CHAPTER 3

Retrieving and Mapping Data

Data retrieval and EDI document mapping are essential to all EDI implementations. The goal of this chapter is to introduce a logical approach to determining how to define and structure the source and target data, how to retrieve the source data and deliver it to BizTalk Server, how to map the source data to the target trading partner schema, and how to test that the map produces the expected (an EDI text file). The following topics are covered in this chapter:

- **Defining the mapping components**: Mapping includes three primary BizTalk components: the source schema, the target schema, and the map. This section describes how to define these components based on the requirements of the trading partner's EDI document and the structure of the source data.
- **Retrieving the source data**: With the components defined, the source data must be returned to BizTalk Server for processing. This section outlines how to query the source data and return it to BizTalk using the SQL Receive Adapter.
- **Mapping the document**: The actual mapping of data from the source schema to the target schema is done through a combination of analyzing business requirements (determining how fields are related in the source and destination schemas) and implementing the logic to accomplish this (through actual map development). Generally, mapping requirements are outlined in the EDI implementation guide supplied by the trading partner. The section on mapping covers in detail some of the more common mapping tasks for EDI documents.
- **Testing the map**: Throughout the development process, the map will need to be tested. This section outlines a step-by-step approach to testing EDI maps and understanding how to correct errors that may be encountered.

Preparing the Solution Files

The exercises in this chapter assume a general understanding of EDI schemas and trading partner configuration. The first exercise (Exercise 3-1) introduces how to set up the sample code files that accompany this book, for use in later exercises.

Exercise 3-1. Preparing the Solution Files

The exercises in this chapter use the solution files located in Apress.Integration\Chapter 3\Apress.Integration.EDI810.sln. Use the following steps to deploy the solution files to a local machine:

1. Open the solution file in Visual Studio on the local computer. The solution file contains two projects:
   a. **Apress.Integration.EDI810.Common**: This project contains the source data schema and is placed into a separate project so that it can be referenced by any trading partner project.
   b. **Apress.Integration.EDI810.CompanyX**: This project contains the map and target EDI schema specific to the EDI810 implementation for trading partner Company X.
2. Add the database to SQL Server. There are two options for creating the database (this assumes the use of SQL Server 2005).
   a. The database and log file can be attached using the following steps. This is the only option to obtain the full database with all test data populated:
      i. Open SQL Server Management Studio.
      ii. Right-click the Databases folder and select Attach**.**
      iii. In the window that opens, click the Add button and browse to the location of the MDF file. It is located in the folder Apress.Integration\Chapter 3\Database Files\Attach and is called Apress.EDI.Custom.mdf.
      iv. Click OK.
   b. Alternatively, the SQL scripts can be run (no data will be included). There are a total of six scripts, all contained in the folder Apress.Integration\Chapter 3\Database Files\Scripts.

In addition, there are two versions of the target schema and target map for Company X. The original map and schema (as used in Exercise 3-4, later in the "Mapping the Document" section) are included in the project and are in the folder Apress.Integration.EDI810.CompanyX. The modified map and schema (as used in Exercise 3-5) are located in the folder Apress.Integration\Chapter 3\Company X Final Schema and Map.

Finally, there is a sample document located in Apress.Integration\Chapter 3\Test Documents. This is used as a test map input for Exercise 3-5, later in the "Testing the Map" section. The content of this represents what is returned by the stored procedure. The contents can be modified using a text editor.

Defining the Mapping Components

The first step in the process of document mapping is to determine the flow of the document(s) being mapped. This includes defining where the source data is coming from, how it will be retrieved and delivered to BizTalk, what the format of the data will be, and what the target schema (or schemas, in the case of multiple trading partners) will be. Figure 3-1 gives a high-level overview of the document flow that is covered in detail in this chapter.

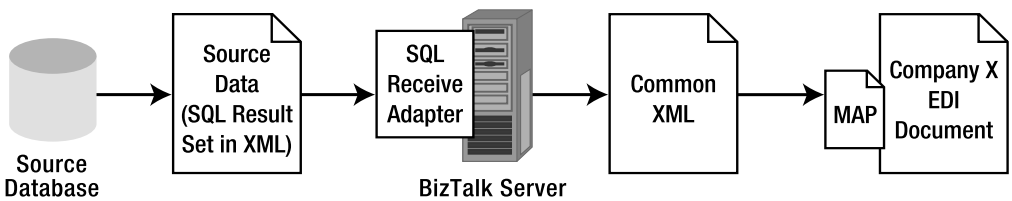

Figure 3-1. *Data mapping flow*

Defining the Target Schema

Schema creation and definition are covered in Chapter 1. At this point, it is assumed that the process for selecting a schema is understood and that the trading partner's requirements have been assessed for whichever type of EDI document is to be delivered. For the sample implementation outlined in this chapter, a modified version of the X12_00401_810.xsd (which represents an invoice) will be used. Many nodes in the schema that are not needed by the trading partner have been removed. This not only simplifies the document for illustration purposes but is good practice when preparing a schema for an actual implementation.

■ **Note** It is important to understand that the data being delivered to trading partners will frequently not adhere to the standard version of the EDI schema provided with BizTalk Server 2006 R2. It is a necessary step to make modifications to the schema where necessary. For instance, a trading partner may be using a specific segment to contain data that does not commonly belong there. The schemas should be modified on a per-partner basis, and should generally include only those nodes that the trading partner is interested in. Always refer to the EDI implementation guide when working with mapping requirements.

Company X represents the trading partner to which EDI documents will be delivered. The sample EDI document being used for demonstration purposes has been greatly simplified but represents a valid version of an X12 810 invoice document. The document shown in Listing 3-1 is a sample instance of the final EDI document that will be delivered to the trading partner (Company X) after each of the components in this chapter is put into place.

Listing 3-1. *Sample 810 EDI Document for Company X*

```
ST*810*1000~
N1*ST*Company Y~
N4*City R*State B*23456~
IT1*1*2**12~
IT1*2*2**10~
CTT*2*44~
SE*7*1000~
```

■ **Note** For X12 documents, the ISA, GS, GE, and IEA segments are never included in the map; these are configured through the BizTalk Administration Console and are automatically generated on outgoing documents. The same holds true for the UNA, UNB, UNG, and UNZ segments for EDIFACT documents. Configuring these segments is covered in Chapter 2.

It will prove to be helpful to define the target document (an actual instance of what is to be delivered to a trading partner) prior to creating or modifying the target schema. Working with the trading partner to determine the format and content of the document is essential in the early stages of requirements gathering and will eliminate a great deal of work during the development stage. Generally, trading partners will have recorded the specifications of EDI documents in an EDI implementation guide. Company X will use a modified version of the standard 810 schema, as shown in Figure 3-2.

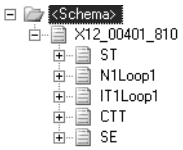

Figure 3-2. *Trading partner target schema (modified from standard schema)*

Defining the Source Data and Schema

The *source data* is defined as all data being extracted from the source data store. It includes all data needed by any trading partner, such as billing and shipping address data, details about the contents of a message (e.g., in the case of 810 documents, invoicing information), and other pertinent information. It also includes all data that may be used during processing by BizTalk Server (such as IDs for tracking) and any other data that may be extracted—used or unused. This data may come from multiple sources, or it may come from a single source. For successful mapping, it is essential to define what the source data is, where it is coming from, and what the delivery format will be.

For the purposes of this section, the source data will come from a single data store (SQL Server 2005) and from a sample database that is intended to mimic a fractional subset of information housed in an accounting application. From this data, invoices will be generated, resulting in 810 EDI documents.

