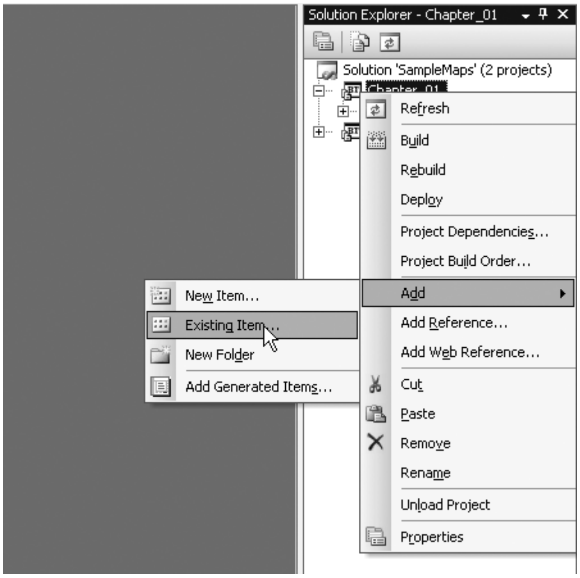


Figure 1-12. *Adding an existing item*

Now, browse to the schema files. Highlight the two schemas, as shown in Figure 1-13, and click the Add button to bring them into your project.

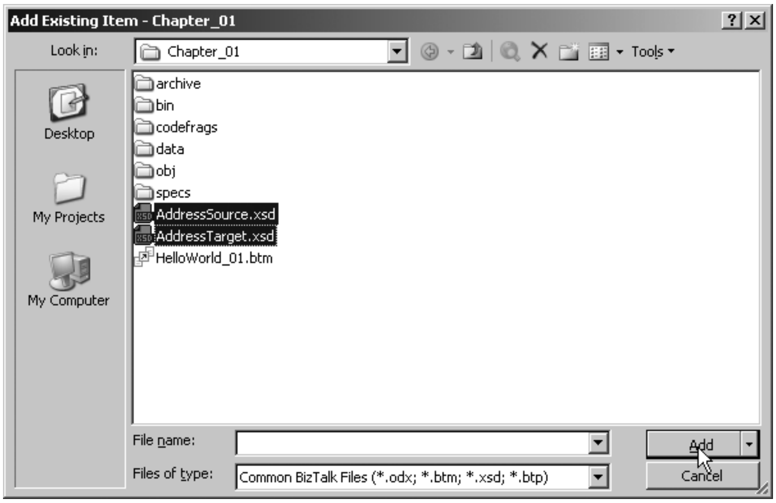
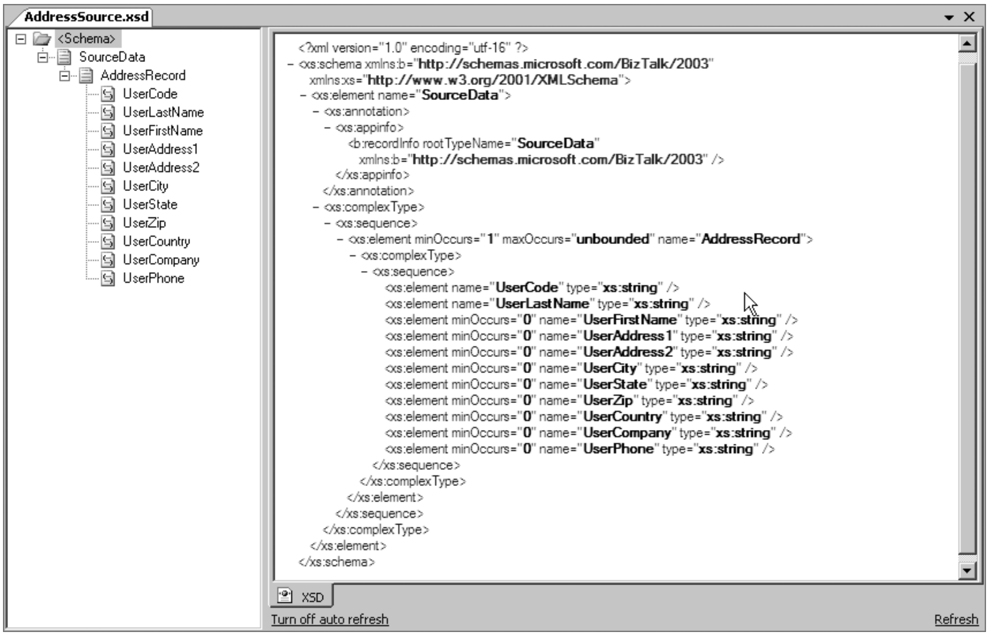


Figure 1-13. *Adding the schemas for Chapter 1*

Now that the schemas are in your project, you can open them in the schema editor. Double-click the schema name `AddressSource` in your project to open the source schema shown in Figure 1-14.

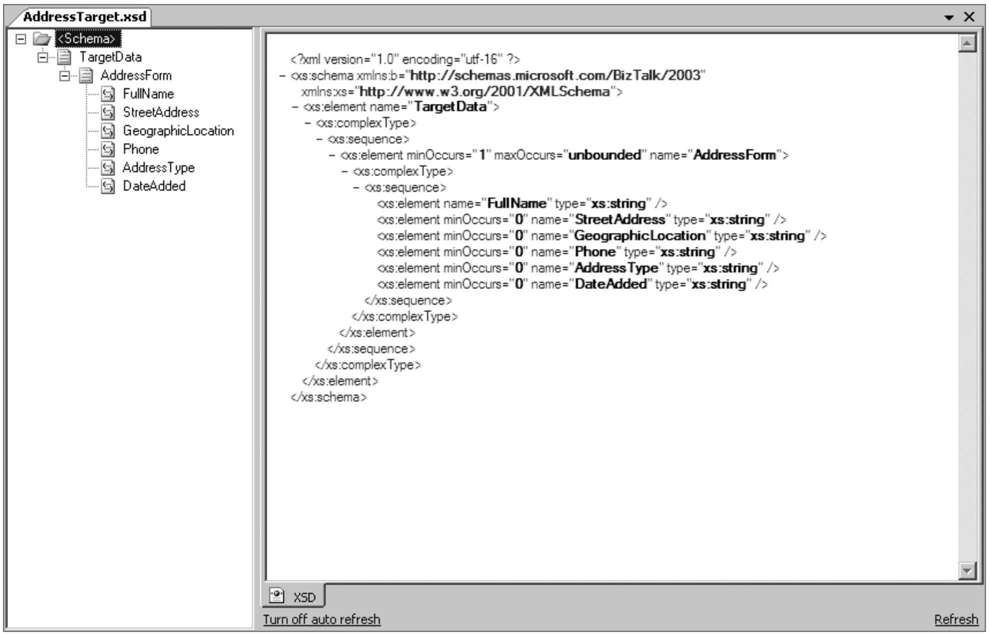


```
<Schema>
  SourceData
    AddressRecord
      UserCode
      UserLastName
      UserFirstName
      UserAddress1
      UserAddress2
      UserCity
      UserState
      UserZip
      UserCountry
      UserCompany
      UserPhone
```

```
  <?xml version="1.0" encoding="utf-16" ?>
- <xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  - <xs:element name="SourceData">
    - <xs:annotation>
      - <xs:appinfo>
          <b:recordInfo rootTypeName="SourceData"
            xmlns:b="http://schemas.microsoft.com/BizTalk/2003" />
        </xs:appinfo>
      </xs:annotation>
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element minOccurs="1" maxOccurs="unbounded" name="AddressRecord">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="UserCode" type="xs:string" />
                <xs:element name="UserLastName" type="xs:string" />
                <xs:element minOccurs="0" name="UserFirstName" type="xs:string" />
                <xs:element minOccurs="0" name="UserAddress1" type="xs:string" />
                <xs:element minOccurs="0" name="UserAddress2" type="xs:string" />
                <xs:element minOccurs="0" name="UserCity" type="xs:string" />
                <xs:element minOccurs="0" name="UserState" type="xs:string" />
                <xs:element minOccurs="0" name="UserZip" type="xs:string" />
                <xs:element minOccurs="0" name="UserCountry" type="xs:string" />
                <xs:element minOccurs="0" name="UserCompany" type="xs:string" />
                <xs:element minOccurs="0" name="UserPhone" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Figure 1-14. *AddressSource.xsd*

The source schema contains an address record. The target schema, shown in Figure 1-15, contains a subset of that information.



