Deployment Framework for BizTalk Server V5.0

Table of contents

Introduction

Deployment Framework for BizTalk Server V5.0

The Deployment Framework for BizTalk eliminates the pain associated with BizTalk application deployments, and goes far beyond BizTalk's out-of-the-box deployment functionality. It also includes tools to enhance developer productivity, such as binding file and runtime settings management.

The Deployment Framework for BizTalk is the single most powerful and customizable, yet easy-to-use toolkit for deploying and configuring your BizTalk solutions.

Top Five Reasons to Use the Deployment Framework for BizTalk:

1. Deploy a complex solution containing orchestrations, schemas, maps, rules -- even ESB itineraries -- in minutes, with no human intervention
2. Eliminate ALL manual steps in your BizTalk deployments
3. Consolidate all of your environment-specific configuration and runtime settings into one, easy-to-use Excel spreadsheet
4. Maintain a SINGLE binding file that works for all deployment environments
5. Make automated deployment a native part of the BizTalk development cycle, then use the same script to deploy to your servers

In this section

Welcome

Let's begin by stating the obvious: BizTalk Server is a complex product. It is both a development platform and a runtime, defining some artifacts of its own, such as orchestrations and pipelines, while welcoming the inclusion of standard XML schemas, XML transforms and any manner of .NET-based components. It includes a rules engine, business activity monitoring framework and more. To make it all work at runtime, BizTalk requires nearly everything to be transformed into .NET code and deployed in a very specific way.

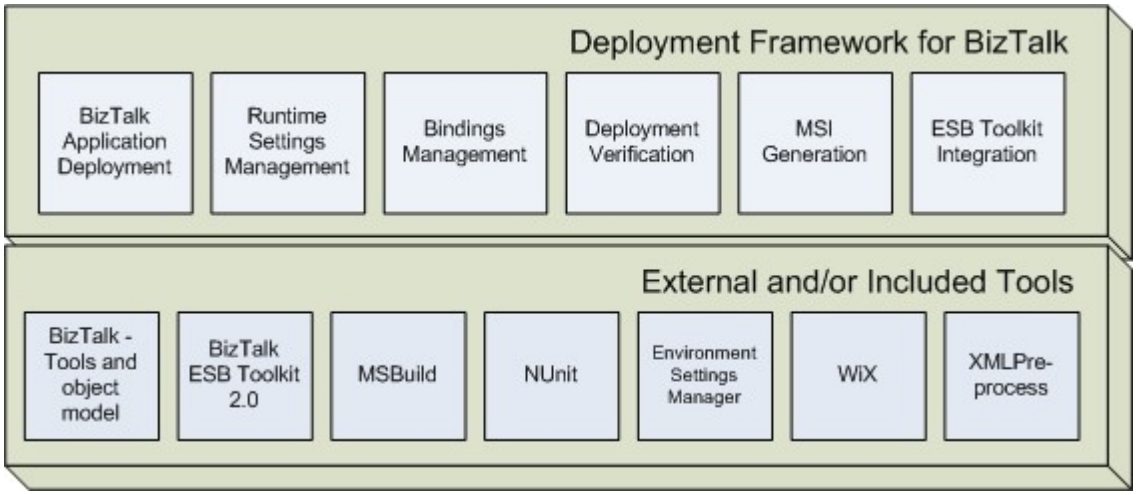
Needless to say, as a BizTalk developer it is far too easy to spend hours on end dealing with the overhead of the product's deployment requirements and other – to put it kindly – "quirks," instead of creating and testing the actual functionality that your solution requires. If you are reading this text, then you have undoubtedly experienced this frustration yourself. There is a clear need for a set of tools that both simplifies the developer experience and boosts developer productivity – the **Deployment Framework for BizTalk, or BTDF**.

The Deployment Framework for BizTalk eliminates the pain associated with BizTalk application deployments, and goes far beyond BizTalk's out-of-box deployment functionality by including various tools to enhance BizTalk developer productivity.

The Deployment Framework includes the following features:

- Automation of the entire BizTalk application deployment and update processes
- Support for BizTalk Server 2006, 2006 R2, 2009 and 2010 and BizTalk ESB Toolkit 2.0 and 2.1 on both 32- and 64-bit Windows
- Support for Windows XP, Vista and 7 and Windows Server 2003 through 2008 R2
- Integration with Visual Studio 2005/2008/2010 menus, toolbars and output window, plus IntelliSense and Add New Project wizard
- Support for deployment of various BizTalk artifacts including:
    - Messaging bindings
    - Orchestrations
    - Schemas
    - Maps
    - Pipelines
    - Custom components (DLL's)
    - Custom pipeline components
    - Custom functoids
    - Rules and vocabularies
    - IIS virtual directories
    - Single Sign-On (SSO) applications
    - BAM activities
    - ESB Toolkit itineraries
    - Configuration settings infrastructure including user-friendly settings management spreadsheet, .NET API for settings access at runtime and custom ESB Resolver
- Templated bindings file processing that automatically targets multiple runtime environments from a single bindings file
- Full and Fast deployment modes to reduce development cycle time
- Detailed logging for informational and troubleshooting purposes
- Enables use of un-encoded XML for adapter and port configurations in binding files, allowing easy manual editing
- Single deployment script serves both development workstations and servers
- Automated packaging of entire application into standard Windows Installer MSI file
- Easily customizable installation wizard for server deployments
- Support for side-by-side deployment of multiple versions of a single application
- Deployment verification via NUnit unit testing tool
- Integrated deployment of Log4Net for runtime event logging
- Automated configuration of BizTalk runtime settings including debugging features and .NET assembly-to-AppDomain mappings
- Automatic creation/cleanup and NTFS permission assignment of FILE adapter physical paths
- Automated restart of one or more BizTalk host instances and IIS or selected AppPools
- Automatic addition of BizTalk application references
- Automatic deployment of debugging PDB files to the Global Assembly Cache (GAC)
- Infinite extensibility through open-source license
- Extensibility through user-defined MSBuild targets and tasks
- **And more**!

The diagram below illustrates the major components that make up the Deployment Framework:



Virtually all of the features mentioned above may be selectively enabled or disabled and easily customized to meet the particular requirements of your application.

What's New

Version 5.0 of the Deployment Framework for BizTalk represents a major overhaul and transition from what was more of a tool/sample into a true product. This is a very important and exciting milestone, and we strongly believe that every BizTalk developer can and will benefit from the Deployment Framework.

There have been so many changes since version 4.0, way back in 2008, that it only makes sense to hit the highlights here. If you'd like to see how much has really changed, you'll find that detail on the Change History page on CodePlex.

Here's an **abridged** list of the biggest changes for V5.0 (see the Change History for the complete list):

- **NEW: Support for BizTalk Server 2009 and 2010** - now supports BizTalk Server 2006, 2006 R2, 2009 and 2010 and BizTalk ESB Toolkit 2.0 and 2.1
- **NEW: Complete conversion from NAnt to MSBuild** with file structure modeled after standard .csproj/ .vbproj project files
- **NEW: Consolidated Deployment Framework files** under a <projectname>.Deployment folder and integrated server MSI build, leaving the solution root clean
- **NEW: Visual Studio 2005/2008/2010 Add-in** including integrated menu and toolbar, solution awareness, IntelliSense and Add New Project wizard
- **NEW: Windows Installer MSI** to install and configure the Deployment Framework
- **NEW: Core Framework enhancements**
  - Complete flexibility on project structure, file naming and locations
  - Integration with ESB Toolkit 2.0 and 2.1
  - Completely revamped support for BRE
  - New samples that demonstrate Deploment Framework features
  - Automatic creation/cleanup of FILE adapter physical paths, including NTFS permission assignment
  - Automated export and deployment of BAM XML from a BAM XLS file to avoid needing Excel on the server
  - Support for Windows Vista and Windows Server 2008 and 2008 R2 (including UAC elevation)
  - Integration with TFS Team Build
- **CHANGE: Discontinued support for BizTalk 2004** (BizTalk 2004 users, please use Version 4.0)

Please see the Issue Tracker for open issues for this and future releases and the Change History for complete details.

Upgrading from V4.0 or Older

The Deployment Framework for BizTalk underwent major changes for V5.0. The biggest change is the conversion from NAnt to MSBuild, but nearly every aspect of the product has changed. Every upgrade will require manual changes.

Many of the files that were formerly kept in the solution root folder and in subdirectories have been removed or moved. The Deployment Framework for BizTalk V5.0 completely separates user-editable files from Framework files. Your BizTalk solution will contain only the files that you must edit and maintain yourself, and all of those files will be stored in one Deployment project folder under your solution.

The rigid project and file naming conventions of V4.0 and earlier are gone. You can name your projects and files whatever you like.

Setting up a new project in V5.0 is much simpler than in previous releases, so in most cases the fastest and simplest approach is to start over. Use the Add New Project Wizard to get started.