Generally, defining the source data and source schema go hand in hand; the schema is often a "work in progress," evolving throughout the course of the project as requirements are defined and additional data is needed for calculations and EDI document generation. In some rare instances, all of the needed data is known at the start of the project, and the complexities of defining where the data comes from are greatly diminished. In either case, whether the data is fully defined or whether it is being defined in parallel with development, implementing the structure of the data and encapsulating it into a source schema is a task that requires a great deal of thought to ensure that the long-term requirements of the trading partners are met.

The creation of the source schema begins with determining how best to structure the data. This is most easily accomplished by assessing how the data is set up in the source data store (i.e., in the case of databases, how the tables and the data within them are related), and from that, setting up the relational hierarchy that will be most easily constructed and matches, to some degree, the target EDI schema (to simplify mapping, if possible). In the case of the sample implementation, the source database is structured as defined in Figure 3-3.

With the data model of the source database known, and the target EDI document schema defined, the next step is to determine which of the fields are of interest to the external systems (including BizTalk and its components: maps, orchestrations, ports, etc.) and trading partners. The source data store will always contain proprietary information used within the data store itself (such as unique identifiers and metadata); this data is generally not needed by external systems and should not be included in the source schema.

For the sample implementation, the proprietary information includes most of the GUIDs (e.g., uidAddressID) which are primary keys within the database but are not generally useful for most purposes outside of the database. However, for this discussion one of the GUIDs will be included in the source schema (uidInvoiceID). It is intended to be used for tracking purposes within BizTalk Server. No trading partner is typically interested in this type of data, but it often makes sense to include it to be able to track the document through BizTalk.

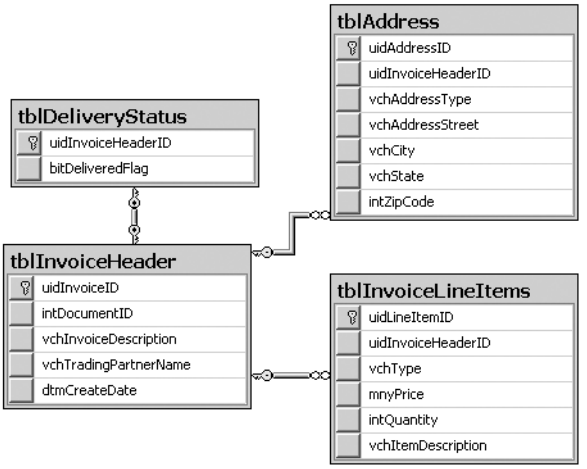

Figure 3-3. *Source database diagram*

From understanding the database diagram (what data is available) and knowing the target trading partner data requirements (what data is needed), it can be determined that the source document schema should be as shown in Figure 3-4.

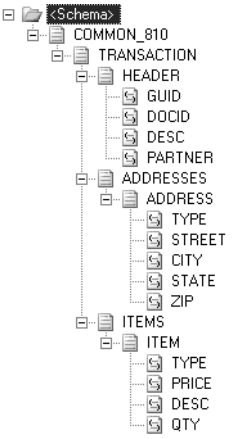

Figure 3-4. *Common source schema*

Retrieving the Source Data

This section outlines an approach to retrieving the data from the source data store, which will simplify schema development and the delivery of data to BizTalk Server. As with all integration projects, there are a large number of potential options in retrieving data that depend on a variety of factors:

- **Where the data is stored**: Different data stores have different methods for accessing the data they contain. This may be via stored procedures, web services, a custom API, flat files, or a number of other objects.
- **Amount of data**: The amount of information in a transaction, whether data will be batched or handled as individual transactions, and the size of data transmissions can have an impact on how the data will be retrieved (or delivered).
- **Frequency**: The frequency that data will be transmitted to BizTalk Server and whether this data will be delivered asynchronously or synchronously (real time) will help determine how best to implement the retrieval of source data.
- **Acknowledgements**: If two-way communication needs to occur between the source system and BizTalk Server (i.e., acknowledgement that delivery is successful/failed), it may make sense to implement one type of data retrieval process vs. another.
- **Network accessibility**: Occasionally the data store may be on a portion of the network that is not directly accessible to BizTalk Server. Security restrictions may require different approaches to retrieving data from the source.

For the purpose of discussion and demonstration, this section outlines the most common form of data retrieval from a data source housing EDI data: the retrieval of data via the BizTalk SQL Receive Adapter. From this process, more complex patterns can be implemented to suit the needs of any EDI implementation.The database component outlined here is the stored procedure, which returns the result set as an XML document. Rendering result sets in XML within the stored procedure (rather than trying to work with schemas on the SQL Receive Adapter to format the result sets after receiving the data) simplifies the data retrieval process, making the delivery of the data to BizTalk Server as seamless as possible. Figure 3-5 illustrates the flow and components used in retrieving data in XML, and shows that XML can be returned directly to the SQL Receive Adapter from the stored procedure.

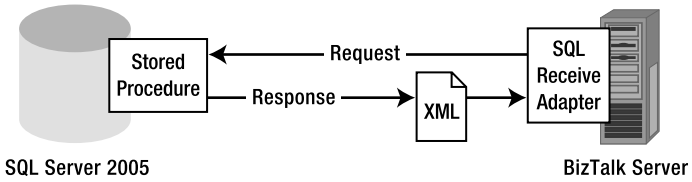

Figure 3-5. *Retrieving data as XML*

SQL Server 2005 has full support of XML and is readily available in the majority of BizTalk 2006 Server deployments (generally, BizTalk 2006 is deployed on SQL Server 2005; SQL 2000 has limited XML support, but can be used in a similar fashion to this implementation, if needed). A number of enterprise-level databases support some level of XML retrieval, and the approach outlined in this section can be used to process data directly from these systems. In many cases, the simplest approach to retrieving data from any external data source is to use linked servers from SQL Server 2005, which allows full flexibility in the stored procedure (on SQL 2005) to process data on external databases.

Creating the Stored Procedure

The steps to implementing the flow shown in Figure 3-5 begin with creating the stored procedure that will return a result set as an XML document. The function of the stored procedure is to return all of the fields needed in the creation of the target EDI schema. The stored procedure outlined in Listing 3-2 retrieves an XML document that matches the source schema defined earlier (see Figure 3-4). A sample instance of the XML document that will be generated by the stored procedure is shown in Figure 3-6.

■ **Note** When retrieving data using the BizTalk SQL Receive Adapter, a root node will always wrap the XML document retrieved from the database. In this case, the root node in the schema is <COMMON_810>, while the root node retrieved by the stored procedure is <TRANSACTION>. The <COMMON_810> node will be automatically added by the SQL Adapter.