```
<Schema>
  TargetData
    AddressForm
      FullName
      StreetAddress
      GeographicLocation
      Phone
      AddressType
      DateAdded
```

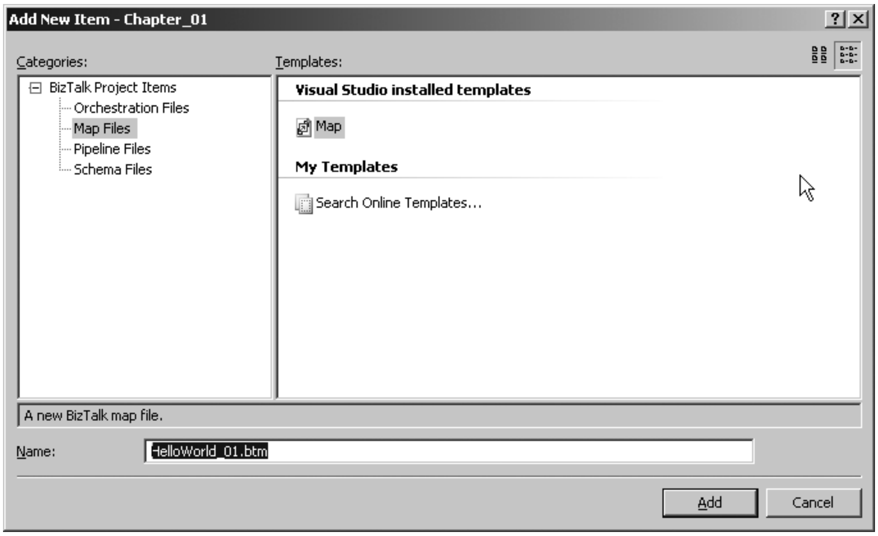
```
  <?xml version="1.0" encoding="utf-16" ?>
- <xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  - <xs:element name="TargetData">
    - <xs:complexType>
      - <xs:sequence>
        - <xs:element minOccurs="1" maxOccurs="unbounded" name="AddressForm">
          - <xs:complexType>
            - <xs:sequence>
                <xs:element name="FullName" type="xs:string" />
                <xs:element minOccurs="0" name="StreetAddress" type="xs:string" />
                <xs:element minOccurs="0" name="GeographicLocation" type="xs:string" />
                <xs:element minOccurs="0" name="Phone" type="xs:string" />
                <xs:element minOccurs="0" name="AddressType" type="xs:string" />
                <xs:element minOccurs="0" name="DateAdded" type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Figure 1-15. *AddressTarget.xsd*

Now, you have the two items required before you can create a map in your project.

Creating a New BizTalk Map File

You create a new map by right-clicking the project name in the Solution Explorer and selecting Add ➤ New Item. The pop-up window shown in Figure 1-16 appears. Select Map Files, then Map, and then type the map name into the Name field. Name the map `HelloWorld.btm`.



A new BizTalk map file.

Name: HelloWorld_01.btm

Figure 1-16. *Creating a new map file*

When you click the Add button, the empty map will open automatically to allow you to choose the source and target schemas that you want to use. Figure 1-17 shows the empty map.

Click the Open Source Schema link in the left pane, and you will open the BizTalk Type Picker window shown in Figure 1-17. Expand the Schemas folder, and select Chapter_01. AddressSource. This is the pointer to your source schema `AddressSource.xsd`.

Your source schema will appear in the left pane. When it does, repeat this process for your target schema. Once both schemas have been opened, your map will look like Figure 1-18.

As you see, all the elements are present for you to begin mapping. Click Save All in the development studio to save your work.