However, if you want to try upgrading your old project, the Developer folder under the Deployment Framework installation folder contains a simple batch file called MigrateToProjectFolderStructure.cmd. You can copy this file to the root of your pre-5.0 BizTalk solution and run it from a Command Prompt to jump start your upgrade. The script will move, rename and delete most of the common files that change during an upgrade. Watch out for files that you may have bound to source control that will be affected by moves, renames or deletions.

Upgrading from V5.0 Pre-Release

Updating Prerelease Deployment Project Files

Deployment project files created in any version of the Deployment Framework prior to V5.0 RTW require a few modifications.

Native support for Team Build was added **after** publication of the final release candidate of Deployment Framework for BizTalk 5.0.

1. Update the DefaultTargets attribute
In the Project element, change the DefaultTargets attribute value to Installer.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="Installer">
```

2. Update the OutputPath elements in the default Debug and Release configuration PropertyGroup's
In the two PropertyGroup elements that are conditional on the Debug and Release configurations, replace the OutputPath element with two lines that are conditional on the property TeamBuildOutDir. The following example includes the new OutputPath elements. You may already have other elements mixed in with them, so just replace your two existing OutputPath elements with the four shown below.

```
  <PropertyGroup Condition="'$(Configuration)' == 'Debug'">
    <DeploymentFrameworkTargetsPath>$(MSBuildExtensionsPath)
\DeploymentFrameworkForBizTalk\5.0\</DeploymentFrameworkTargetsPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' == ''">bin\Debug\</OutputPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' != ''">$(TeamBuildOutDir)</OutputPath>
    <!-- Get our PDBs into the GAC so we get file/line number information in stack
traces. -->
    <DeployPDBsToGac>false</DeployPDBsToGac>
  </PropertyGroup>
```

```
  <PropertyGroup Condition="'$(Configuration)' == 'Release'">
    <DeploymentFrameworkTargetsPath>$(MSBuildExtensionsPath)
\DeploymentFrameworkForBizTalk\5.0\</DeploymentFrameworkTargetsPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' == ''">bin\Release\</OutputPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' != ''">$(TeamBuildOutDir)</OutputPath>
    <!-- Get our PDBs into the GAC so we get file/line number information in stack
traces. -->
    <DeployPDBsToGac>false</DeployPDBsToGac>
  </PropertyGroup>
```

For further reference, the Deployment Framework for BizTalk sample applications already include the correct configuration for Team Build.

3. Add ToolsVersion attribute for BizTalk Server 2010
If you are using BizTalk Server 2010 (only), add the ToolsVersion attribute to the Project element.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="Installer" ToolsVersion="4.0">
```


License Agreement

Microsoft Public License (Ms-PL)

This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software.

1. Definitions

The terms "reproduce," "reproduction," "derivative works," and "distribution" have the same meaning here as under U.S. copyright law.

A "contribution" is the original software, or any additions or changes to the software.

A "contributor" is any person that distributes its contribution under this license.

"Licensed patents" are a contributor's patent claims that read directly on its contribution.

2. Grant of Rights

(A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create.

(B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software.

3. Conditions and Limitations

(A) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks.

(B) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically.

(C) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and

attribution notices that are present in the software.

(D) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license.

(E) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

Getting Started

This section identifies the system requirements for the Deployment Framework for BizTalk and provides detailed installation instructions.

In this section

System Requirements

The Deployment Framework has slightly different requirements for servers vs. development workstations.

BizTalk Servers

The Deployment Framework requires the following software on BizTalk servers:

1. Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, Windows Server 2008 R2
2. Microsoft BizTalk Server 2006, 2006 R2, 2009 or 2010
3. Optional: ESB Toolkit 2.0 or 2.1

Development Workstations

The Deployment Framework requires the following software on BizTalk development workstations:

1. Windows XP SP3, Windows Vista SP1, Windows 7 SP1, Windows Server 2003/2003 R2, Windows Server 2008/2008 R2
2. Microsoft BizTalk Server 2006, 2006 R2, 2009 or 2010
3. Optional: ESB Toolkit 2.0 or 2.1
4. Microsoft Excel 2003 or newer (required for editing environment settings spreadsheet)

Getting Help

Ideally you'll find the help you need here in the documentation, but if you don't, then please hop over to our CodePlex website and check out the Issue Tracker and Discussions areas. In the Issue Tracker you can review existing bugs and feature requests or submit your own. The Discussions area should be your first stop to both search for and ask for help. Of course, you'll always find the latest release and the latest work-in-progress code in the Source Code area.

Installation

Before installing the Deployment Framework for BizTalk, make sure that the prerequisite software is installed and close any running instances of Visual Studio.

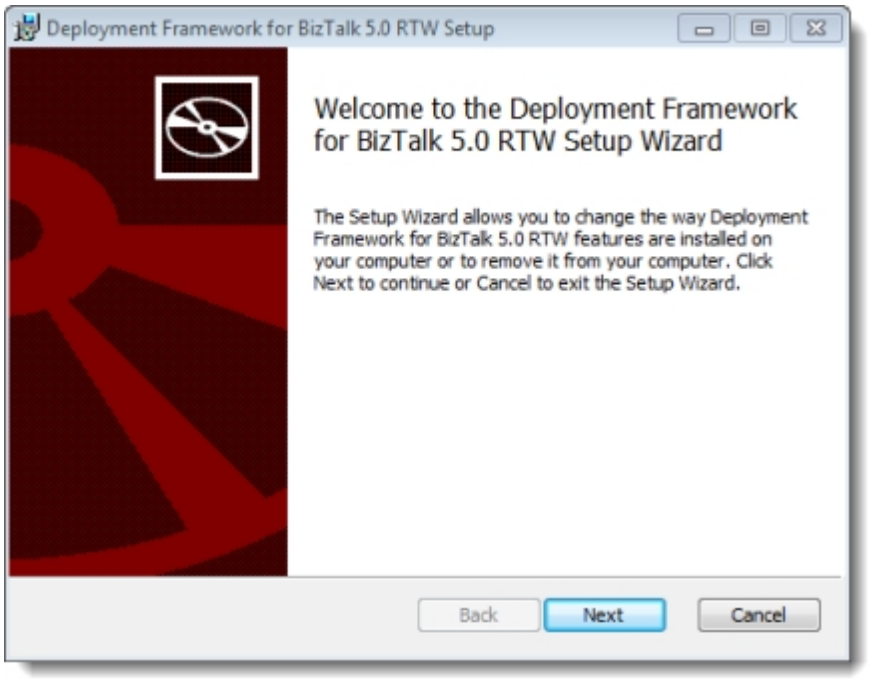
Installing on a Developer Workstation

Install the Deployment Framework for BizTalk on each BizTalk developer's workstation. In the majority of situations the Typical install is appropriate, as it includes everything but the source code for the Deployment Framework's tools and MSBuild tasks.

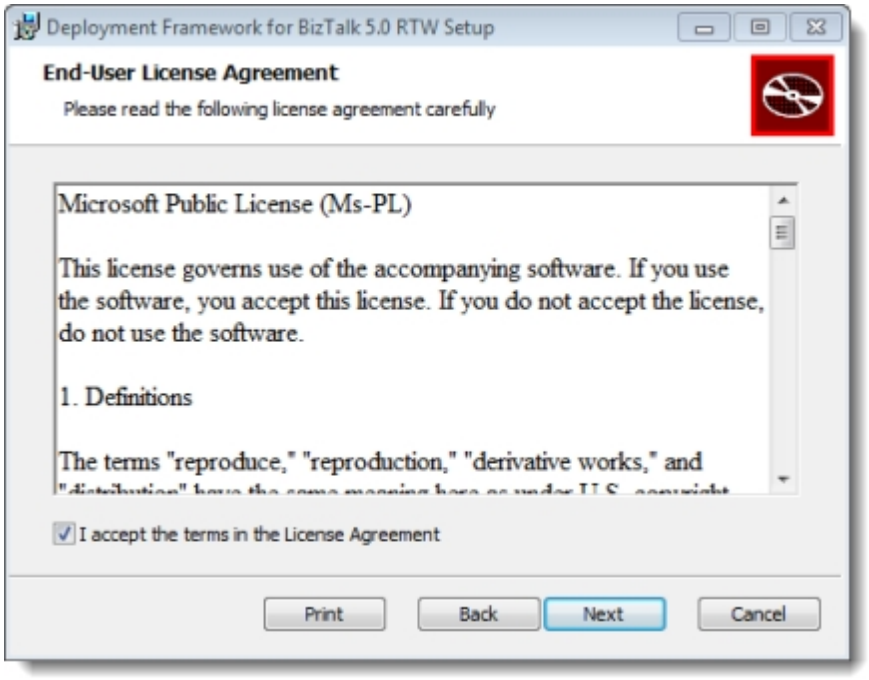
Note for BizTalk 2010 Users: After completing the Deployment Framework installation, an extra step is required to complete installation of the ESB Toolkit 2.1 integration.