Listing 3-2. *Stored Procedure to Return XML Document*

```
CREATE PROCEDURE [dbo].[spGetInvoice]
AS
BEGIN
 SET NOCOUNT ON;

 -- select invoice to be extracted
 DECLARE @uidInvoiceID As uniqueidentifier
  ,@intCounter As int

 -- get the first document returned that has not already
 -- been delivered.  Remaining documents will be processed
 -- on subsequent calls to this stored procedure.
 SET @uidInvoiceID =
  (SELECT TOP 1 uidInvoiceID
   FROM tblInvoiceHeader
   WHERE uidInvoiceID NOT IN
   (SELECT uidInvoiceHeaderID
    FROM tblDeliveryStatus
    WHERE bitDeliveredFlag = 1
   )
   ORDER BY dtmCreateDate
  )

 -- Only execute the code if an ID is present
 IF @uidInvoiceID IS NOT NULL
  BEGIN
   -- wrapping in a tran can be useful with multiple updates
   -- shown here for demonstration purposes only
   BEGIN TRAN
    BEGIN TRY
     -- generate XML document as result set
     -- note that BizTalk will add its own root node when
     -- extracted using the SQL Receive Adapter
```

```
    -- the DOCID is unique and can be used to trace
    -- document throughout entire process
    SELECT NULL
    ,(SELECT uidInvoiceID AS "GUID"
      ,intDocumentID AS "DOCID"
      ,vchInvoiceDescription AS "DESC"
      ,vchTradingPartnerName AS "PARTNER"
      FROM tblInvoiceHeader
      WHERE uidInvoiceID = @uidInvoiceID
      FOR XML PATH('HEADER'), BINARY BASE64, TYPE)
      ,(SELECT NULL
       ,(SELECT vchAddressType AS "TYPE"
                      ,vchAddressStreet AS "STREET"
                      ,vchCity AS "CITY"
                      ,vchState AS "STATE"
                      ,intZipCode AS "ZIP"
         FROM tblAddress
         WHERE uidInvoiceHeaderID = @uidInvoiceID
         FOR XML PATH('ADDRESS'), BINARY BASE64, TYPE)
        FOR XML PATH('ADDRESSES'), BINARY BASE64, TYPE)
        ,(SELECT NULL
         ,(SELECT vchType AS "TYPE"
                        ,mnyPrice AS "PRICE"
                        ,vchItemDescription AS "DESC"
                        ,intQuantity AS "QTY"
           FROM tblInvoiceLineItems
           WHERE uidInvoiceHeaderID = @uidInvoiceID
           FOR XML PATH('ITEM'), BINARY BASE64, TYPE)
          FOR XML PATH('ITEMS'), BINARY BASE64, TYPE)
         FOR XML PATH('TRANSACTION'), BINARY BASE64

    -- Now set the delivered flag in the table to "true"
    UPDATE tblDeliveryStatus
    SET bitDeliveredFlag = 1
    WHERE uidInvoiceHeaderID = @uidInvoiceID

   END TRY
   BEGIN CATCH
    -- error occurred, rollback
    ROLLBACK TRAN
    RETURN @@ERROR
   END CATCH
  -- success
  COMMIT TRAN
 END
END
```

```
- <TRANSACTION>
  - <HEADER>
      <GUID>4FA0B48B-ACCB-40F0-B1D2-7E22FFF820BA</GUID>
      <DOCID>2000</DOCID>
      <DESC>Sample Invoice for Company X </DESC>
      <PARTNER>Company X </PARTNER>
    </HEADER>
  - <ADDRESSES>
    - <ADDRESS>
        <TYPE >Billing</TYPE >
        <STREET >1234 A Road </STREET >
        <CITY>City G </CITY>
        <STATE >State A </STATE >
        <ZIP>12345 </ZIP>
      </ADDRESS>
    - <ADDRESS>
        <TYPE >Shipping</TYPE >
        <STREET >6789 F Road </STREET >
        <CITY>City G </CITY>
        <STATE >State A </STATE >
        <ZIP>12345 </ZIP>
      </ADDRESS>
    </ADDRESSES>
  - <ITEMS >
    - <ITEM >
        <TYPE >Standard </TYPE >
        <PRICE>100.0000 </PRICE>
        <QTY >1</QTY >
        <DESC>Item A </DESC>
      </ITEM >
    - <ITEM >
        <TYPE >Hidden </TYPE >
        <PRICE>0.0000 </PRICE>
        <QTY >3</QTY >
        <DESC>Service Charge </DESC>
      </ITEM >
    </ITEMS >
  </TRANSACTION>
```

Figure 3-6. *Sample XML result set generated by stored procedure*

Configuring the BizTalk SQL Receive Adapter

Once the stored procedure is created, it is necessary to set up a process that will call it and return the XML result to BizTalk Server for processing. The component that most easily accomplishes this is the BizTalk SQL Receive Adapter. It can be configured to run on a timed interval and will continue to call the stored procedure until all available invoices have been returned (the stored procedure shown in Listing 3-2 returns a single document each time it is called). The steps outlined in Exercise 3-2 provide all of the information needed to configure the SQL Adapter to return the result set in the format that matches the source data schema shown previously in Figure 3-4.

Exercise 3-2. Configuring the BizTalk SQL Receive Adapter

To create a BizTalk 2006 SQL Receive Adapter that is configured to execute a stored procedure on a timed interval and deliver the resulting XML to the BizTalk MessageBox, take the following steps:

1. Open BizTalk Administration Console and expand the EDI.Demonstration.Chapter3 application (or the application that contains the BizTalk components). Right-click Receive Ports and select New ➤ One-Way Receive Port.
2. On the General tab, name the port appropriately (the sample implementation names this EDI.Demonstration.SQL.Receive.810).

3. Click the Receive Locations tab. In the right-hand window, click New to create a receive location. Name the receive location appropriately (in this case, SQLReceive). The name will distinguish it from other receive locations that may be associated with this port.
4. Next to Type, select SQL. Click the Configure button. Figure 3-7 shows the window with each of the properties set in the following steps.

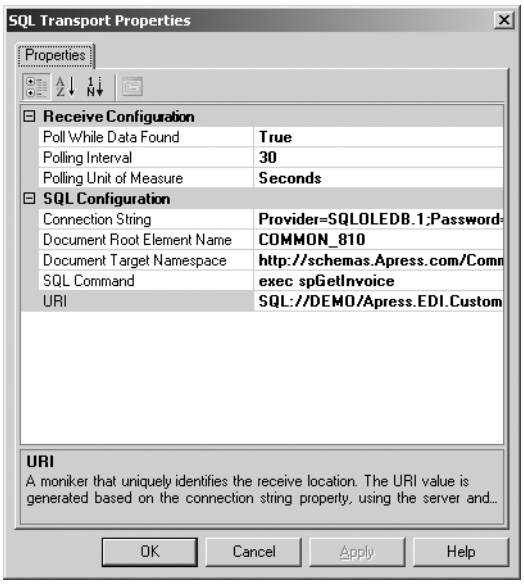

Figure 3-7. *Configuring the SQL receive location properties*

a. **Poll While Data Found**: This indicates whether the adapter will continue to call the stored procedure until no results are returned, or whether it will only be called once per execution period. For this solution, set this value to True. Each time the stored procedure is called, only one 810 document will be returned. By setting this value to True, the procedure will continue to execute until all 810 documents have been returned to BizTalk.

■ **Note** When using the Poll While Data Found property, it should be understood that if a SQL Receive Adapter is running on a BizTalk group, all of the BizTalk servers within the group may end up calling the stored procedure simultaneously. Depending on how the stored procedure is written, this may pose a problem. For instance, in the procedure called in this exercise, a flag is set indicating that a row has been delivered after the data has been returned. If the stored procedure is to be called simultaneously by multiple servers in a BizTalk Group, the result set may be returned more than once due to the fact that the flag will not be updated in time to prevent the second call from retrieving the same record. In a case such as this, the Poll While Data Found would need to be set to False, or a more robust way of determining whether data has already been retrieved would need to be added to the stored procedure.