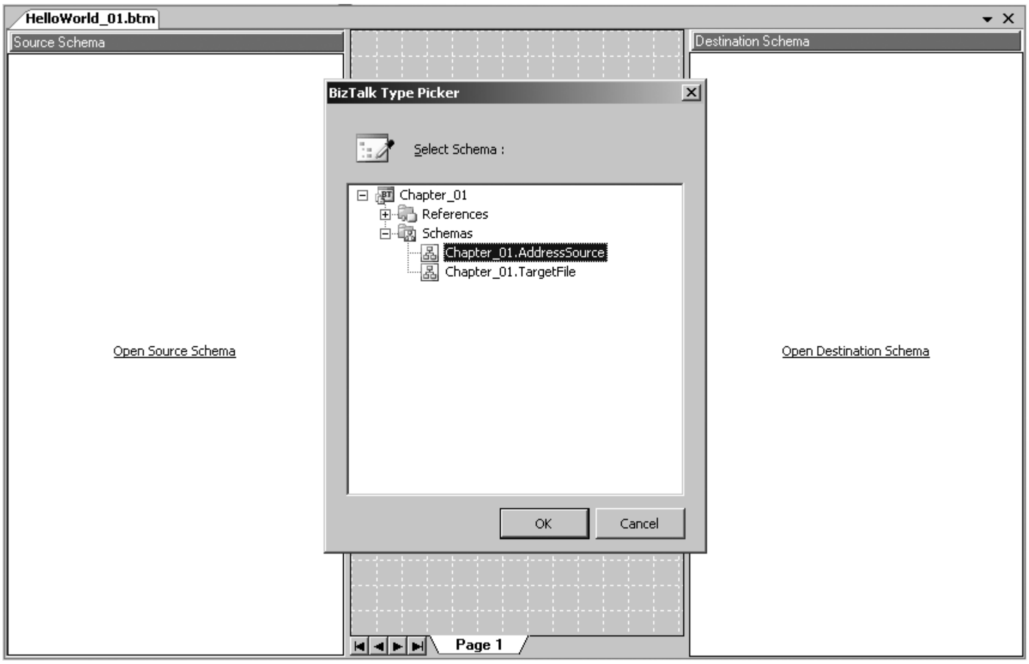


Figure 1-17. *An empty map*

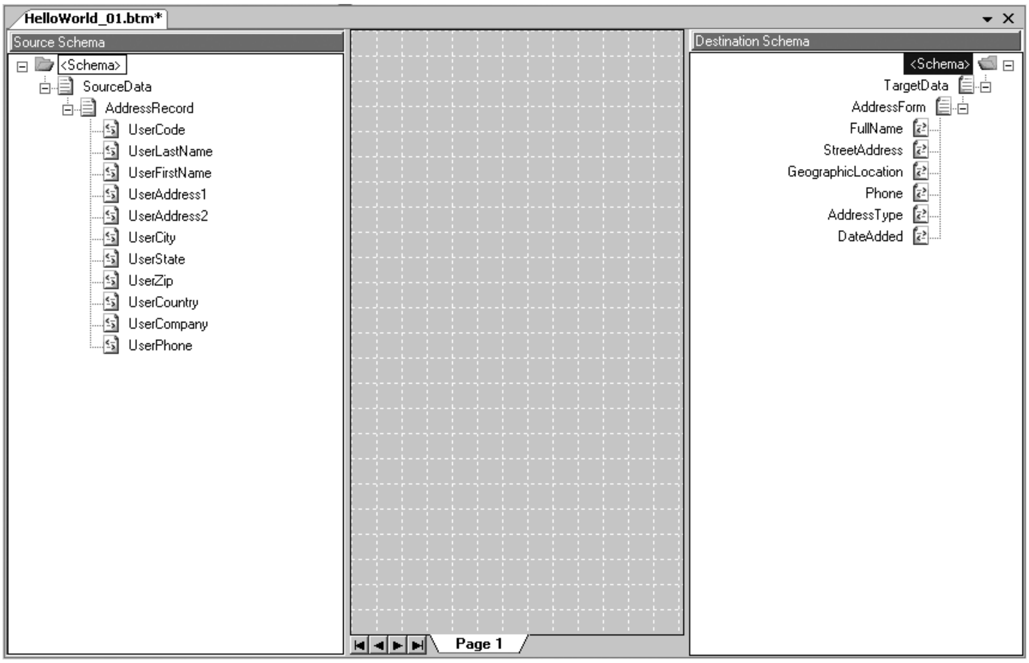


Figure 1-18. *Your new map*

A Test Input file

Before working on the map, we want to create or find a source data file. Copy the test file named `HelloWorldInput.xml` from the download files for Chapter 1 into the data subdirectory under your project. Listing 1-1 shows the input file.

Listing 1-1. *HelloWorldInput.xml*

```
<SourceData>
  <AddressRecord>
    <UserCode>001</UserCode>
    <UserLastName>Wainwright</UserLastName>
    <UserFirstName>John</UserFirstName>
    <UserAddress1>7008 Robbie Drive</UserAddress1>
    <UserAddress2/>
    <UserCity>Raleigh</UserCity>
    <UserState>NC</UserState>
    <UserZip/>
    <UserCountry>USA</UserCountry>
    <UserCompany>SSPS</UserCompany>
    <UserPhone>555-612-2222</UserPhone>
  </AddressRecord>
  <AddressRecord>
    <UserCode>002</UserCode>
    <UserLastName>Dawson</UserLastName>
    <UserFirstName>Jim</UserFirstName>
    <UserAddress1>230 Shadow Smoke Lane</UserAddress1>
    <UserAddress2/>
    <UserCity>Siler City</UserCity>
    <UserState>NC</UserState>
    <UserZip/>
    <UserCountry>USA</UserCountry>
    <UserCompany>SSPS</UserCompany>
    <UserPhone>555-444-3331</UserPhone>
  </AddressRecord>
  <AddressRecord>
    <UserCode>003</UserCode>
    <UserLastName>Accounts Payable</UserLastName>
    <UserFirstName/>
    <UserAddress1>7008 RedWing</UserAddress1>
    <UserAddress2/>
    <UserCity>Apex</UserCity>
    <UserState>NC</UserState>
    <UserZip/>
    <UserCountry>USA</UserCountry>
    <UserCompany>SSPS</UserCompany>
    <UserPhone>555-876-1234</UserPhone>
  </AddressRecord>
</SourceData>
```

■ **Tip** There are many ways that you can open and view data, raw schema, and map files. We rely on two editors that make mapping easier. Altova XMLSpy is an XML editor and development environment that integrates with Visual Studio. IDM Computer Solutions, Inc., provides UltraEdit, a text editor with many features including the capability to view files in binary.

At this point, we have completed all the preliminaries necessary to begin creating the map. We have set up a project in the development studio with the input and output schemas, and we have created a test file.

Building the Map

Finally, we have reached the fun stuff, actually creating a map. The basics of building a map consist either of dragging links from the source window directly to the target window or of dragging links to and from functoids that have been inserted into the mapper grid. You will implement five different processes in this map:

- Simple drag-and-drop
- Concatenation
- Data addition
- Conditional selection
- Custom scripting

Dragging and Dropping Fields

This is the most basic of all mapping steps. We want the map to move the data found in the source node `UserPhone` into the target node `Phone`.

1. Click `UserPhone` in the source schema pane, and hold down the left mouse button.
2. Move the mouse to `Phone` in the target schema pane. When `Phone` highlights, release the mouse button.

As you move the mouse from `UserPhone` to `Phone`, a line forms behind the mouse. This line represents the link that you are creating between the source and target nodes. When you release the mouse button, the line remains, depicting the completed link, as shown in Figure 1-19.

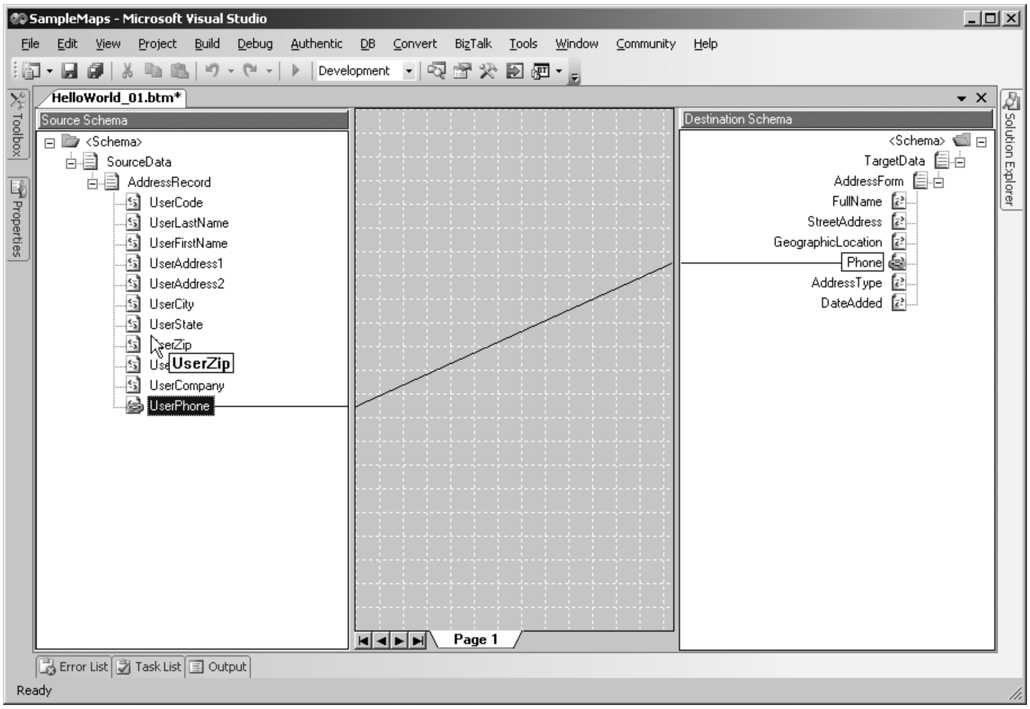
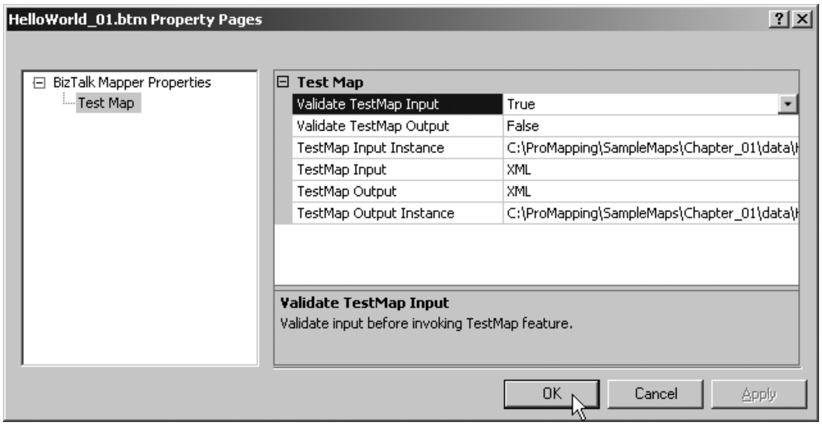


Figure 1-19. *Dragging and dropping a link*

Next, we want to test the link to make sure it does what we expect. Building a map and testing the map in BizTalk are not separate activities. Each change that you make to the map may have an impact on previous work, even to the extent of changing the behavior. This is particularly true when you work with loops.

