Development strategies for creating efficient, simple, extendable BizTalk solutions

BizTalk 2013 EDI for Supply Chain Management

Working with Invoices, Purchase Orders and Related Document Types

Mark Beckner

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

BizTalk is a development platform and lends itself to a wide variety of implementation styles. After you've worked with the platform on a number of projects, however, you will find that there are really only a few patterns that meet the critical requirements of a well-developed solution. With every project, you must ask the following questions:

1. Do I have all of the requirements, and is the business communicating the correct requirements to me?
2. Is there a way to architect this solution that will ensure I have the simplest set of components required to successfully deliver?
3. Will I be able to easily communicate how to work with this to someone else?
4. Is my solution maintainable and supportable?

The examples outlined in this book are intended to give you a pattern on which to build your own implementations. They outline the common tasks that will be encountered on virtually any EDI project, and they also show some unique approaches to solving complex problems. Most importantly, they demonstrate that BizTalk EDI solutions can be simple and fairly quick to implement.

My hope is that this book will aid you in realizing success with your own projects.

Contacting the Author

If you have questions regarding the concepts in this book, or would like to discuss architecture, mentoring, or development options, please contact me at mbeckner@inotekgroup.com.

Solution: Receiving 850 Data

Chapter 1

This chapter will walk through a complete end-to-end solution on how to build out BizTalk to receive 850 (Purchase Order) documents from an external trading partner and send an acknowledgement back. The data will be received via an SFTP adapter and then it will be archived and processed by an orchestration. In the orchestration, it will be determined whether the purchase order needs to be reviewed manually by an internal user prior to approval, or whether it can be delivered automatically as a flat file to the internal order processing application. This will introduce many of the key concepts required in working with inbound data.

The data will be received in unencrypted format on an SFTP site, will be archived by a BizTalk orchestration using a .NET library, and will be mapped to a flat file format. The rules determining whether the data can be automatically approved, or whether it required manual approval, will be handled in the orchestration, but additional ideas on how to make this more robust will be presented at the end of this chapter. The overview of this specific solution is shown in Figure 1-1.

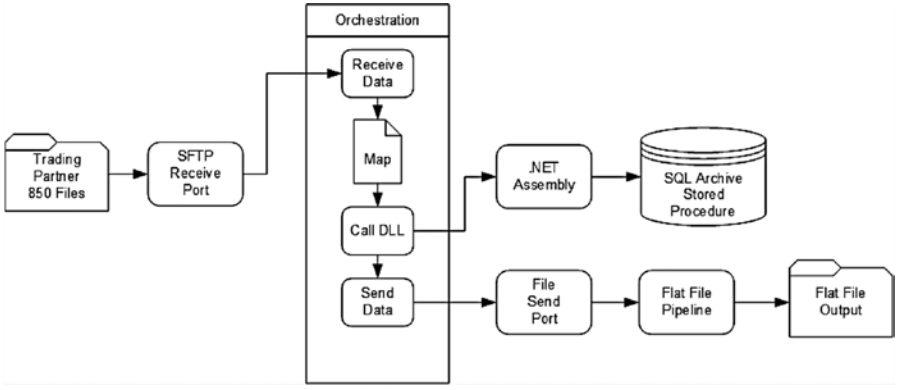

Figure 1-1. Inbound 850 solution overview

Visual Studio Solution

It is critical that your project structure and namespaces are correct from the start. If these are not exactly what you need for the proper architecture and organization of your solution, you'll be spending a great deal of time later in the process rewriting and retesting. For this solution, the namespace is in a structure that you should be able to use directly within your own solutions, substituting the wording, but not the structure. You will also be creating a Visual Studio project structure that will be generic enough to fit within any model you may encounter. In the case of the solution being built out, the following Visual Studio projects and namespaces will be used:

- **Solution Name:** Demo.BizTalk. This is a generic solution that can hold inbound and outbound projects. You may have several projects that are common to many projects, so having everything in one solution can be very helpful. Examples of such projects would be common schemas and .NET helper classes.
- **Schemas:** There are three schemas that will be used in this solution. All three of these will be contained in a single project called Demo.BizTalk.Schemas.X850.
  - The 850 Schema ships with BizTalk. All of the out-of-the-box schemas are contained in the `MicrosoftEdiXSDTemplates.exe` file found in `C:\Program Files (x86)\Microsoft BizTalk Server 2013\XSD_Schema\EDI`.
  - The Target flat file schema, which represents the target data to which the 850 is being mapped. This must be created using the flat file wizard.

> **Note** When setting a namespace, never use a numeric value alone without at least one leading text character (such as the 850 in `Demo.BizTalk.Schemas.850`), as it could result in a variety of potential naming conflicts, unexpected errors, and challenges in testing. If you wish to refer to an EDI document type directly in your namespace, use a pattern such as a leading "X", like X850.

- **Maps:** The map project will contain all maps required by the solution, and will have a namespace of Demo.BizTalk.Maps.X850.
- **Helper Library:** There will be one external .NET assembly project with the namespace of Demo.BizTalk.Helper.X850.
- **Orchestration:** There will be one orchestration used, which will be in its own project called Demo.BizTalk.Orchestrations.X850.
- **Pipeline:** There will be one custom Send pipeline project, which will be called Demo.BizTalk.Inbound.Pipelines.X850.