---

b. **Polling Interval**: This setting indicates the amount of time to wait between executions of the stored procedure. The time will depend on the number of documents being returned and the frequency that these documents become available. For this demonstration, set the value to 60.
c. **Polling Unit of Measure**: Units are Hours, Minutes, or Seconds. Set this to Seconds.

d. **Connection String**: This is the connection string to the database where the stored procedure exists. Click the ellipsis to configure this setting. Once the values have been set, click the Test Settings button to ensure that the values are valid. Note that allowing the password to be saved will ensure that when exporting to an MSI file, the password will be stored. If it has not been saved, it will have to be set when imported to a new environment.

e. **Document Root Element Name**: This is the root element of the schema that the document being retrieved will adhere to. In this case, this will be set to COMMON_810. The root node of the document being retrieved by the stored procedure (<TRANSACTION>) will be wrapped by this node.

f. **Document Target Namespace**: This value must match the target namespace of the schema that the document being retrieved is set to. To find this value, open the schema in Visual Studio and find the Target Namespace property. For this solution, set the value to http://schemas.Apress.com/Common/810/v1.

g. **SQL Command**: When calling a stored procedure that returns a formatted XML document, all that needs to be set for this property is the SQL command to execute the stored procedure. In this case, set the value to exec spGetInvoice.

h. **URI**: The URI identifies this document from other documents. This must be a unique value and can generally be set to the default value that the receive location will be set to automatically. Even though this is set automatically, ensure that this value is unique when there is more than a single SQL receive location. For this solution, this property should be set to SQL://DEMO/Apress.EDI.Custom/Invoice. Note that each time this property comes into view (when the window is opened) BizTalk may automatically reset it to the default value. Make sure each time the SQL Transport properties are viewed (or modified), this value is set to the correct value.

5. Back on the main receive location configuration screen, set the Receive Pipeline property to XMLReceive.

6. On the Schedule tab, configure the receive location to execute during the appropriate hours. For example, there are frequently database backups that run during the early morning hours, networks are occasionally offline for maintenance, and so on. It may be appropriate to force the processing of documents to occur only during business hours. For this solution, the receive location will always be turned on (no modifications to the schedule are necessary).

7. Click OK. All values have been configured to call the stored procedure and return an XML document. Later chapters will discuss exception handling and tracking options.

Implementing Linked Servers

Generally speaking, data is stored in an enterprise-level database, which may or may not be SQL Server 2005. In cases where this is not SQL 2005, it is possible to use the same model discussed in the previous section, by implementing linked servers. Exercise 3-3 outlines the modifications that need to be made to the stored procedure shown in Listing 3-2 to allow it to interact with a linked server.

There are some limitations to linked servers that should be looked at prior to implementing them in a solution—most importantly, the increased demand on the network for the traffic generated between the database that houses the stored procedure and the database that is being linked to. For many EDI implementations, documents are retrieved from source systems on timed intervals (perhaps once a day, hourly, etc.), and the demand on the network is negligible; in such cases, linked servers are an ideal solution. In other cases, where large numbers of documents may be returned on a very frequent basis, it will likely make more sense to develop an interface directly between BizTalk Server and the source data store (such as through the use of a custom adapter). For purposes of discussion here, it is assumed linked servers will fulfill the requirements. Figure 3-8 depicts the components used when retrieving XML data from a linked server.

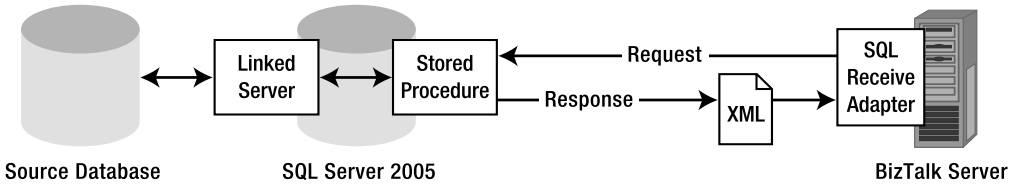

Figure 3-8. *Retrieving data using XML and linked servers*

Exercise 3-3. Using a Linked Server

The following steps indicate how to add a linked server to SQL Server 2005 and how to modify the stored procedure created earlier in this chapter (see Listing 3-2) to interact with this linked server (instead of with tables on the same database as the stored procedure):

1. Open SQL Server Management Studio. Expand Server Objects ➤ Linked Servers.
2. Right-click Linked Servers and select New Linked Server.
3. Name the linked server (this is how it will be referred to in the stored procedure) and configure all of the details about how to connect to it on the General tab. For this exercise, the name will be TestLinkedServer.
4. On the Security tab, indicate how to connect to the target system. This should be set to use the same BizTalk account as is configured to run the SQL Receive Adapter; when the stored procedure executes, it will run under the context of this account. In turn, the linked server connection will impersonate this account. The account that runs the SQL Receive Adapter is generally the BizTalkServerApplication, about which information can be obtained in the BizTalk Administration Console under Platform Settings ➤ Host Instances. Right-click the BizTalkServerApplication account and select Properties to see how this account has been configured.

   Once the linked server has been set up and verified, the stored procedure must be modified to be able to interact with it. This is a very simple modification, consisting primarily in changing the table names to the full linked server table name. Take the following steps:

   a. Add a new command at the top of the stored procedure as follows. This will ensure that connections to the linked server are handled properly:

      ```
      BEGIN -- this is existing in the original
       SET XACT_ABORT ON -- this is new for linked server
       SET NOCOUNT ON; -- this is existing in the original
      ```

   b. Change the names of the tables that stored the data to the full name of the linked server for all SELECT and UPDATE statements. For example, if the original select statement is

      ```
      SELECT vchType As "TYPE"
      FROM tblInvoiceLineItems
      ```

   For the linked server it would be

      ```
      SELECT vchType AS "TYPE"
      FROM TestLinkedServer.[database].[table name]

      -- or --

      SELECT vchType AS "TYPE"
      FROM TestLinkedServer.[instance].[database].[table name]
      ```

   c. Change any of the field names as appropriate for all UPDATE and SELECT statements.

Mapping the Document

Once the source and target schemas have been defined, and it is determined how the data will be retrieved, the actual mapping can take place. Mapping requirements can be simple or complex, depending on the requirements outlined in the implementation guide (or defined by the trading partner). In some cases, it may be as easy as dragging the source element and dropping it on the target element. In other cases, it may be more involved, requiring custom functoids, external .NET assemblies, and lookup tables. In any case, simple or complex, some amount of logic will need to be put into place to ensure that the document is mapped correctly.

This section outlines some common tasks that may be encountered during the mapping of EDI documents. Exercise 3-4 walks step-by-step through the creation of the map, including the configuration and placement of functoids. Before working through the exercise, however, it is important to look at several items. When mapping EDI documents, keep the following in mind:

- **Target node**: Information about the target element can be obtained from one of the properties on the target schema: the Notes property (see Figure 3-9). This property can be useful when trying to determine what different elements represent, as it describes the expected content of the node.

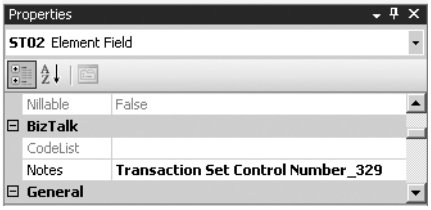

Figure 3-9. *The Notes property on the EDI schema*

- **Functoid input parameter order**: When configuring any functoid, ensure that the input fields are in the correct order. For example, when using a value mapping functoid, the first parameter must always return a boolean value, while the second parameter should always represent the value to be mapped when the boolean value is True.
- **Transaction ID**: The ST segment (or the UNH segment for EDIFACT documents) contains one element that can either be mapped or configured in the trading partner EDI settings. This element is ST02 (or UNH1 for EDIFACT), and it represents the unique identifier of the document (known as the *transaction set control/reference number*). For tracking purposes, it is very useful to take a unique document identifier from the source system and map it to this field; this way, the document can be tracked from the source system, through BizTalk Server, to the document that is delivered to the trading partner. However, in some cases it may make more sense to have BizTalk override this with a unique ID that is automatically generated by the system. The first approach is done through the use of a map, while the second can be set using BizTalk Administration Console in the EDI properties for a party using the GS and ST segment definition (or UNG and UNH segment definition for EDIFACT).