■ **Tip** Testing should be a continuous process as you build your map. Although you don't need to test after every new link, you should test your map as soon as you complete any significant block of work. The drag-and-drop link in this map doesn't warrant testing, but we are inserting a quick overview of testing here to emphasize that it is *never* too early to start testing your map. Of course, the more complex the map, the more important testing becomes.

First, we must set the testing properties for the map. Right-click the map name in the Solution Explorer window, and select Properties. This will bring up the map properties window shown in Figure 1-20.


HelloWorld_01.btm Property Pages

Test Map

Validate TestMap Input	True
Validate TestMap Output	False
TestMap Input Instance	C:\ProMapping\SampleMaps\Chapter_01\data\
TestMap Input	XML
TestMap Output	XML
TestMap Output Instance	C:\ProMapping\SampleMaps\Chapter_01\data\

Validate TestMap Input
Validate input before invoking TestMap feature.

Figure 1-20. *Map properties*

Set the Validate Test Map Output option to False. This allows you to test an incomplete map without the test failing for missing mandatory data. Set the TestMap Input Instance by browsing to your input data file, `HelloWorldInput.xml`. Use the same path for your TestMap Output Instance, changing the file name to `HelloWorldOutput.xml`. This last step is optional but handy because it causes a copy of the output to be produced in your data directory where you can manipulate it if needed. Select OK to test your link.

Execute the test by right-clicking the map name and selecting Test Map, as shown in Figure 1-21.

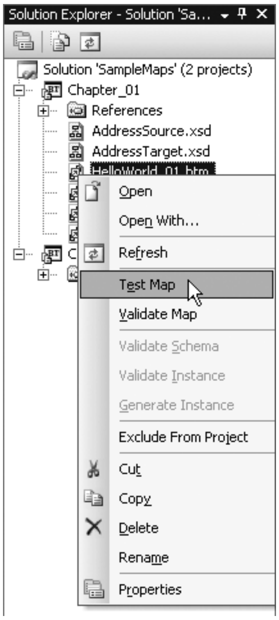
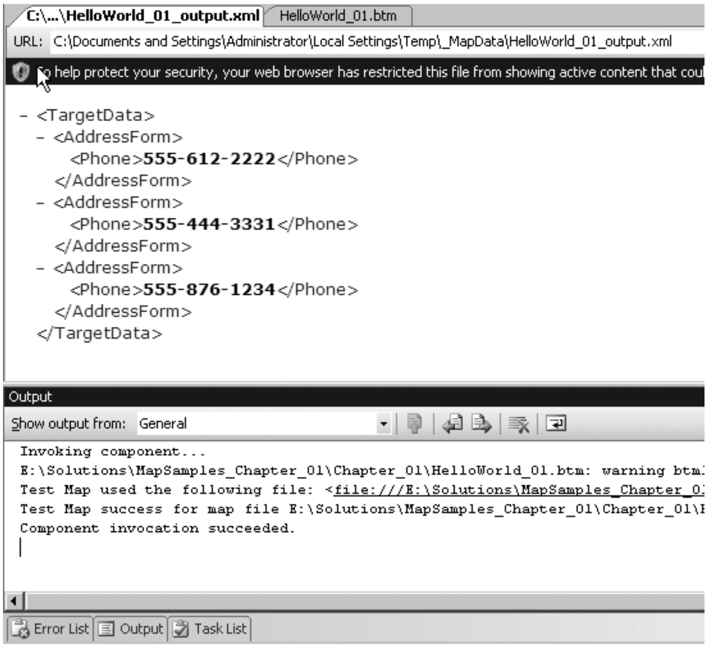


Figure 1-21. *Executing the test*

The test should run instantaneously. If the test is successful, the output window will pop up at the bottom of the screen. Hold down the Ctrl key, and click the link to the output file to cause the map editor to open the output. As Figure 1-22 shows, the map outputs one `Phone` node for each `UserPhone` node in the input file.


C:\...\HelloWorld_01_output.xml
HelloWorld_01.btm
URL: C:\Documents and Settings\Administrator\Local Settings\Temp_MapData\HelloWorld_01_output.xml
To help protect your security, your web browser has restricted this file from showing active content that cou
Output
Show output from: General
Error List
Output
Task List

```
- <TargetData>
  - <AddressForm>
      <Phone>555-612-2222</Phone>
    </AddressForm>
  - <AddressForm>
      <Phone>555-444-3331</Phone>
    </AddressForm>
  - <AddressForm>
      <Phone>555-876-1234</Phone>
    </AddressForm>
  </TargetData>
```

```
Invoking component...
E:\Solutions\MapSamples_Chapter_01\Chapter_01\HelloWorld_01.btm: warning btm
Test Map used the following file: <file:///E:\Solutions\MapSamples_Chapter_0
Test Map success for map file E:\Solutions\MapSamples_Chapter_01\Chapter_01\
Component invocation succeeded.
```

Figure 1-22. *Output from first test*

You can validate this test by comparing the source data with the output, checking the phone numbers to make sure that they were correct and in the correct order.

Concatenating Fields

One common mapping need is to combine two or more strings from the input. The mapping specifications for this map are that the values from three source nodes, `UserCity`, `UserState`, and `UserZip` must be output as one value in the target node `GeographicLocation`. The mapping specifications also states that the fields must be separated by single spaces. Here are the steps to concatenate fields:

1. Open the Toolbox window, and drag the String Concatenate functoid onto the grid.
2. Drag and drop a link from UserCity to the String Concatenate functoid.
3. Drag and drop a link from UserState to the String Concatenate functoid.
4. Drag and drop a link from UserZip to the String Concatenate functoid.
5. Drag and drop a link from the String Concatenate functoid to GeographicLocation in the target.

The order of the inputs to the functoid is important, since the concatenation will be done in the order that the inputs are received. Your map now looks like Figure 1-23.