The installation is quick and easy. Here are the steps of the installation wizard:

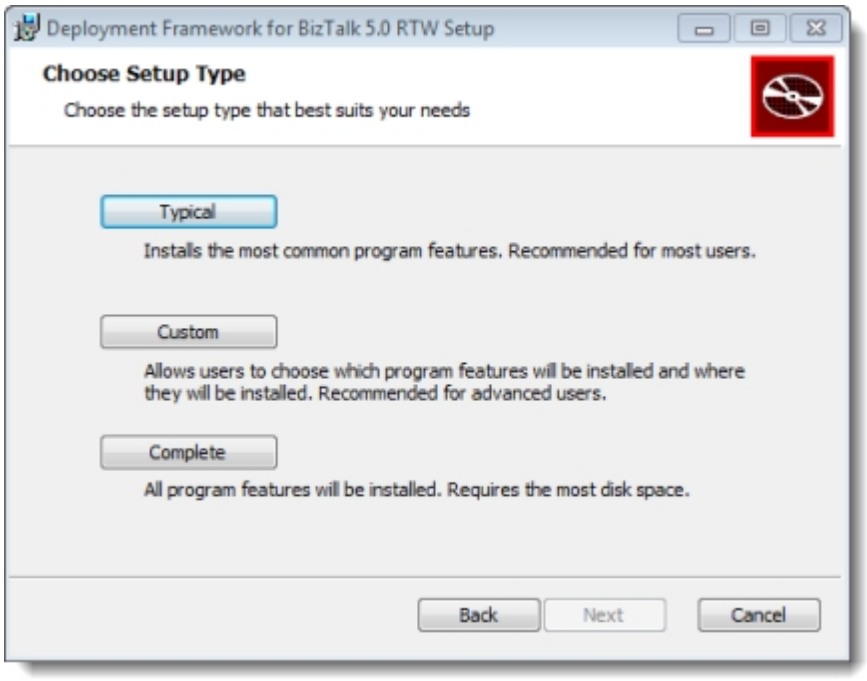
Step 1: Welcome



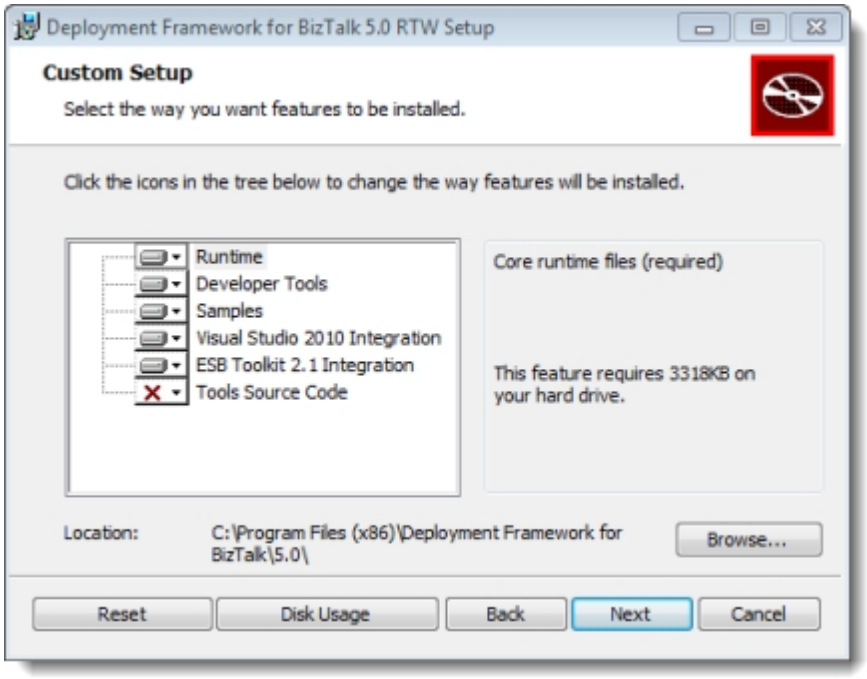
Step 2: License Agreement



Step 3: Setup Type

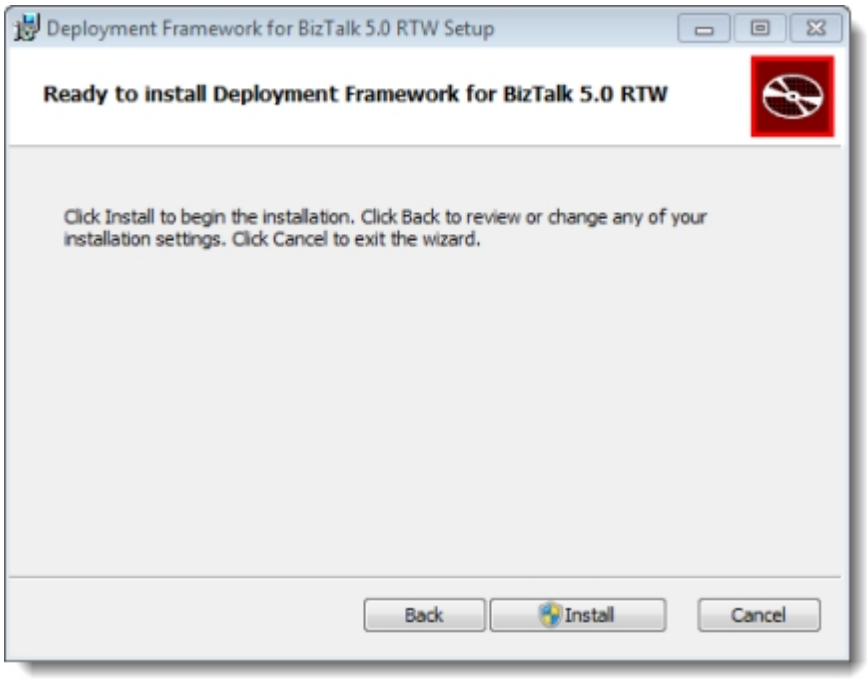


Optional Step: Customize Install

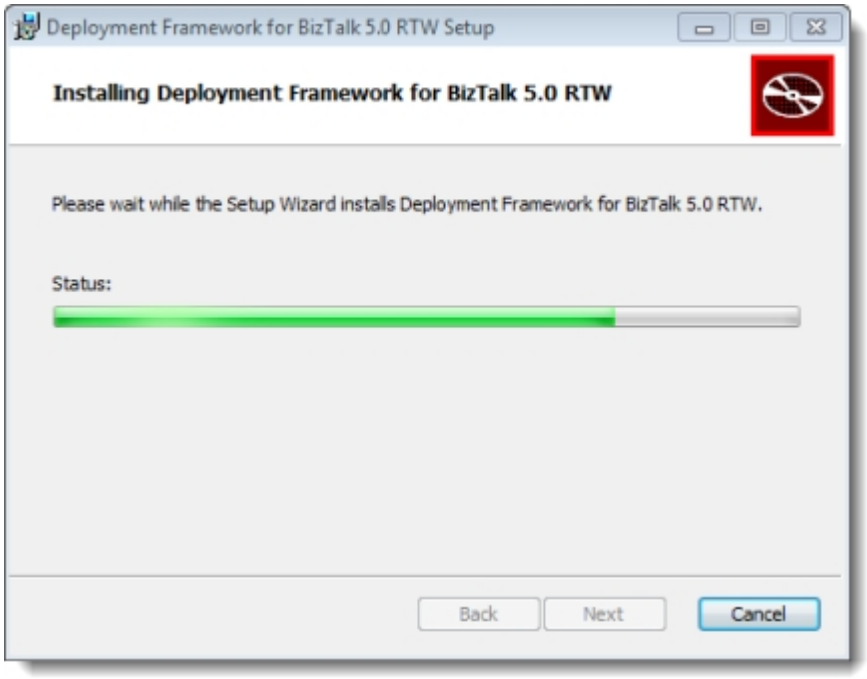


The Visual Studio version will vary depending on which version of BizTalk (and thus Visual Studio) you have installed, and the ESB Toolkit Integration feature will only appear if you have installed the optional Microsoft ESB Toolkit for BizTalk.