- **Conditional incremental counters**: The IT101 and CTT01 elements generally contain incremental counters indicating line item counts. These counters can be complex to calculate when there are line items in the source document that are not mapped, and therefore not counted. In simple cases, when all line items are mapped from the source document to the target document, an iteration functoid or record count functoid could be used to set these values. But in cases where there are unmapped line items, a global variable in the map is useful. A global variable allows a counter to be set and incremented for all items that are mapped, ignoring unmapped values. Listings 3-3 and 3-4 are used in the functoids included in the calculation of the IT101 and CTT01 elements.

The code in Listing 3-3 shows how to set and increment a global variable in a map each time the script functoid that the code is contained within is executed. In Exercise 3-4, this value is used in the IT101 element for each line item that is mapped.

Listing 3-3. *Iteration Counter Using a Global Variable Within a Map*

```
System.Collections.ArrayList _globalArrayList = new System.Collections.ArrayList();

public int intIterator(bool blnInput)
{
 int intRetVal = 0;
 if(blnInput == true)
 {
  // each time add 1 to the array
  int intCount = 1;
  globalArrayList.Add(intCount);

  for (int intI=0; intI<_globalArrayList.Count; intI++)
  {
   intRetVal += (int)_globalArrayList[intI];
  }
 }
 return intRetVal;
}
```

With the global variable (globalArrayList) incrementing each time a line item is mapped to the target schema, a running total is kept. Once all line items have been mapped, the final value can be reused to populate the CTT01 element, which represents the total number of line items mapped in the EDI document. Listing 3-4 shows the code used to access the count set in Listing 3-3. The full implementation of this code in a functoid within a map is outlined in Exercise 3-4.

Listing 3-4. *Accessing Total Iteration Counter*

```
public int intAccessTotal()
{
 int intRetVal = 0;
 for (int intI=0; intI<_globalArrayList.Count; intI++)
 {
  intRetVal += (int)_globalArrayList[intI];
 }
 return intRetVal;
}
```

Exercise 3-4. Creating an EDI Map

This exercise demonstrates how to add a new or existing map to a Visual Studio BizTalk 2006 project and create the mapping nodes needed for a sample EDI 810 (invoice) document. It is based on the EDI implementation guides shown in the figures throughout this exercise:

1. Open the Visual Studio solution. For this exercise, open Apress.Integration.EDI810.sln.
2. To add a new or existing map, right-click the project to which the map will be added. In this example, right-click Apress.Integration.EDI810.CompanyX.
   a. To add a new map, select Add ➤ New Item. In the window that opens, click Map Files and select Map from the right-hand window. This will add a new map to the project. In this case, create a new map named CommonXMLToCompanyX810.btm.
   b. To add an existing map, select Add ➤ Existing Item. Browse to the location of the map, select it, and click Add. This will add an existing map. The rest of this exercise assumes that the map does not already exist.
3. Once the new map is added, make sure that it is open in Visual Studio. Two links will appear: a link for the source schema, and a link for the target (or destination) schema.
   a. To add the source schema, click Open Source Schema. In the BizTalk Type Picker window that opens, select the document that will be the source schema. Because this solution uses a separate project to store the source schema, it appears under References. Expand References ➤ Apress.Integration.EDI810.CompanyX ➤ Schemas and select Apress.Integration.EDI810.Common.CommonXML. This will set the source schema for the map.
   b. To add the target schema, click Open Destination Schema. In the BizTalk Type Picker, select Schemas ➤ Apress.Integration.EDI810.CompanyX. X12_00401_810_CompanyX. This sets the target schema for the map.
4. Map the fields according to the trading partner requirements, as shown in these steps:
   a. All maps are created with a single tab. To aid in organizing a complex map, rename this tab to ST. Additional tabs will be created for different EDI document segments. The ST segment is known as the *transaction set header*. The map for the ST segment is shown in Figure 3-11 and is based on the requirements outlined in the implementation guide shown in Figure 3-10.

Ref	Data Element Summary / Mapping Details	Length
ST01	Transaction Set Identifier Code	3/3
	Code uniquely identifying a Transaction Set	
	Set to 810 Invoice	
ST02	Transaction Set Control Number	4/9
	Identifying control number - must be unique within functional group.	

Figure 3-10. *Mapping the ST segment*

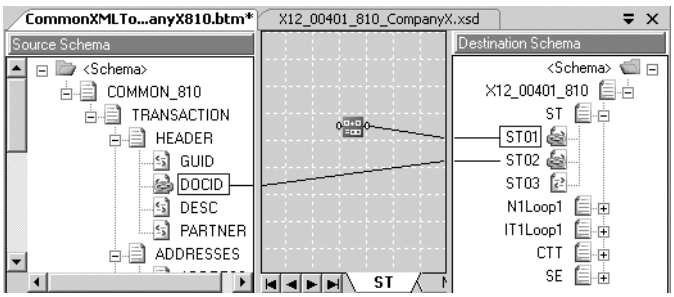

Figure 3-11. *ST segment (mapping details)*

i. **ST01**: Drag and drop a Concatenate functoid onto the map surface. Double-click the shape and set it to 810. Connect it to the ST01 node in the target schema. This segment indicates the Transaction Set Identifier Code, which in this case is the same as the document type.

ii. **ST02**: Map the DOCID element from the source schema to the ST02 element on the target schema. This is the unique document ID (transaction set control number) that can be used to identify a document in the source system, throughout BizTalk Server components, and with the trading partner once the document has been delivered.

b. Create a new tab on the map for the N1 loop. Right-click the ST tab (at the bottom of the map surface) and select Add Page. Name the new page N1 loop. The N1 loop contains segments with name, address, and other company contact information. Refer to Figure 3-12 and Figure 3-13 for the mapping specifications. The final map is shown in Figure 3-14.

Ref	Data Element Summary / Mapping Details	Length
N101	Entity Identifier Code	2/3
	Identify organization, physical location, property, or individual	
	ST or BT	
N102	Name	1/60
	Free form name of entity.	

Figure 3-12. *N1 segment (mapping details)*

Ref	Data Element Summary / Mapping Details	Length
N401	City Name	2/30
	Free form string representing city	
N402	State or Province Code	2/2
	Code defined by government agency.	
N403	Postal Code	3/15
	Exclude dashes and blanks. Example: 98765	

Figure 3-13. *N4 segment (mapping details)*

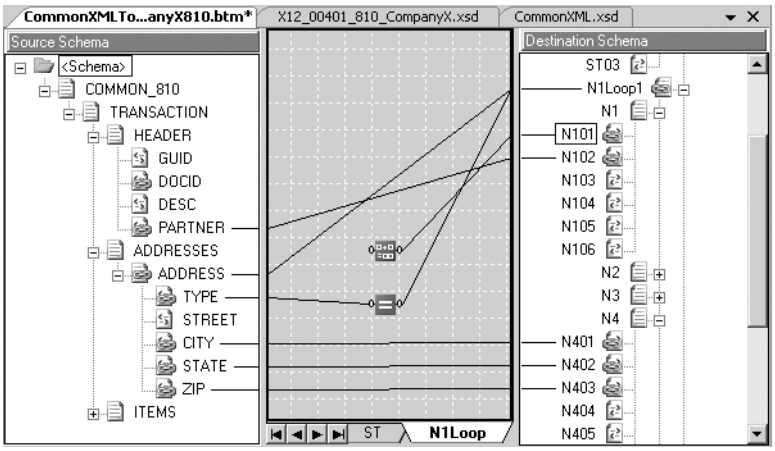

Figure 3-14. *Mapping the N1 loop segments*

To map the N1 loop segments, follow these steps:

i. **N1 loop**: In this demonstration, the source schema may contain multiple ADDRESS nodes. The only address that should be used for mapping the N1 node is the address with a Type = "Billing". The following steps ensure that no subnode to the N1 loop will be mapped in the target element unless the address type is appropriate:

   1. Drag and drop an Equals functoid onto the mapping surface.
   2. Double-click the Equals functoid and create a constant value of Billing.
   3. Drag the TYPE element from the source ADDRESS node to the Equals functoid.
   4. Drag the output of the Equals functoid to the N1Loop1 node on the target schema.
   5. Map the ADDRESS node from the source to the N1Loop1 node in the target.

ii. **N101**: Drop a Concatenate functoid on the map surface. Set the value of this functoid to ST. Drag the output of this functoid to the N101 node in the target schema. This represents the entity identifier code.

iii. **N102**: Drag the source schema element PARTNER to the target schema node N102. This represents the name of the trading partner.

iv. **N401**: Drag the source element CITY to the target N401 element. This represents the city name.

v. **N402**: Drag the source element STATE to the target N402 element. This represents the state name.

vi. **N403**: Drag the source element ZIP to the target N403 element. This represents the postal code.

c. Create a new tab on the map for the IT1 loop and CTT segment. Because the data in both is closely related, some of the functoids will be used to generate data for multiple elements. Right-click one of the page tabs at the bottom of the map and select Add Page. Name the page IT1Loop/CTT. The IT1 loop represents line item information, while the CTT loop contains information about the total transaction (summed from line items). The specification for the IT1 loop is shown in Figure 3-15. The CTT loop will be covered in later steps in this exercise.

Ref	Data Element Summary / Mapping Details	Length
IT101	Assigned Identification	1/20
	This is a line item counter - numeric	
IT102	Quantity Invoiced	1/10
	Numeric value	
IT104	Unit Price	1/14
	Two decimal places, numeric	

Figure 3-15. *IT1 segment (mapping details)*

i. **IT1 loop**: A line item will only be mapped if TYPE = "Standard". The following steps will ensure that only valid line items are mapped:

   1. Drop a Scripting functoid on the mapping surface.
   2. Right-click the functoid and select Configure Functoid Script.
   3. Select Inline C# for the Script Type property.
   4. Enter the code in Listing 3-5 into the script window (as shown in Figure 3-16).

Listing 3-5. *Item Mapping Functoid*

```
// if ITEM is Hidden, it will not be
// mapped or counted in the Iteration total

public string strItemType (string strTYPE)
{
 string strRetVal = string.Empty;
 switch (strTYPE) {
 case "Hidden":
  strRetVal = string.Empty;
  break;
 default:
  strRetVal = "Map";
  break;
 return strRetVal;
}
```

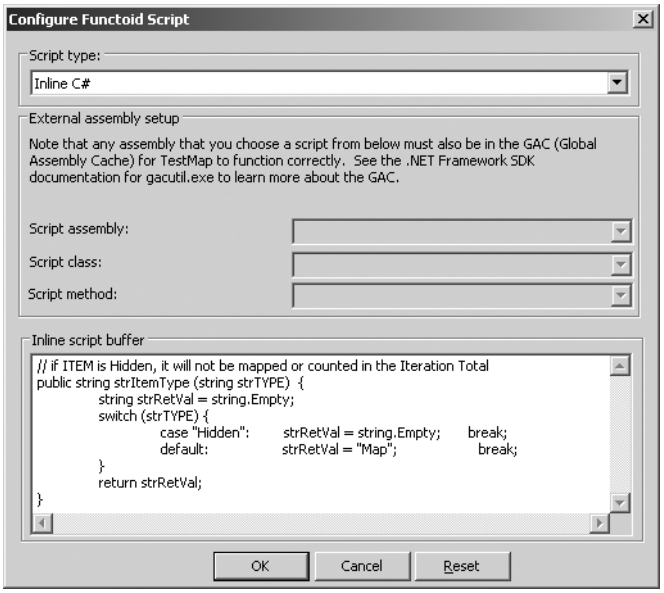

Figure 3-16. *Inline script for item type (scripting functoid)*

5. Click OK to save the code for the scripting functoid.
6. Drop an Equals functoid on the mapping surface, to the right of the Scripting functoid.
7. Right-click the Equals functoid and set a constant value of Map.
8. Drag the output of the Scripting functoid and drop it on the Equals functoid.
9. Drag the output of the Equals functoid and drop it on the It1Loop1 node in the target.

ii. **IT101**: This element represents the identifier for the line item. In this case, it will be a sequential number that begins at 1 and only increments for line items that are of a valid ("Standard") type. Figure 3-17 shows the mapping for the IT1 loop.

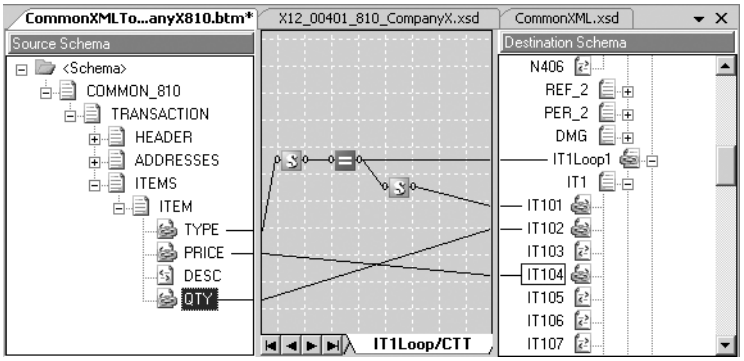

Figure 3-17. *Mapping the IT1 loop segment*

1. Drop a Scripting functoid on the mapping surface to the right of the Equals functoid.
2. Enter the inline C# code in Listing 3-3 for the scripting functoid. This will ensure that the line item is only incremented for valid line items.
3. Make the input for the Scripting functoid the output of the equals functoid.
4. Map the output of the Scripting functoid to the target schema element IT101.

iii. **IT102**: Drag the QTY element from the source document and drop it on the IT102 element in the target schema. This represents the quantity invoiced.

iv. **IT104**: Drag the PRICE element from the source document and drop it on the IT104 element in the target schema. This element is the unit price.

Using the same tab, the CTT mapping will now be added. Refer to Figure 3-18 for the implementation.

Ref	Data Element Summary / Mapping Details	Length
CTT01	Number of Line Items	1/6
	Sum of line items in IT1 Loop	
CTT02	Total value invoiced	1/20
	Sum of all line item prices multiplied by the quantity of that item.	

Figure 3-18. *CTT segment (mapping details)*

v. **CTT01**: This represents the total number of line items that are mapped in the IT1 loop. This must be a total of only valid line items (i.e., Type = "Standard"). Drop a Scripting functoid on the mapping surface and enter the inline C# code shown in Listing 3-4.

vi. **CTT02**: This represents a total of all line items. Some line items may indicate more than a single item as the quantity; therefore quantity and price are a factor in calculating this field. The completed map for the CTT01 and CTT02 segments is shown in Figure 3-19.