The Schema Project

There will be two schemas required for this project. The first is the 850 schema that ships with BizTalk. BizTalk has thousands of EDI schemas that come with it, crossing all of the document types and versions available. The second is the Target flat file that you'll be mapping the 850 to, which represents the format that the internal order processing application requires. An example of the source 850 schema is show in Figure 1-2.

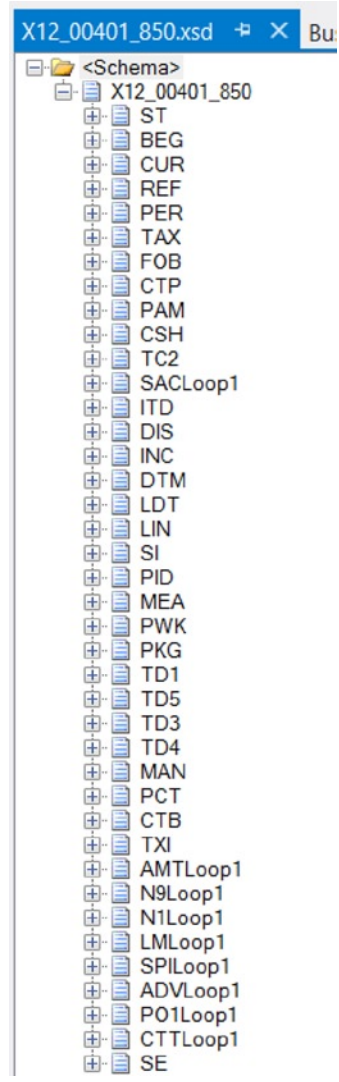

Figure 1-2. The 850 Schema

> **Note** To access the EDI schemas with BizTalk, browse to the Microsoft BizTalk Server 2010 root folder and go to `XSD_Schema\EDI`. In this directory you will find a file called `MicrosoftEdiXSDTemplates.exe`. Running this file will extract all available schemas.

The schema project is the foundation of many of the other projects you will be creating, as these other projects reference them. If the schemas change during the course of development, all of the other projects will be impacted. Do everything you can to get the schemas namespaced and laid out correctly at the start of your development in order to minimize the impact to projects (especially maps) that reference these schemas.

The Map Project

The map project will allow for the mapping of the inbound 850 data into the target flat file format. The 850 data can be mapped into the target format using whatever mapping techniques are required—standard functoids, lookups, even XSLT. Chapter 4 is completely dedicated to mapping, and describes how best to approach this task. For this exercise, the map project structure should be as follows:

- Create a new project in Visual Studio called Demo.BizTalk.Maps.X850.
- Add a reference to the schema project you have created.

The .NET Helper Library Project

The .NET helper library is used by the orchestration to archive the inbound XML version of the EDI file to a database in its native XML format. This is an invaluable way of being able to access and report on data through SQL Business Intelligence (BI) platforms, such as SSRS, without having to push the 850 data to a tradition database model. The .NET class will have a single method in it that looks similar to the code shown in Listing 1-1. You can pass as many or as few parameters as you would like, depending on the needs of your reporting.

> **Note** Always mark your .NET classes as Serializable, so that they can be called from anywhere within BizTalk. To do this, type [Serializable] directly above the class declaration in your helper library.

Listing 1-1. Method Called from Orchestration to Archive Data to SQL

```
public void ArchiveInboundData(string
strSourceFileName, string
strTradingPartner, XmlDocument xmlSource, string
strApprovalStatus, string strConnectionString)
{
    SqlConnection sqlConnection = new
SqlConnection(strConnectionString);
    SqlCommand sqlCommand = sqlConnection.CreateCommand();
    sqlCommand.CommandText = "spInsertInboundData";
    sqlCommand.CommandType = CommandType.StoredProcedure;
    sqlConnection.Open();

    SqlParameter sqlParameter = new SqlParameter();

    sqlParameter.ParameterName = "@vchSourceFileName";
    sqlParameter.SqlDbType = SqlDbType.VarChar;
    sqlParameter.Direction = ParameterDirection.Input;
    sqlParameter.Value = strSourceFileName;
    sqlCommand.Parameters.Add(sqlParameter);

    sqlParameter = new SqlParameter();
    sqlParameter.ParameterName = "@vchTradingPartner";
    sqlParameter.SqlDbType = SqlDbType.VarChar;
    sqlParameter.Direction = ParameterDirection.Input;
    sqlParameter.Value = strTradingPartner;
    sqlCommand.Parameters.Add(sqlParameter);

    sqlParameter = new SqlParameter();
    sqlParameter.ParameterName = "@xmlSourceData";
    sqlParameter.SqlDbType = SqlDbType.Xml;
    sqlParameter.Direction = ParameterDirection.Input;
    sqlParameter.Value = new XmlNodeReader(xmlSource);
    sqlCommand.Parameters.Add(sqlParameter);

    sqlParameter = new SqlParameter();
    sqlParameter.ParameterName = "@vchApprovalStatus";
    sqlParameter.SqlDbType = SqlDbType.VarChar;
    sqlParameter.Direction = ParameterDirection.Input;
    sqlParameter.Value = strApprovalStatus;
    sqlCommand.Parameters.Add(sqlParameter);

    sqlCommand.ExecuteNonQuery();
    sqlConnection.Close();
}
```