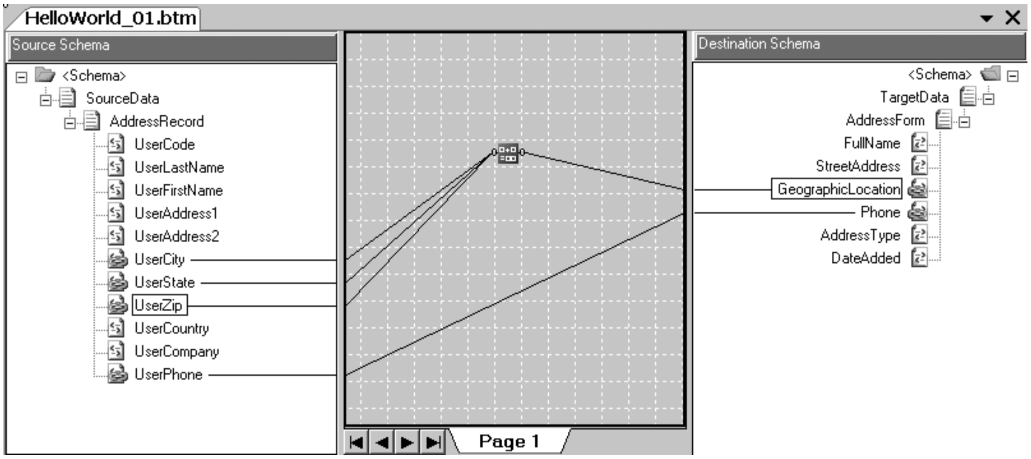
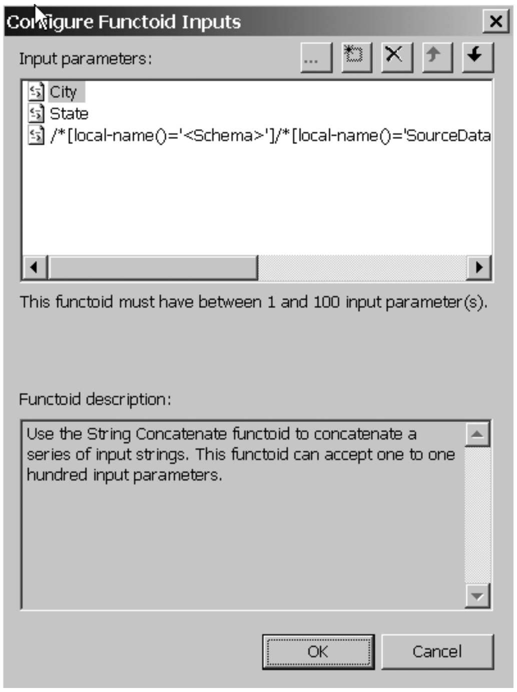


Figure 1-23. *Concatenating UserCity, UserState, and UserZip*

You now must add the spaces to the concatenation. Double-click the String Concatenate functoid to open the Configure Functoid Inputs window shown in Figure 1-24.



City
State
/*[local-name()='<Schema>']/*[local-name()='SourceData

This functoid must have between 1 and 100 input parameter(s).

Use the String Concatenate functoid to concatenate a series of input strings. This functoid can accept one to one hundred input parameters.

Figure 1-24. *The Configure Functoid Inputs window*

You can see your three inputs in the Input Parameters. We have named two of the links and left the third with the default name to illustrate the difference that naming links can make in understanding which input is which.

Tip You can give a link a name by right-clicking the line in the grid that represents the link and selecting Properties. The Properties window will have a field, Label, in which you type the name.

Remember that the mapping specifications stated that the fields should be separated with single spaces. You now must add the spaces. The second icon at the top of the Configure Functoid Inputs window is Insert New Parameter. Click the input parameter City and then the Insert New Parameter icon. A new input parameter, along with an open text box, will appear following City, as shown in Figure 1-25.

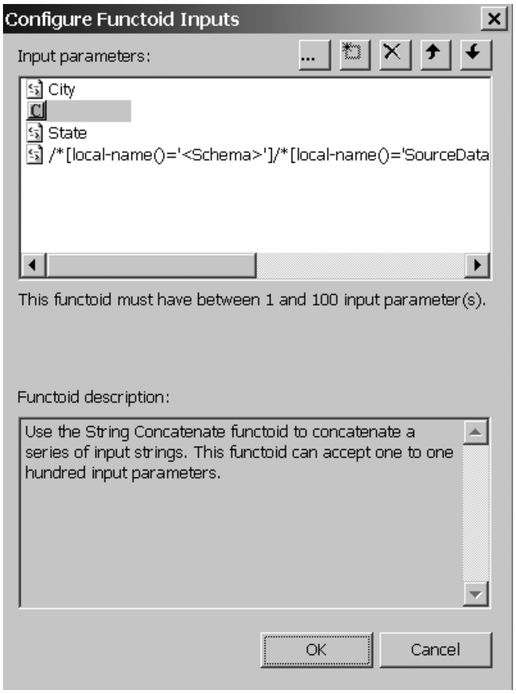

Configure Functoid Inputs
Input parameters:
...
City
State
/*[local-name()='<Schema>']/*[local-name()='SourceData
This functoid must have between 1 and 100 input parameter(s).
Functoid description:
Use the String Concatenate functoid to concatenate a series of input strings. This functoid can accept one to one hundred input parameters.
OK
Cancel

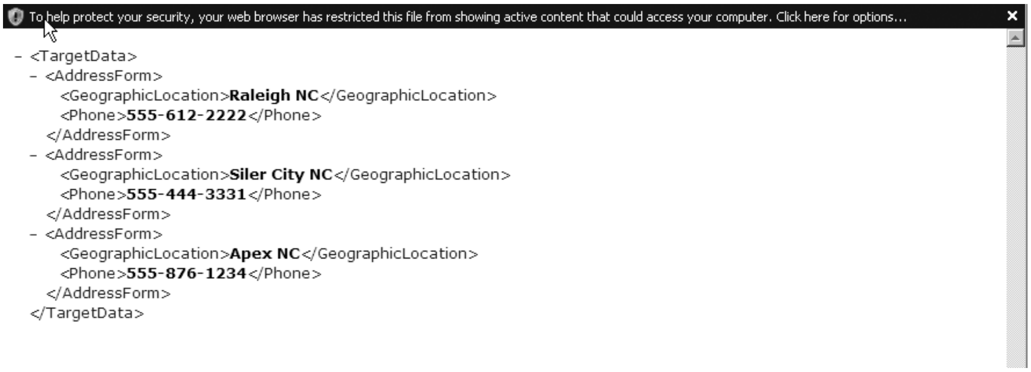
Figure 1-25. *Adding spaces to the String Concatenate functoid's input*

Type a single space into the text box and press the Enter key. You now have a space between the City and State. Next, insert a space between the State and Zip input parameters. When you finish, close the Configure Functoid Inputs window by clicking OK.

Tip If you accidentally enter the new parameter in the wrong location, change the order of the parameters by using the up and down arrow icons to move your input parameters up and down in the list.

Now, you need to make sure that you have set up your concatenation properly. Test your map again by right-clicking the map name in the Solution Explorer window and selecting Test Map.

Your output should look like the output shown in Figure 1-26. Since there are no ZIP codes in the source data, only the city and state appear.



```
To help protect your security, your web browser has restricted this file from showing active content that could access your computer. Click here for options...

- <TargetData>
  - <AddressForm>
      <GeographicLocation>Raleigh NC</GeographicLocation>
      <Phone>555-612-2222</Phone>
    </AddressForm>
  - <AddressForm>
      <GeographicLocation>Siler City NC</GeographicLocation>
      <Phone>555-444-3331</Phone>
    </AddressForm>
  - <AddressForm>
      <GeographicLocation>Apex NC</GeographicLocation>
      <Phone>555-876-1234</Phone>
    </AddressForm>
  </TargetData>
```

Figure 1-26. *Output with the String Concatenate functoid in the map*

Adding a System Date

Another common mapping step is adding a system date to the output. In this case, you will put the system date into the `DateAdded` node. Use the Date functoid from the toolbox as follows to retrieve the current system date:

1. Drag the Date icon from the Toolbox window onto the mapper grid.
2. Drag a link from the Date functoid to the DateAdded node in the target.

The Date functoid does not accept input parameters, as it simply returns the system date in the format YYYY-MM-DD. Your map should look like Figure 1-27 when you've added this mapping rule.