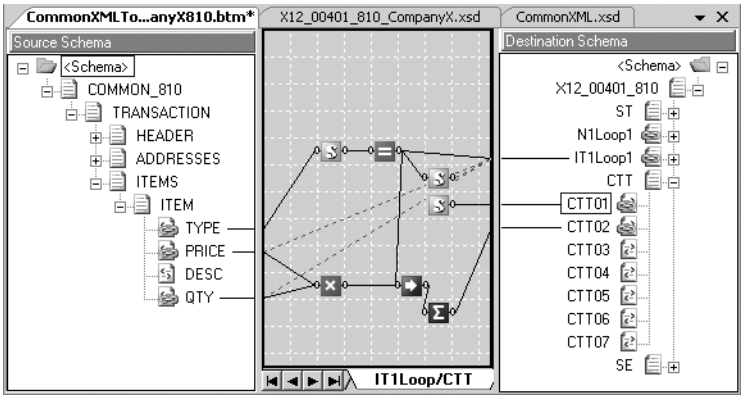

Figure 3-19. *Mapping the CTT segment (after IT1 loop mapping)*

1. Drop a Multiplication functoid on the mapping surface. Drag and drop two inputs to this functoid (in any order) from the source schema: the PRICE and QTY elements.
2. Drop a Value Mapping functoid on the map surface, to the right of the Multiplication functoid (and to the right of the Equals functoid created for the IT1 loop). This functoid has two inputs; the first is the output from the Equals functoid, ensuring that the line item is valid (equal to "Standard"); the second is the output from the Multiplication functoid. Validate that the output from the Equals functoid is the first input parameter, as it is a boolean value.
3. Drop a Cumulative Sum functoid on the map surface to the right of the others. The input to this functoid is the output of the Value Mapping functoid. The cumulative sum is a total of all line items mapped in the IT1 segment.
4. Drop the output of the Cumulative Sum functoid on the CTT02 element in the target.

5. All of the mapping steps have been completed, and the map is ready to be saved and tested. Save the map file in Visual Studio.

Testing the Map

Throughout the development of the map, it is essential to test the output. For the majority of mapping development, all map tests can be done within the Visual Studio environment. Working in Visual Studio, it is easy to test the map, giving a sample input instance and generating the output document. This section provides a detailed exercise demonstrating how to test the map created in the previous section.

There are several fields that have been configured to fail (in the exercise just completed) when testing the map. This is to demonstrate how to find and fix errors during testing. The errors occur on the following nodes:

- **IT104**: By default, this element is mapped with decimal places (from the source schema element PRICE), since the value in SQL Server is returned as SQL data type Money. The target schema has a restriction on this field that requires that it is a valid currency value (two decimals or less). A functoid will be used to solve this issue.

- **N402**: The target schema indicates that this should be a two-character code indicating the state or province. The source data contains the entire name of the state and is a standard string. The trading partner requires that the entire string be passed in the EDI document. The target schema will be modified to solve this issue.

The document shown in Listing 3-6 is a sample instance of a result set returned by the stored procedure configured earlier in this chapter. Testing a map is most easily done when using a valid input document (it can be done with an automatically generated instance, but this often does not adhere to valid data). This is why defining the source data and putting into place the components for retrieval are done before beginning any mapping development. Exercise 3-5 demonstrates how to test a map.

Listing 3-6. *Sample Input Instance*

```
<COMMON_810 xmlns="http://schemas.Apress.com/Common/810/v1">
<TRANSACTION>
  <HEADER>
    <GUID>8F97D20B-687F-4B82-A231-13BB30944E47</GUID>
    <DOCID>1000</DOCID>
    <DESC>Sample Invoice #1 for Company Y</DESC>
    <PARTNER>Company Y</PARTNER>
  </HEADER>
  <ADDRESSES>
    <ADDRESS>
      <TYPE>Billing</TYPE>
      <STREET>99 Highway R</STREET>
      <CITY>City R</CITY>
      <STATE>State B</STATE>
      <ZIP>23456</ZIP>
    </ADDRESS>
  </ADDRESSES>
  <ITEMS>
    <ITEM>
      <TYPE>Standard</TYPE>
      <PRICE>12.0000</PRICE>
      <DESC>Item K</DESC>
      <QTY>2</QTY>
    </ITEM>
    <ITEM>
      <TYPE>Standard</TYPE>
      <PRICE>10.0000</PRICE>
      <DESC>Item K</DESC>
      <QTY>2</QTY>
    </ITEM>
    <ITEM>
      <TYPE>Hidden</TYPE>
      <PRICE>100.0000</PRICE>
      <DESC>Item J</DESC>
      <QTY>1</QTY>
    </ITEM>
  </ITEMS>
</TRANSACTION>
</COMMON_810>
```

Exercise 3-5. Testing the EDI Map

Use the following steps to test the map created in Exercise 3-4:

1. Open the Apress.Integration.EDI810 solution that contains the Visual Studio map created in the previous section. This can be found in the sample files at Apress.Integration\Chapter 3\Apress.Integration.EDI810.sln.
2. Build the solution in Visual Studio (right-click the solution and select Build Solution).
3. In the Apress.Integration.EDI810.CompanyX project, right-click the map that is going to be tested (CommonXMLToCompanyX810.btm) and select Properties. Begin with an XML output that is not validated. Aside from the TestMap Input Instance, set the properties as shown in Figure 3-20.

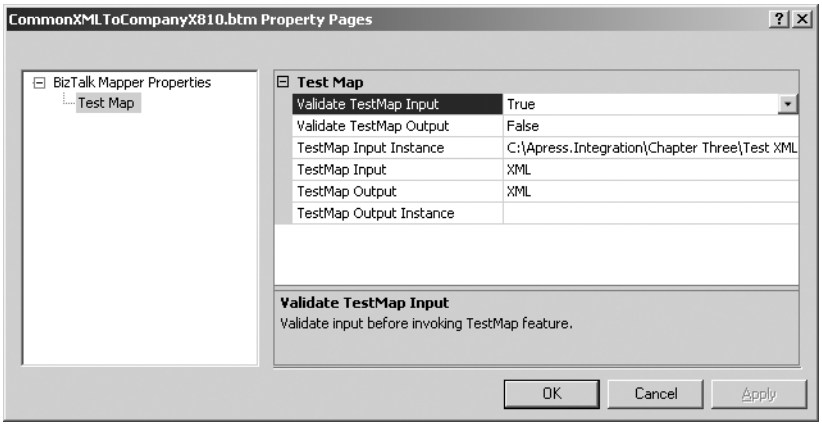

Figure 3-20. *The map property page with initial configuration*

4. Set the TestMap Input Instance to a valid source document. This is the document that is returned by the SQL Server stored procedure, along with the root node that wraps the document when the SQL Adapter retrieves the document. See Listing 3-6 for a valid instance, or open an actual instance from the sample files at Apress.Integration\Chapter 3\Test Documents\Source 810 XML.xml.
5. Click OK to save the settings and close the window.
6. Right-click the map and select Test Map. This will generate a nonvalidated XML instance of the map output. The output box will display a link to the output document. If the output box is not available, click View ➤ Output on the menu bar. Additionally, if the X12 EDI Instance Properties dialog box pops up, accept the default values and click OK. The output document can be viewed by holding the CTRL key and clicking the link next to the output: The Output Is Stored in the Following File. Figure 3-21 shows the instance as it appears after clicking this link.

Figure 3-21. *Test map output in XML*

7. With the output visually validated (i.e., matching what is expected by the developer to be output), the test map settings can be set to a higher level of validation. Right-click the map and select Properties. This time, set the Validate TestMap Output property to True.