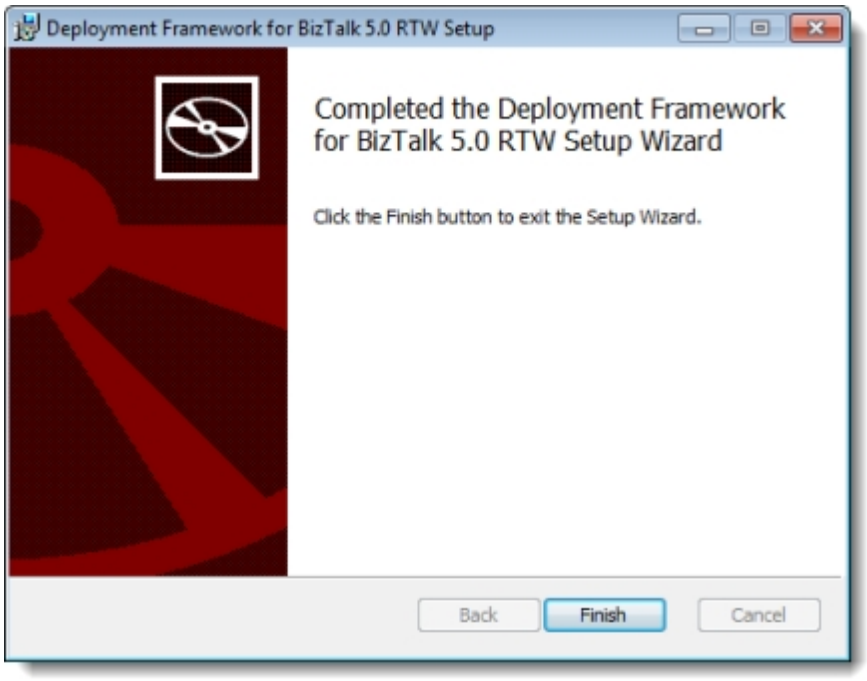
Step 4: Confirm Install



Step 5: Installation Progress



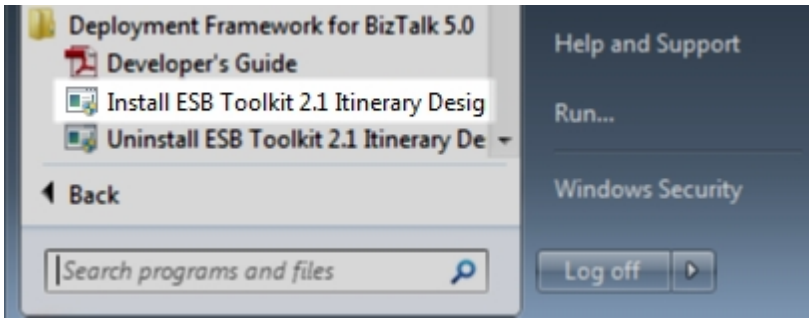
Step 6: Installation Complete



Installing the ESB Toolkit 2.1 Itinerary Designer Extension

This step applies only to **BizTalk 2010** developers who are using the ESB Toolkit 2.1.

After the Deployment Framework installation is complete, there is one more quick step to install the ESB Toolkit Itinerary Designer Extension. In your Start menu, locate the Deployment Framework for BizTalk 5.0 program group. Click the "Install ESB Toolkit 2.1 Itinerary Designer Extension" shortcut to complete the installation.



PLEASE NOTE: To remove the Designer Extension, you must use the "Uninstall ESB Toolkit 2.1 Itinerary Designer Extension" shortcut. If you uninstall the Deployment Framework without using this shortcut, the extension will **not** be removed.

Installing on a BizTalk Server

It is not necessary to install the Deployment Framework on your BizTalk servers UNLESS you are using the BTDF ESB Toolkit SSO Resolver.

To install the ESB Toolkit SSO Resolver on a BizTalk server, use the same Deployment Framework for BizTalk MSI that you used on your developer workstations. This time, choose a "Custom" setup, de-select all features except Runtime and ESB Toolkit 2.x Integration and then complete the installation wizard.

Important Directories

It is helpful to know a few directories where Deployment Framework files are located after installation:

- The Deployment Framework for BizTalk installation folder, usually "Program Files\Deployment Framework for BizTalk\5.0," contains the sample solutions, the developer tools and most of the runtime.

- The MSBuild folder, usually "Program Files\MSBuild\DeploymentFrameworkForBizTalk\5.0," contains the MSBuild target definitions and custom task assemblies.
- The ESB Toolkit 2.x Tools\Itinerary Designer\Lib folder, if you installed the ESB Toolkit 2.x Integration

Overview of Major Components

In order to understand and work effectively with the Deployment Framework, it helps to understand how it works and its various components.

The Deployment Framework is comprised of a runtime and developer tools. The developer tools help to accelerate the development process and, among other things, allow the developer to create an installer package that can be used to deploy the BizTalk solution to your servers. The same runtime underlies both developer and server deployments to ensure a consistent and repeatable process from development all the way through server deployment.

The Deployment Framework for BizTalk installer includes:

1. Core Deployment Framework files
2. MSBuild integration
3. Visual Studio 2005/2008/2010 integration
4. ESB Toolkit 2.x integration
5. Sample BizTalk solutions
6. Documentation
7. Source code for the Deployment Framework tools and MSBuild tasks

In this section

- MSBuild
- WiX (Windows Installer XML)
- Environment Settings Manager

MSBuild

At the core of the Deployment Framework is a tool used to build most modern .NET applications, and BizTalk applications starting with the 2009 release: MSBuild. MSBuild is a "build engine" that ships with the .NET Framework. When you build a C# or VB.NET solution in Visual Studio 2005 or newer, MSBuild is orchestrating the build. Each .csproj or .vbproj file (just to name two common ones) is actually an MSBuild build file.

In essence, MSBuild is an XML-driven process execution engine. The XML manifest/build files define build targets (similar to subroutines) and dependencies, reference files, define properties, and can call .NET extension methods (tasks). The MSBuild engine is responsible for carrying out those instructions in the correct order. MSBuild is much too complex to cover here, so please refer to MSDN and other online and print sources for detailed information.

The important thing to understand is that all actions carried out by the Deployment Framework are driven by MSBuild. **Your deployment project file (.btdfproj file extension) is an MSBuild file.**

You need understand only the basics of MSBuild in order to work with the Deployment Framework. If you wish to customize its behavior or add additional custom deployment steps of your own, then you may need to spend some time learning MSBuild.

TIP: If you are not familiar with MSBuild, I highly recommend reading this introductory blog post from the Visual Studio team blog.

WiX (Windows Installer XML)

Once you have developed a robust, repeatable deployment process for your BizTalk solution, you need a way to package up all of your artifacts – DLL's, configuration files, binding files and so on – and deliver them to your BizTalk servers. For this task, the Deployment Framework uses standard Windows Installer MSI files.

WiX (Windows Installer XML) is an open-source toolset sponsored by Microsoft that allows MSI files to be built up from XML definition files. In the Deployment Framework, all of the details of your BizTalk solution are automatically detailed out in an XML file, and WiX does the magic to pull all of the pieces together and create an MSI that is ready to move to your servers. The WiX processing is driven by MSBuild.

It is not necessary for you to know how to use WiX – just be aware that it is the engine behind the MSI creation process that is built into the Deployment Framework.

Environment Settings Manager

Most of you have probably struggled with the task of managing numerous runtime settings that vary across your deployment environments. For example, your solution calls a web service to obtain some data, and your development environment uses one URL, while your test and production environments use two more distinct URL's. Your BizTalk binding files are a perfect example of this problem.

The Deployment Framework builds upon an open-source tool known as the Environment Settings Manager to help tackle this complexity. The core of the solution is a simple, easy-to-use Excel workbook. In this workbook, you may list out all of your runtime environments and all of your runtime settings in a simple matrix that also contains the actual values for each combination. This gives you instant documentation and a simple method with which to maintain your settings and their values.

Using the Deployment Framework, when you deploy to a particular environment the Framework will automatically extract just that environment's settings and values from the spreadsheet and deploy them to the BizTalk SSO database. An included .NET class library allows easy access to the settings from your BizTalk solution at runtime. You may also automatically extract setting values from the spreadsheet and utilize them anywhere in your Deployment Framework MSBuild build file.

All you need to get started with the Environment Settings Manager workbook is Excel 2003 or newer.

Working in Visual Studio

The Deployment Framework for BizTalk's Visual Studio integration makes it easy to deploy and undeploy your BizTalk applications and create new Deployment Framework for BizTalk projects. It also provides commands that help speed up your BizTalk development cycle.

NOTE: Most commands require elevated permissions, so you must run Visual Studio "as administrator" on Windows Vista or newer operating systems.