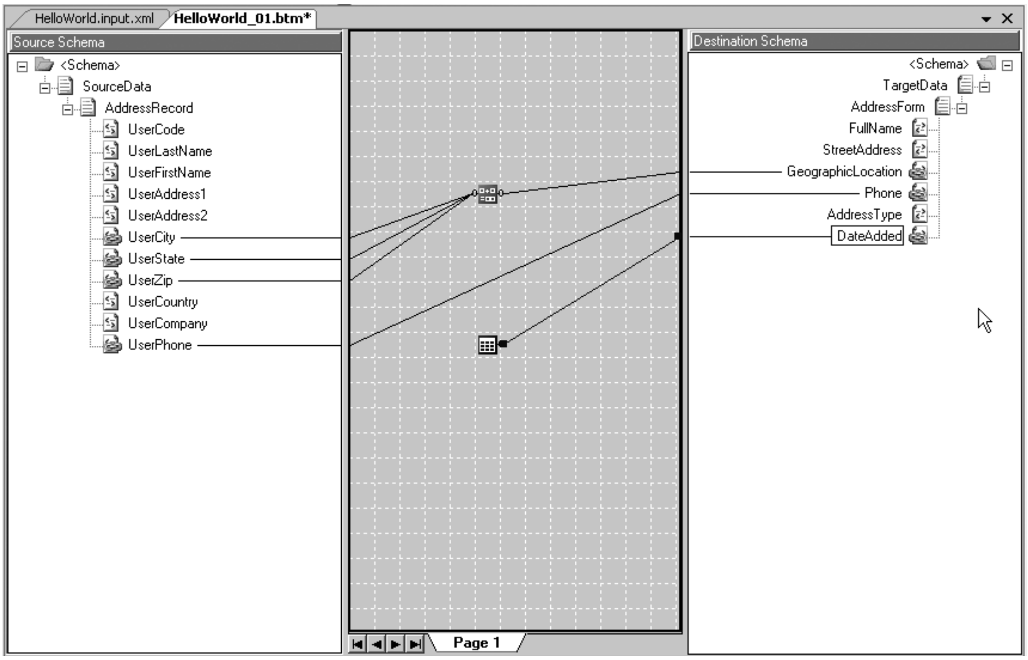
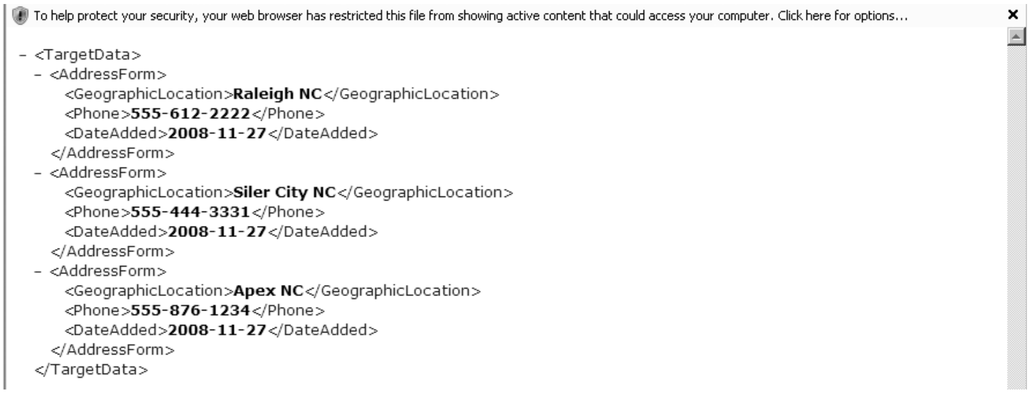


Figure 1-27. *The System Date functoid*

Once again, run a quick test to verify your output; the result should look like Figure 1-28.


To help protect your security, your web browser has restricted this file from showing active content that could access your computer. Click here for options...

```
- <TargetData>
  - <AddressForm>
      <GeographicLocation>Raleigh NC</GeographicLocation>
      <Phone>555-612-2222</Phone>
      <DateAdded>2008-11-27</DateAdded>
    </AddressForm>
  - <AddressForm>
      <GeographicLocation>Siler City NC</GeographicLocation>
      <Phone>555-444-3331</Phone>
      <DateAdded>2008-11-27</DateAdded>
    </AddressForm>
  - <AddressForm>
      <GeographicLocation>Apex NC</GeographicLocation>
      <Phone>555-876-1234</Phone>
      <DateAdded>2008-11-27</DateAdded>
    </AddressForm>
  </TargetData>
```

Figure 1-28. *Output of map with System Date functoid*

Moving Data Conditionally

You won't always be able to simply move data from source to target. Sometimes, you will need to generate an output node only under certain conditions. We'll use a simple requirement for this map. You are to output the `StreetAddress` node only if there is data in the `UserAddress1` node. As with many map operations, there are multiple ways to achieve the desired result. You should do it this time using the Logical String and Value Mapping functoids.

The Logical String functoid tests one value from the source to see if the value is a string and outputs the result of that test as either true or false. The Value Mapping functoid provides a means for us to control the output of a value from the source using the `true`/`false` value received from the Logical String functoid. When the Value Mapper receives true from the Logical String functoid, it outputs the value from the source.

Here are the steps to add these functoids to your map:

1. Drag a Logical String functoid to the grid.
2. Drag a link from UserAddress1 to the Logical String functoid (this is the first input).
3. Drag a Value Mapping functoid to the grid.
4. Drag a link from the Logical String functoid to the Value Mapping functoid.
5. Drag a link from UserAddress1 to the Value Mapping functoid (this is the second input).
6. Drag a link from the Value Mapping functoid to the StreetAddress node.

Your map should now look like Figure 1-29.

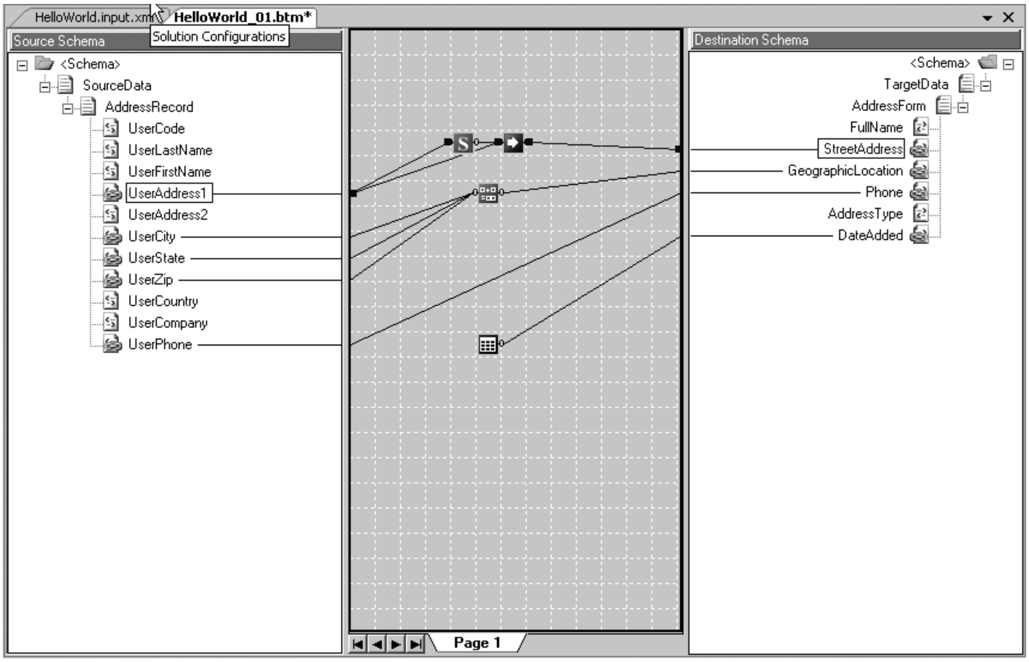
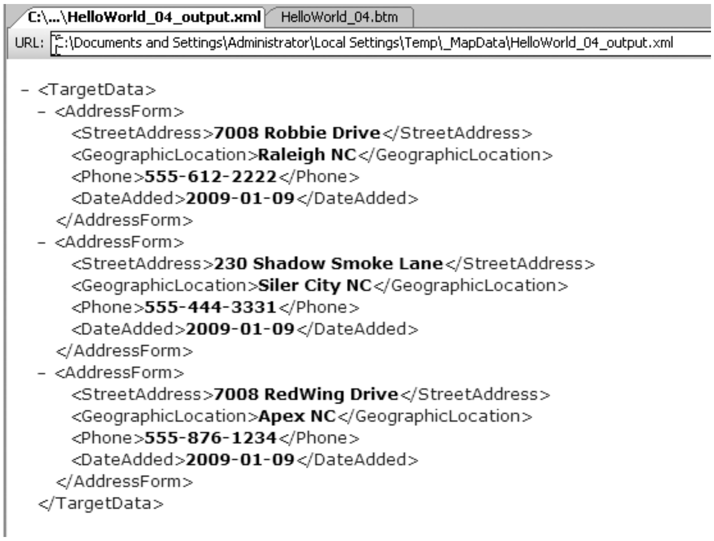