8. Click OK, right-click the map again, and select Test Map. This time an error is thrown. There is invalid data in the N402 node. The sample data contains the string *"*State B*,"* whereas the target schema indicates that the N402 node must contain a two-character code (state or province abbreviation). In this case, there are three options for solving this mapping issue:

   a. **Change the source data**: Instead of extracting the string from the database exactly as it is stored, logic could be implemented in the stored procedure to return the data in a two-character code.

   b. **Change the map**: Logic could be put into place in the map to convert the full string to a valid two-character string. This would most likely require either a lookup table in a database (string to state code characters) or a large case statement (case "California" then "CA").

c. **Change the target schema**: Depending on the requirements of the trading partner, it may be that the best place to change the requirement is in the target schema. Remove the restriction from the N402 element and save the schema. For this exercise, the approach that will be taken is to modify the target schema. In the case of the trading partner, Company X, the entire string denoting the state is required (i.e., the partner wants the full string, not just the two-character code). The steps for modifying the target schema are as follows:

   i. Open the target schema (X12_00401_810_CompanyX.xsd) in Visual Studio.

   ii. Expand the N1Loop1 node and click the N4 element N402. The properties should be displayed automatically. If they are not visible, right-click the element and select Properties. Change the Length property to nothing—remove the 2. Save the schema. This allows any length of string to be present in the element.

   iii. Rebuild the solution (right-click the solution and select Rebuild Solution).

      1. If the Clean Up Global Types window is open, click OK. There is no need to do anything with the information in this window.

      2. If the X12 EDI Instance Properties window opens, skip it for now. It will be used later in this exercise.

      3. If a dialog appears indicating that the destination schema for a map has changed, click OK.

9. Retest the map. Right-click the map and select Test Map. Two additional errors occur; this time on the IT104 node. The error indicates that the current value has too many decimal places and is not recognized as a valid value in the target schema. Again, there are three options in solving this issue:

   a. **Change the source data**: The data is stored in SQL as a data type of Money and automatically sets four decimal places. The stored procedure could be modified to return two decimals or round to the nearest dollar. However, by fixing this in the stored procedure, the data will be extracted the same for all trading partners. Some partners may wish to have two decimals, while others may want it rounded to the nearest dollar (no decimal places). Changing this in the stored procedure would limit the result to one or the other.

   b. **Change the target schema**: The target schema for the trading partner could have the restriction removed from the IT104 node. However, for this exercise, the trading partner requires that the restriction remain.

   c. **Change the map**: The map can be updated with additional logic to ensure that the price is rounded and matches the target schema requirements. In this case, changing the map is the desired solution. See Figure 3-22 for a completed view of the modified map.

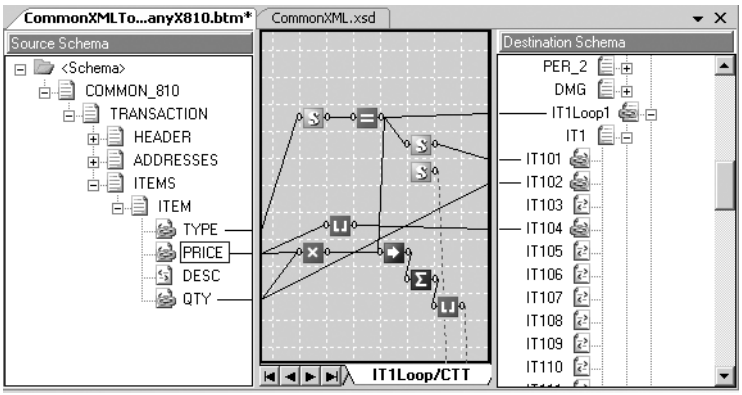

Figure 3-22. *Adding the round functoid to fix the IT1 and CTT rounding issue*

10. Open the map in Visual Studio. Remove the link between PRICE and IT104 by right-clicking it and selecting Delete.

11. Drop a Round functoid on the mapping surface.

12. Set the first input to this functoid as the PRICE element from the source schema. The second input should be a 2, which is the number of decimals to round the value to.
13. Drag the output of the Round functoid and drop it on the IT104 element in the target schema.
14. A similar approach can be taken for the Cumulative Sum functoid, which outputs to the CTT02 element. Drop a Round functoid to ensure that the value output from this functoid is rounded to two decimal places.
15. Save the map. Retest by right-clicking the map and selecting Test Map. This time the output will be validated.
16. The final step is to set the test map output to the Native EDI format. Right-click the map and select Properties. For the TestMap Output property, select Native. This time when the map is tested, the output will be generated in standard EDI format (instead of XML). Different trading partners require output in different formats, with different delimiters and trailing separators. All of this is configured for each trading partner in the EDI properties in the party settings. For testing the map, however, this is configured each time the map is tested in native format.
    a. Right-click the map and select Test Map.
    b. The X12 EDI Instance Properties dialog box will open; up to this point, this window has been ignored, as it has no impact on the XML output. However, in native format, the output of the document can be controlled. The most important setting to configure is the segment separator suffix. This value can be set to control whether each segment is on its own line, or whether the output is all on a single line. The only purpose for this during map testing is to allow a developer to more easily validate the output document, especially when checking against existing EDI documents. These settings have no impact on the map itself. Set the segment separator to CR LF and ensure that the Use Trailing Delimiters property is set to Yes. Continue to click OK until this value has been set (this dialog box may appear multiple times).
    c. The output box will again give a link to the generated output. Clicking the link will open the document shown in Figure 3-23. The output shown in Figure 3-23 matches the output shown earlier in the chapter in Listing 3-1.

```
ST*810*1000~
N1*ST*Company Y~
N4*City R*State B*23456~
IT1*1*2**12~
IT1*2*2**10~
CTT*2*44~
SE*7*1000~
```

Figure 3-23. *Test map output in native EDI format*

17. Save and rebuild the solution. The testing of the map is complete.

Final Discussion

This chapter covers many aspects of defining, retrieving, and mapping data. The mapping steps were specific to X12 EDI documents, but the principles of mapping are the same for all EDIFACT document types. Once the maps have been fully developed and tested, they must be deployed to BizTalk and added to the trading partner ports or incorporated into orchestrations. The steps for deploying, configuring, and adding the maps to trading partners is covered in detail in later chapters.

Additional topics that may be of use in EDI document mapping are the following:

- **Result sets as flat file feeds**: On occasion, it may be necessary to have the external data source process and deliver the data to BizTalk, rather than using BizTalk to poll it through a SQL Receive Adapter. Perhaps linked servers are not an option (the processing/network overhead is deemed to be too high or SQL Server is unable to connect), or perhaps it makes better business sense to process the data in large batches (on a daily or weekly schedule, rather than a single document at a time). In such cases other delivery methods must be implemented. An example of this would be using the SAP Adapter to retrieve IDOCS from an SAP system.
- In Figure 3-24, a high-level flow of publishing data to BizTalk server is shown. Whether that data is preformed in XML, or whether it is a flat file and requires a pipeline on the file adapter, the process of delivery is incumbent on the source application. BizTalk is a passive listener and waits for the data to be delivered.

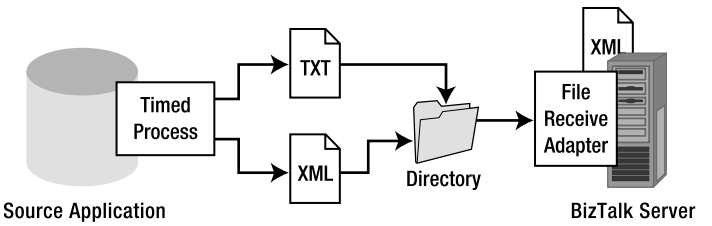

Figure 3-24. *Retrieving data as a file*

- **Data lookup tables**: Source data may require lookup tables in the map to properly convert the value to what the target party is expecting. In such situations, using some of the database functoids may be of use, or creating a custom functoid may be required.
- **XSLT and complex mapping**: For complex mapping situations, it may be necessary to use XSLT either in combination with, or as a substitute for, the graphical BizTalk mapping interface. XSLT documents provide a developer with complete flexibility in mapping data, provided that the developer understands XSLT and that it is a viable solution that can be supported once it has been developed.