In this section

Features of the Visual Studio Add-in

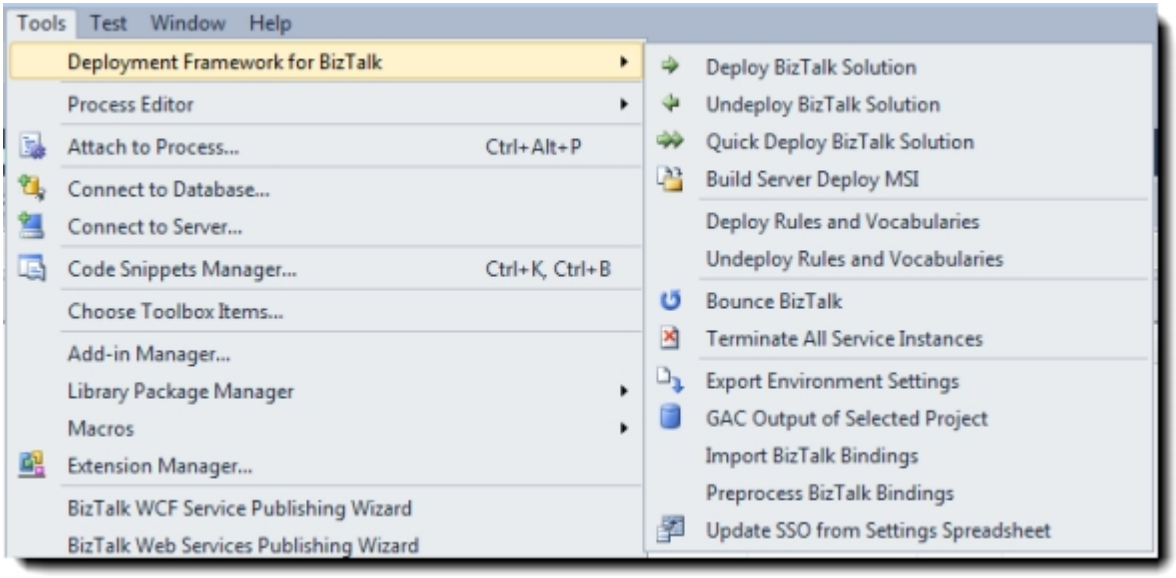
Most of your time as a BizTalk developer is spent in Visual Studio, so it makes sense that the Deployment Framework should integrate with the VS IDE. The add-in provides the following points of integration:

1. Commands that can be mapped to shortcut keys, toolbars or menus
2. Deployment Framework for BizTalk menu under the Tools menu
3. Deployment Framework for BizTalk toolbar
4. Activity logging to the Output window
5. Awareness of the solution status and active build configuration
6. Add New Project wizard to create a new deployment project
7. IntelliSense while editing .btdfproj project files

The add-in provides commands specific to deployment as well as some handy developer tools:

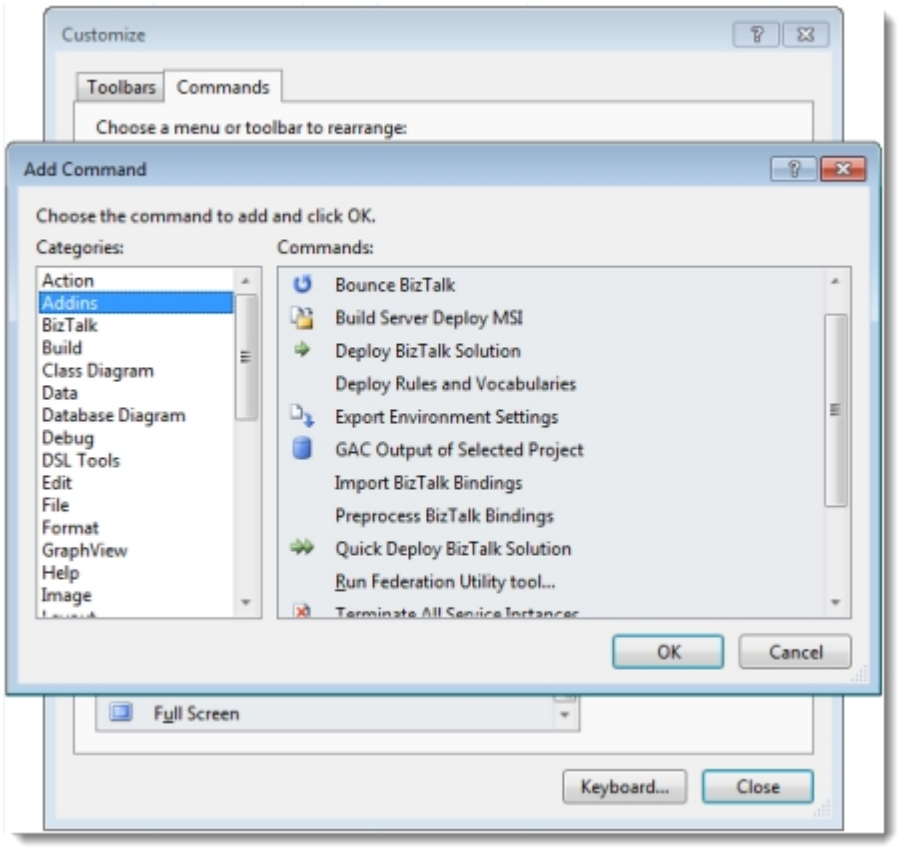
1. A command to deploy your solution
2. A command to un-deploy your solution
3. A command to deploy your solution quickly, useful when schemas and orchestration ports have not changed
4. A command to deploy or un-deploy BRE policies and vocabularies
5. A command to build a Windows Installer MSI for your solution
6. A command to export settings from your settings spreadsheet into XML files
7. A command to restart BizTalk host instances and optionally IIS or selected AppPools
8. A command to terminate all instances associated with the target BizTalk application

This screenshot shows the Deployment Framework for BizTalk menu open while a BizTalk solution is open in the IDE:

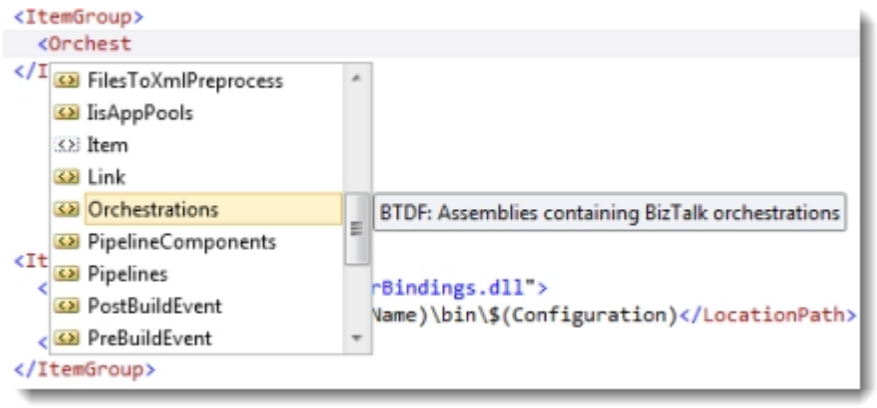


This screenshot shows the Deployment Framework for BizTalk toolbar while a BizTalk solution is open in the IDE:

The Deployment Framework commands may be mapped to shortcuts, menus or toolbars, as shown below in the Visual Studio 2010 Tools\Customize dialog:



This screenshot shows IntelliSense in action for a Deployment Framework MSBuild ItemGroup:

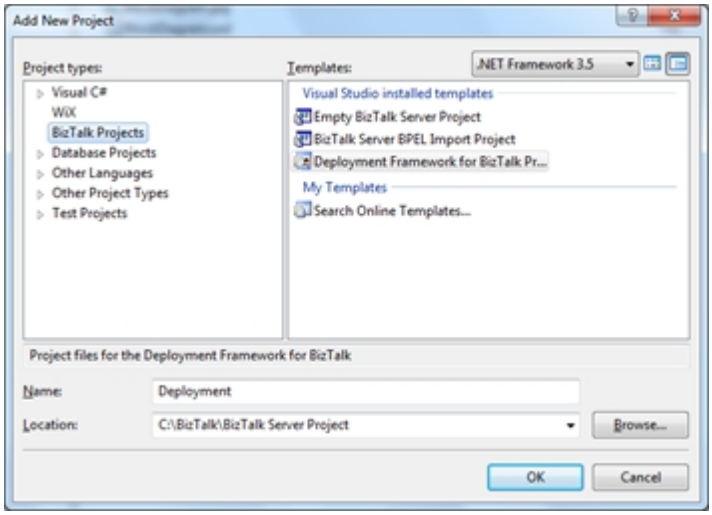


Using the Add New Project Wizard

The Deployment Framework for BizTalk includes a wizard that allows you to quickly create a new deployment project.

Create a New Deployment Project

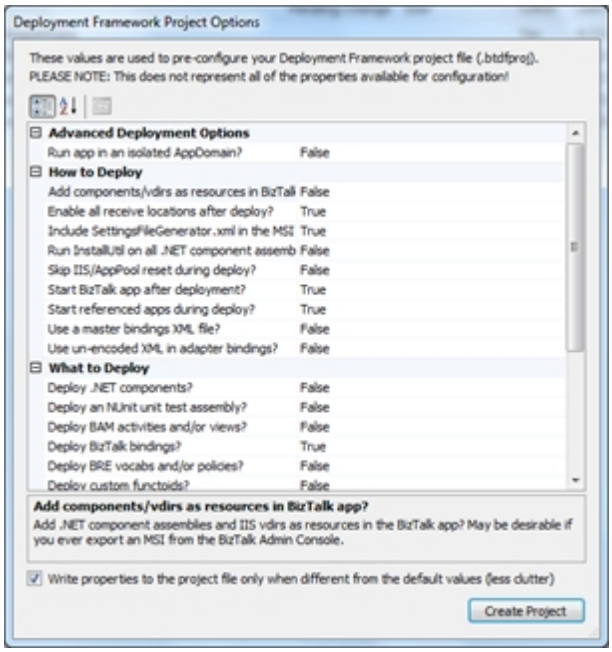
With your BizTalk solution open, right-click the solution and choose “Add New Project…” Select the “BizTalk Projects” node. Next to the familiar “Empty BizTalk Server Project” item, you'll find “Deployment Framework for BizTalk Project.”