Figure 1-29. *A conditional statement using the Logical String and Value Mapping functoids*

Test your map again to get the output shown in Figure 1-30.


C:\...\HelloWorld_04_output.xml
HelloWorld_04.btm
URL: C:\Documents and Settings\Administrator\Local Settings\Temp_MapData\HelloWorld_04_output.xml

```
- <TargetData>
  - <AddressForm>
      <StreetAddress>7008 Robbie Drive</StreetAddress>
      <GeographicLocation>Raleigh NC</GeographicLocation>
      <Phone>555-612-2222</Phone>
      <DateAdded>2009-01-09</DateAdded>
    </AddressForm>
  - <AddressForm>
      <StreetAddress>230 Shadow Smoke Lane</StreetAddress>
      <GeographicLocation>Siler City NC</GeographicLocation>
      <Phone>555-444-3331</Phone>
      <DateAdded>2009-01-09</DateAdded>
    </AddressForm>
  - <AddressForm>
      <StreetAddress>7008 RedWing Drive</StreetAddress>
      <GeographicLocation>Apex NC</GeographicLocation>
      <Phone>555-876-1234</Phone>
      <DateAdded>2009-01-09</DateAdded>
    </AddressForm>
  </TargetData>
```

Figure 1-30. *Conditional output*

As expected, the map works again. Stay in the habit of incremental testing as we are doing with this map, though, because you won't always be this lucky.

Using a Scripting Functoid

The last item that we need to move from the source data to the output data is the name. The target has a `FullName` node; the source has `UserFirstName` and `UserLastName` nodes. At first, this looks like a simple concatenation like you did earlier. But we're adding a wrinkle. We want the `FullName` node to contain a job description followed by the values from the `UserLastName` and `UserFirstName` nodes. Your output should be in the format `<job description><last name>, <first name>`.

We want the `FullName` value to contain either `Customer` or `Supplier` as the job description. The job description value must be derived from the value in the `UserCode` node, which can be either `001` or `002`. The value `001` signifies that the address is for a customer while `002` signifies that the address is for a supplier. And last, you must not output the comma if the `UserFirstName` node has no data. We can do all this using only the Scripting functoid as follows:

1. Drag the Scripting Functoid to the grid.
2. Drag the UserCode to the Scripting functoid as the first input.
3. Drag the UserLastName to the Scripting functoid as the second input.
4. Drag the UserFirstName to the Scripting functoid as the third input.

The result should look like Figure 1-31.

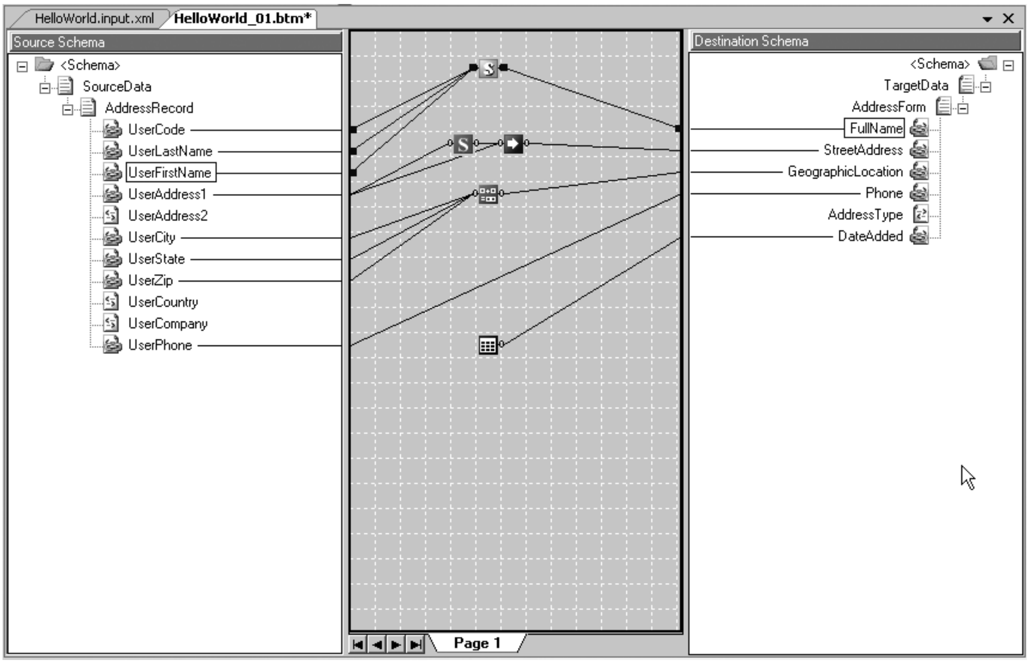


Figure 1-31. *Using the Scripting functoid*

Now, right-click the Scripting functoid, and select Configure Functoid Script to bring up the Configure Functoid Script window shown in Figure 1-32.

■**Note** You can also reach the Configure Functoid Script window by left-clicking the Scripting functoid, going to the Properties window, and clicking the ellipsis next to Configure Functoid Script. Having several paths to choose from is typical of the development environment; it's a feature that allows you to devise a methodology that best fits you.