Change the Name to Deployment from the default of Deployment1, then click OK.

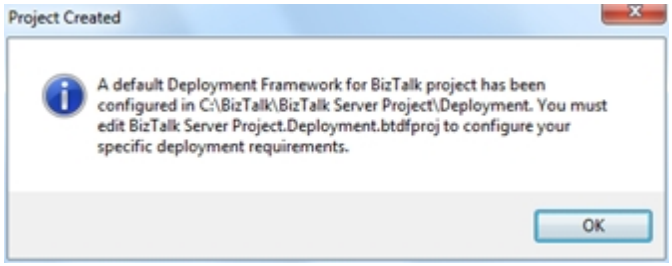
NOTE: Name the project either "Deployment" or "<solutionNameMinusExtension>.Deployment". Other names will not be recognized by the Visual Studio add-in. However, other names *are* valid when automating the Deployment Framework from scripts or the command line.

Next, you'll see the following dialog box which allows you to configure many of the options supported by the Deployment Framework:



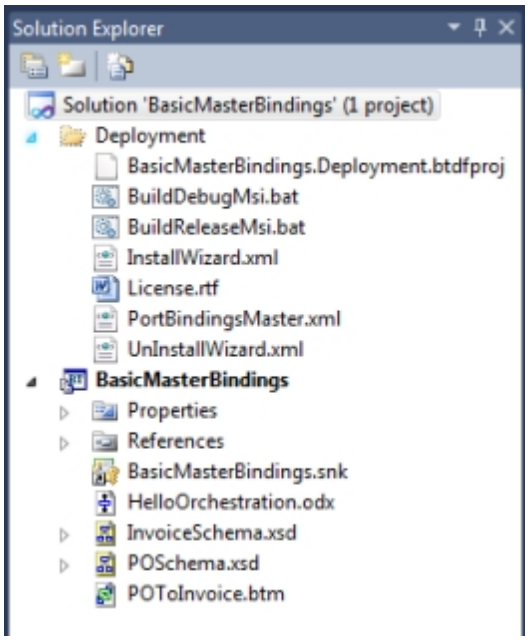
NOTE: This dialog box does not include every option available in the Deployment Framework. See the Technical Reference section for a complete list.

Once you have made your selections, click “Create Project.” You'll see the following dialog box when your project is ready.



NOTE: Since the Deployment Framework does not include a custom Visual Studio project type (like C#, BizTalk, etc.), the new files will not appear as a project in the solution. If desired, use the “Add Existing Item” feature on the solution to include the files from the deployment project folder in your solution.

Your final solution should look something like this (assuming that you add the deployment project files to your solution):



Edit the Generated Project Files

Your project is now ready for final customization. The new .btdfproj file should have automatically opened in Visual Studio's XML editor. You will need to configure ItemGroups for each artifact that you chose to deploy (orchestrations, schemas, ESB itineraries, etc.) so that the Deployment Framework knows how to find all of the files that need to be deployed. The Visual Studio XML Editor provides custom IntelliSense for the Deployment Framework for BizTalk project (.btdfproj) file.

NOTE: The project file generated by the wizard **must** be edited. At minimum, you'll need to specify the locations of your BizTalk project output(s). Look for the TODO section.

Deploy and Undeploy an Application

It's easy to deploy and undeploy your BizTalk application within Visual Studio. Once you've created a Deployment Framework for BizTalk project and edited it to point to your BizTalk artifacts, you're ready to go!

There are several ways to initiate a deployment or undeployment: menu, toolbar or command.

TIP: Consider enabling the "Show Output window when build starts" option in Visual Studio, located in the Tools\Options dialog under "Projects and Solutions." It will automatically show the Output window when you activate any Deployment Framework command, allowing you to easily see the output from the command.

You must have your BizTalk solution open in Visual Studio for the commands to be available for use.

To deploy or undeploy from the menu:

The *Deploy BizTalk Solution* option will deploy your application into the local BizTalk server. If the application is already deployed, then it will be undeployed and re-deployed. The Visual Studio Output window will display the complete output from the deployment process.

The *Undeploy BizTalk Solution* option will undeploy your application from the local BizTalk server. As with all Deployment Framework commands, the Output window will display the complete output.

The *Quick Deploy BizTalk Solution* option will perform an abbreviated deployment of your application into the local BizTalk server. This command is particularly useful in the application development cycle because it significantly cuts deployment time.

The *Quick Deploy* option is appropriate when you've made changes that are "internal" to your BizTalk artifacts and not directly visible to BizTalk.

Common changes that are **compatible** with this command include:

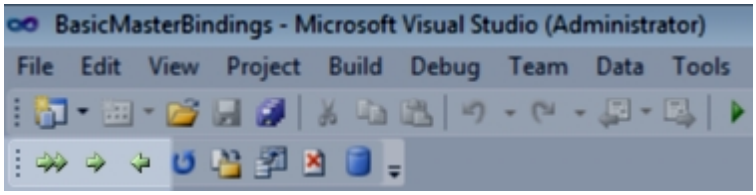
- changing Expression shape code
- adding or removing (most) shapes inside an orchestration
- changing code inside an existing .NET helper method
- changing links or functoids within an existing map

Examples of common changes that are visible to BizTalk and are **incompatible** with this command include:

- changing ports in an orchestration
- changing the source or destination schemas in a map
- changing schema namespace or root node properties
- changing a property schema

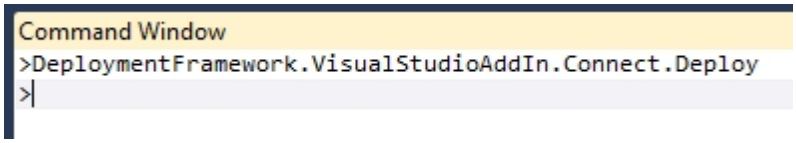
Use a bit of trial and error with this command -- give it a try and if you end up with an error during the deployment due to your change, or if your change does not take effect, then use the full *Deploy BizTalk Solution* command instead.

To deploy or undeploy from the toolbar:



The commands are identical to the menu commands described above. From left to right, the toolbar buttons represent *Quick Deploy BizTalk Solution*, *Deploy BizTalk Solution* and *Undeploy BizTalk Solution*.

To deploy or undeploy from the Command Window:

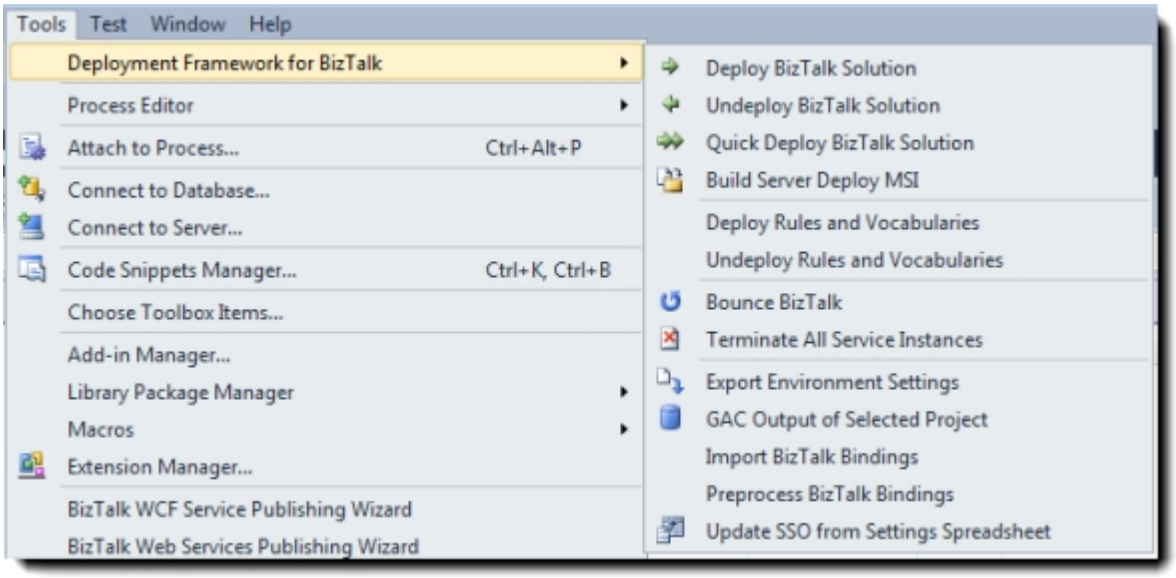


All of the Deployment Framework's commands are also available from the Visual Studio Command window. All of the commands begin with the prefix "DeploymentFramework".

Increase Productivity with Developer Tools

The Deployment Framework for BizTalk's Visual Studio add-in provides a number of commands besides application deploy/undeploy that help make your BizTalk development time more productive.

Here's a screenshot of the Deployment Framework for BizTalk menu:



As with the Deploy and Undeploy commands, the output of every command will appear in the Visual Studio Output window, and the commands are only available when your BizTalk solution is open and a Deployment Framework project folder exists below the solution root folder.

The **Bounce BizTalk** command will restart one or more local BizTalk host instances. The command respects the BizTalkHosts ItemGroup in your deployment project file (.btdfproj), which allows you to limit the hosts affected by host instance restarts.

The **Terminate All Service Instances** command will terminate all service instances associated with the current BizTalk application (the one your deployment project targets). The most common usage is to terminate all suspended messages in the application so that you can undeploy it.

The **GAC Output of Selected Project** command will install (or reinstall) in the GAC the .NET assembly built by the project currently selected in Solution Explorer. The command respects the current solution build configuration (Debug, Release, etc.).

The **Export Environment Settings** command will export one environment-specific XML file for each environment defined in your Excel settings spreadsheet, usually located in EnvironmentSettings \SettingsFileGenerator.xml. That process always happens during deployment, so this is not a commonly used command.

The **Import BizTalk Bindings** command allows you to quickly re-import your project's bindings XML file. This can be useful if you've accidentally altered a port configuration or orchestration binding.

The **Preprocess BizTalk Bindings** command, which is only useful when using a master (template) bindings XML file, will preprocess the master bindings XML file (usually PortBindingsMaster.xml) to produce the final bindings file (usually PortBindings.xml). That process always happens during deployment, so this command is often used for testing.

The **Update SSO from Settings Spreadsheet** command, which is only useful when IncludeSSO is set to True, will load the most current environment settings XML into SSO for the *Local* environment. The environment settings XML file originates with the Excel settings spreadsheet mentioned above.

Limitations of the Add-in

Only one Deployment Framework project within a single solution

The Visual Studio add-in can only recognize a single Deployment Framework for BizTalk project within the BizTalk solution. The Deployment Framework itself does not impose any restrictions on how many deployment projects you create or their locations relative to the solution. However, the add-in will only look for a .btdfproj file in several predefined locations under the solution root directory:

- <solutionNameNoExtension>.Deployment\<solutionNameNoExtension>.Deployment.btdfproj
- Deployment\<solutionNameNoExtension>.Deployment.btdfproj
- Deployment\Deployment.btdfproj

Deployment Framework project does not appear in Solution Explorer like BizTalk and C# projects

Ideally, Deployment Framework for BizTalk projects would appear in Solution Explorer and act like other project types, with property pages, project references, etc. Unfortunately, it is extremely difficult to create a full-blown custom project type for Visual Studio. It would probably take hundreds of hours to create a custom project type, which would consume virtually all project resources for a long time.

The best answer today is to use the Add/Solution Folder option to create a virtual solution folder in your BizTalk solution named "Deployment" (or whatever you named your project folder), then add all of the files in the deployment project folder to the solution folder.

Configuring BizTalk Artifacts for Deployment

The Deployment Framework for BizTalk natively supports the majority of BizTalk artifacts within an easily extensible model. This section takes you through the basics of setting up your Deployment Framework project, understanding the project file format and configuring each type of BizTalk artifact.

In this section

Naming Conventions

The Deployment Framework for BizTalk may be used with existing or newly created BizTalk solutions, and it works fine with single-project or multi-project solutions. By default, many advanced features are disabled and default behaviors are in effect. As you become more comfortable with the Deployment Framework, you can take advantage of more of the functionality that it provides.

Naming Conventions

This section explains how to structure and reference your BizTalk projects and artifacts for use with the Deployment Framework.

Solution Configuration Name Change for BizTalk 2006/2006 R2 (Required)

There is one naming convention that is not optional, but applies only to BizTalk 2006 or 2006 R2: the solution configuration names for BizTalk projects must be "Debug" and "Release".

The default solution configuration names are often set to Development and Deployment. These must be renamed to Debug and Release using the Configuration Manager dialog, available when the solution is open in Visual Studio (right-click the solution). This makes them consistent with standard .NET projects.

Alternative 1: After you have created all of your projects, close your solution and use a file-level search and replace utility to rename the "Development" and "Deployment" configurations within your BTPROJ files to "Debug" and "Release", respectively. After you are done, open the solution again and go to the Configuration Manager. Select "Debug" as the "Active Solution Configuration" and then select "Debug" in the "Configuration" column for all of your projects. Likewise for "Release".

Alternative 2: Change the template for BizTalk projects, found in BTSApp.btproj file in %BizTalkInstallDir% \Developer Tools\BizTalkWizards\BTSApp\Templates\1033, to reflect Debug and Release targets.

Optional Naming Conventions

The following naming conventions are all optional, but serve the purpose of simplifying your deployment project configuration and/or using accepted patterns for BizTalk development.

> **NOTE**: These naming conventions were more or less required in prior releases of the Deployment Framework for BizTalk, but that is **no longer the case**. Feel free to restructure/reorganize your old solutions if you wish.

If you follow these conventions, your deployment project file (.btdfproj) will be a bit shorter and simpler because the Deployment Framework can automatically identify the paths to your schemas, orchestrations

and other projects.

If you do not follow these conventions, don't worry – you will just need to include a bit more configuration in your deployment project file.

1. The name of the folder containing a BizTalk project is identical to the project's output DLL name (without the extension).
For example, if a BizTalk project resides in a folder named MyProject.Orchestrations, then that project must output a DLL named MyProject.Orchestrations.dll.

2. Each project must build to a bin\Debug and bin\Release folder under the project folder.
For example, if you have a BizTalk project in a folder named MyProject.Orchestrations, then it must build to MyProject.Orchestrations\bin\Debug\MyProject.Orchestrations.dll (or bin\Release). This is the default for BizTalk 2009, but requires a change to the default project configuration for BizTalk 2006 and 2006 R2.

3. Split your BizTalk projects by artifact type: a project for orchestrations, a project for schemas, a project for maps and so on.
This model is built into the Deployment Framework and is the preferred structure. However, if your project isn't set up this way, don't worry – the Deployment Framework is extremely flexible and supports even single-project solutions.

A typical example of this project naming structure is:
1. MyProject.Orchestrations
2. MyProject.Schemas
3. MyProject.Transforms
4. MyProject.Pipelines
5. MyProject.Components (for ordinary .NET components)

The Deployment Framework pre-defines the folder names for orchestrations, schemas, etc. as $(ProjectName).Orchestrations, $(ProjectName).Schemas, etc. You'll find the definition for the ProjectName property near the top of your .btdfproj file. This naming convention can be easily changed. If you have a different naming scheme or are working with an existing solution, then you can explicitly specify the DLL names and locations for each of your BizTalk artifacts.

Using these optional conventions, and taking a schemas assembly as an example, the Deployment Framework essentially fills in the following ItemGroup automatically (you would not need to put this in your .btdfproj file):

```
<ItemGroup>
  <Schemas Include="$(ProjectName).Schemas.dll">
    <LocationPath>..\$(ProjectName).Schemas\bin\$(Configuration)</LocationPath>
  </Schemas>
</ItemGroup>
```

Remember that all of these naming conventions are optional!

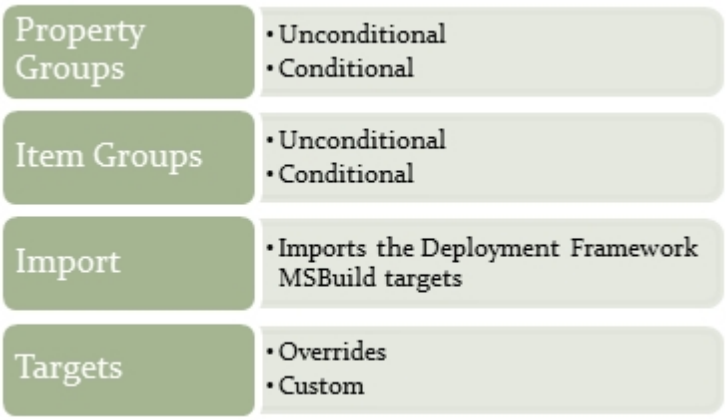
Project File Structure

Just like BizTalk (.btproj), C# (.csproj) or VB.NET (.vbproj) projects, the Deployment Framework for BizTalk is based around a project file in MSBuild XML format. A Deployment Framework for BizTalk project file carries a **.btdfproj** extension and can be edited with any text or XML editor.

> **TIP**: Use the Visual Studio XML editor to edit your .btdfproj files, because the Deployment Framework's extensions to IntelliSense will guide you as you edit the file.

Let's look at the structure of the XML in the project file. The project always declares a single <Project>

element, and within it is found, in order, the following elements:



The property groups (one or more <PropertyGroup> elements) define simple MSBuild properties. A property is defined like this:

```
<PropertyGroup>
  <MyPropertyName>MyPropertyValue</MyPropertyName>
</PropertyGroup>
```

One property group may contain any number of property declarations.