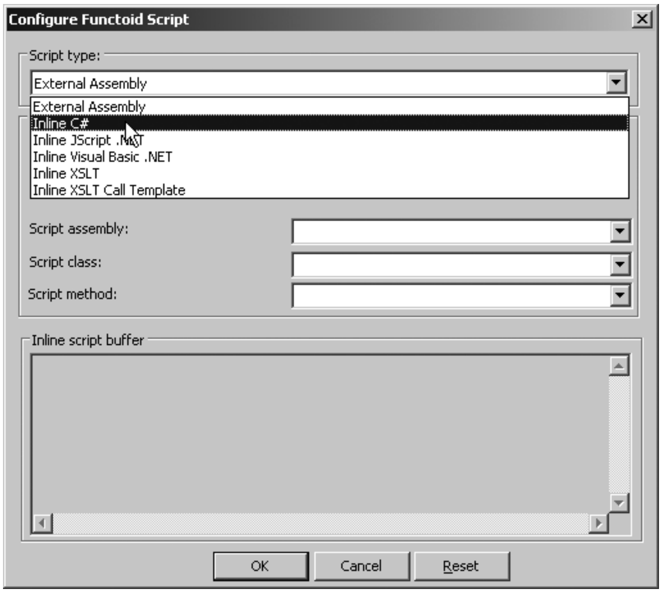
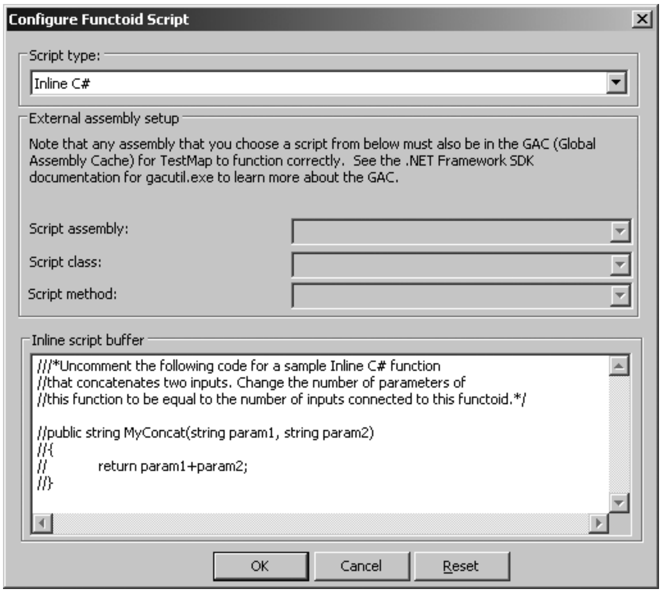


Figure 1-32. *The Configure Functoid Script window*

Go to the "Script type" drop-down box, and select Inline C#. The "Inline script buffer" box will display a sample script, as shown in Figure 1-33.



Note that any assembly that you choose a script from below must also be in the GAC (Global Assembly Cache) for TestMap to function correctly. See the .NET Framework SDK documentation for gacutil.exe to learn more about the GAC.

```
///*Uncomment the following code for a sample Inline C# function
//that concatenates two inputs. Change the number of parameters of
//this function to be equal to the number of inputs connected to this functoid.*/

//public string MyConcat(string param1, string param2)
//{
//          return param1+param2;
//}
```

Figure 1-33. *The Configure Functoid Script window with sample code*

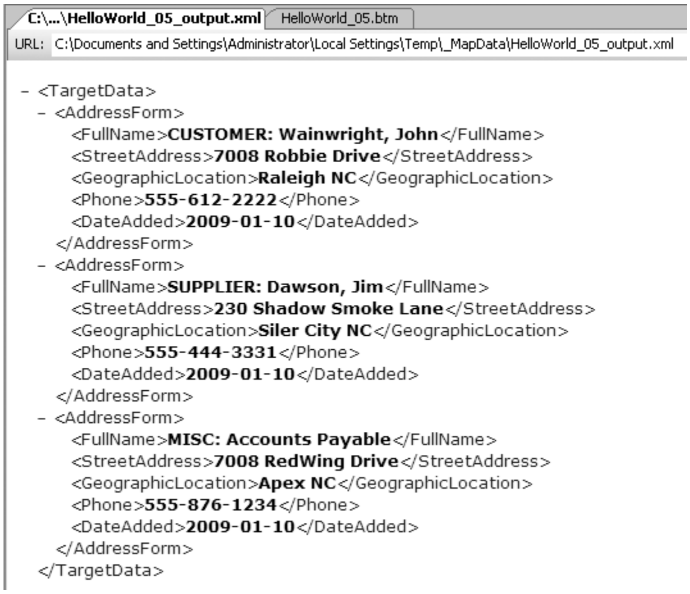
You can enter the script by typing it directly in the "Inline script buffer" box, but we don't recommend that you do. The window cannot be resized, making it difficult to keep track of more than ten or so lines. We create simple scripts in a text-editing window and then copy the script into the functoid window. The script used in this map is shown in Listing 1-2.

Tip If the script has complex logic or is just long, you may want to create and test the script in a development studio project. In this example, the project would be a C# project.

Listing 1-2. *C# Code to Paste into the Configure Functoid Script Window*

```
public string HelloWorld(string inCode, string inLastName, string inFirstName)
{
string retval = "MISC: ";
if (inCode == "001")
    retval = "CUSTOMER: ";
else if (inCode == "002")
    retval = "SUPPLIER: ";
retval = retval + inLastName;
if (inFirstName != "")
    retval = retval + ", " + inFirstName;
return retval;
}
```

The variable `retval` is populated in three steps. First, the code puts `"CUSTOMER: "` or `"SUPPLIER: "` into the variable depending on the value in the input variable `inCode`, or it uses `"MISC: "` if there is no value in the input. The second step appends the value from the input variable `inLastName` to that data. Finally, if the input variable `inFirstName` contains a value, then a comma, a space, and the value from `inFirstName` are appended. The Scripting functoid returns the value in `retval` as output to the map. Figure 1-34 contains the output from the completed map.



URL: C:\Documents and Settings\Administrator\Local Settings\Temp_MapData\HelloWorld_05_output.xml

```
- <TargetData>
  - <AddressForm>
      <FullName>CUSTOMER: Wainwright, John</FullName>
      <StreetAddress>7008 Robbie Drive</StreetAddress>
      <GeographicLocation>Raleigh NC</GeographicLocation>
      <Phone>555-612-2222</Phone>
      <DateAdded>2009-01-10</DateAdded>
    </AddressForm>
  - <AddressForm>
      <FullName>SUPPLIER: Dawson, Jim</FullName>
      <StreetAddress>230 Shadow Smoke Lane</StreetAddress>
      <GeographicLocation>Siler City NC</GeographicLocation>
      <Phone>555-444-3331</Phone>
      <DateAdded>2009-01-10</DateAdded>
    </AddressForm>
  - <AddressForm>
      <FullName>MISC: Accounts Payable</FullName>
      <StreetAddress>7008 RedWing Drive</StreetAddress>
      <GeographicLocation>Apex NC</GeographicLocation>
      <Phone>555-876-1234</Phone>
      <DateAdded>2009-01-10</DateAdded>
    </AddressForm>
  </TargetData>
```

Figure 1-34. *Output from map with Scripting functoid*

Now you have completed your first map. See how easy it is?

Summary

One thing that we learn again and again as we explore the world of BizTalk mapping is that there is *never* only a single way to solve a mapping problem. Quite often, deciding which way is best can be difficult. The map you created in this chapter would look very different if we had chosen other ways for you to implement the steps. When you encounter a mapping issue, use your imagination and look at different approaches to finding a solution. Give John and Jim the same mapping specification and two things can be guaranteed: First, both maps will produce the same output. Second, the maps will not look the same.

This chapter introduced you to the methods and tools that are used to create a map in BizTalk. For many of you, this may have been your first look at Visual Studio and at BizTalk. You cannot create maps in BizTalk without understanding the fundamentals of these environments. This book focuses on mapping and does not discuss those topics in detail, thus you should add books on your version of BizTalk and your version of Visual Studio to your reference library.

In this chapter, we covered the basic steps required to create a map in BizTalk. However, what you saw in the map editor when you completed the work was a graphical depiction of a map, not the map. The map is an XSLT file that is generated when you validate the map. In Chapter 2, we will examine the XSLT that is generated from the work you did in this chapter.