Entire property groups may be made conditional on MSBuild properties. For example, to include a property group only during a debug build:

```
<PropertyGroup Condition="'$(Configuration)' == 'Debug'">
</PropertyGroup>
```

The text $(Configuration) is a property reference. MSBuild replaces it with the value of the property at runtime.

Item groups (one or more <ItemGroup> elements) define MSBuild items, which are typically used for variable-length lists of items. Each item in an item group may also contain zero or more metadata elements. In the following example, the Orchestrations item group has a metadata element LocationPath:

```
<ItemGroup>
  <Orchestrations Include="MyOrchestrations.dll">
    <LocationPath>..\Orchestrations\bin\$(Configuration)</LocationPath>
  </Orchestrations>
</ItemGroup>
```

As with property groups, item groups may also be conditional on MSBuild properties:

```
<ItemGroup Condition="'$(Configuration)' == 'Debug'">
</ItemGroup>
```

After the property and item group definitions, you must include an <Import> element that brings in the Deployment Framework's own MSBuild property, item and target definitions. This is already part of the default project file created by the Add New Project wizard.

```
<Import Project="$(DeploymentFrameworkTargetsPath)BizTalkDeploymentFramework.targets" />
```

The final section of the project file may contain zero or more MSBuild target definitions (<Target> elements). These are like subroutines that the MSBuild engine can execute. In many cases you will not need to include any targets, but if you need to do some custom work that is not handled by the Deployment Framework out of the box, this is where you can do it. This is an example of overriding one of the targets pre-defined by the Deployment Framework:

```
<Target Name="CustomDeployTarget">
</Target>
```

Everything that has been discussed in this section is standard MSBuild syntax. For more information, including how to implement your own custom targets, please see the MSDN documentation or the many online and print resources for MSBuild.

ItemGroup Structure

The Deployment Framework for BizTalk uses MSBuild ItemGroup elements to identify and locate all of the files that make up your BizTalk solution. Most of the common artifact types use the same structure, including those for orchestrations, schemas, transforms/maps, pipelines and more.

Typical ItemGroup structure

Most artifact types use a standard ItemGroup structure, shown below:

```
<ItemGroup>
  <ArtifactName Include="MyFilename.ext">
    <LocationPath>..\ProjectName\bin\$(Configuration)</LocationPath>
  </ArtifactName>
  <ArtifactName Include="MyFilename2.ext">
    <LocationPath>..\ProjectName2\bin\$(Configuration)</LocationPath>
  </ArtifactName>
  ...
</ItemGroup>
```

ArtifactName is a placeholder for the actual artifact name, such as Orchestrations, Schemas, Pipelines, etc. One *ArtifactName* element corresponds to exactly one physical file.

The ***Include*** attribute specifies the file name without path information.

The ***LocationPath*** element text specifies the path to the file named in the *Include* attribute, and is **always** resolved starting in the directory containing the .btdfproj file. Most of the time the *LocationPath* value should be a relative path within the BizTalk solution structure, but that is not a restriction.

While most ItemGroups reference files in the file system, some do not. They include VDirList, BizTalkHosts, AppsToReference, PropsFromEnvSettings and IisAppPools.

Properties within an ItemGroup

Element values within an ItemGroup are free to use any MSBuild property in the standard $(PropertyName) format. The most commonly used property is $(Configuration), which contains the name of the active solution configuration (commonly Debug or Release). Another common property is $(ProjectName), which is defined within your project file.

Any MSBuild property that you reference must be defined at runtime. It may be defined by being passed in via command-line to MSBuild, explicitly defined within your .btdfproj file, pre-defined by the Deployment Framework or created at runtime from a setting in the environment settings spreadsheet (SettingsFileGenerator.xml) via the PropsFromEnvSettings ItemGroup.

Best Practices

Here are a few best practices related to ItemGroup elements:

- Use a new ItemGroup element for each type of item (i.e. Orchestrations, Schemas, etc.)
- Place the ItemGroup **after** any PropertyGroup elements and **before** Import elements
- Use relative paths for LocationPath whenever possible

Artifact Types

This section describes how to configure the various types of BizTalk artifacts supported by the Deployment Framework for BizTalk.

In this section

.NET Assemblies

To deploy one or more assemblies that contain custom .NET code (a.k.a. components), edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeComponents property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeComponents>true</IncludeComponents>
  ...
</PropertyGroup>
```

2. Add a Components ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <Components Include="$(ProjectName).Components.dll">
    <LocationPath>..\Components\bin\$(Configuration)</LocationPath>
  </Components>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any Components elements in your project file.

```
<ItemGroup>
  <Components Include="$(ProjectName).Components.dll">
    <LocationPath>..\$(ProjectName).Components\bin\$(Configuration)</LocationPath>
  </Components>
```

```
</ItemGroup>
```

If you have more than one assembly that contains custom .NET code, then repeat the <Components> element for each assembly (usually within the same ItemGroup).

NOTE: If you are deploying a third-party assembly that must be deployed to the GAC but does not have source code within your solution, then place it in an ExternalAssemblies ItemGroup.

BAM Activities and Views

BizTalk's Business Activity Monitoring (BAM) model consists of activities, views and security associated with the views. The activities and views translate into physical SQL Server database artifacts, which makes it difficult to change a model once it has been deployed and real data is stored in it.

Model Deployment

The first thing that is required to deploy a BAM model is the model definition itself. The model should be saved as an Excel XLS/XLSX binary file. The Deployment Framework will automatically extract and export the BAM XML definition from the Excel workbook, so *do not store the XML file in your source control system*.

To deploy one or more BAM models, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeBAM property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeBAM>true</IncludeBAM>
  ...
</PropertyGroup>
```

2. Add a BamDefinitions ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <BamDefinitions Include="$(ProjectName).BAM.xls">
    <LocationPath>..\BAM</LocationPath>
  </BamDefinitions>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any BamDefinitions elements in your project file.

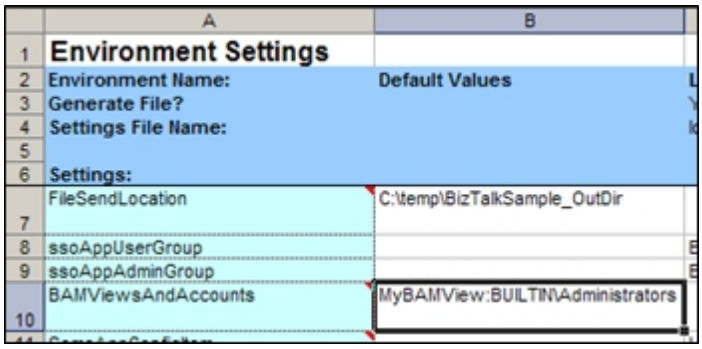
```
<ItemGroup>
  <BamDefinitions Include="$(ProjectName).BAM.xls">
    <LocationPath>..\$(ProjectName).BAM</LocationPath>
  </BamDefinitions>
</ItemGroup>
```

If you have more than one BAM model, then repeat the <BamDefinitions> element for each file (usually within the same ItemGroup).

Continue reading for Steps 3 and 4 (below).

Security for Views

Before users can view data from a BAM model in the BAM Portal website, they must be granted access to the views. The settings spreadsheet (typically EnvironmentSettings\SettingsFileGenerator.xml) contains a row titled "BAMViewsAndAccounts" that allows you to specify one or more view names and one or more accounts to be granted access to each view. As with the other settings, this value may be different for each runtime environment. After deploying the BAM model, the Deployment Framework will apply security to the views based on this setting.



	A	B
1	**Environment Settings**	
2	**Environment Name:**	**Default Values**
3	**Generate File?**	
4	**Settings File Name:**	
5		
6	**Settings:**	
7	FileSendLocation	C:\temp\BizTalkSample_OutDir
8	ssoAppUserGroup	
9	ssoAppAdminGroup	
10	BAMViewsAndAccounts	MyBAMView:BUILTIN\Administrators

NOTE: On an application re-deploy when the BAM model already exists, it is normal to see an error message during this phase of deployment. The error is simply reporting that the specified users already have rights to the views and will not interrupt the script.

3. Configure permissions for the views in the settings spreadsheet
In the BAMViewsAndAccounts row in the spreadsheet, enter a formatted permissions string for each environment, and/or enter a default value in the Default Values column. If there is no such row, just enter BAMViewsAndAccounts in the first column of an empty row and proceed to enter the values.

The format of the values is:
ViewName1:DOMAIN\GroupName1;ViewName1:DOMAIN\GroupName2;ViewName2:BUILTIN\Administrators;<etc.>

4. Add (or update) a PropsFromEnvSettings ItemGroup
In your project file (.btdfproj), add to or update a PropsFromEnvSettings ItemGroup to cause the BAMViewsAndAccounts setting from the spreadsheet to be transferred to an MSBuild property at runtime:

```
<ItemGroup>
  <PropsFromEnvSettings Include="BAMViewsAndAccounts" />
</ItemGroup>
```

If you need more than one setting value to be transferred to MSBuild properties, you may include one or more settings spreadsheet row names in the Include attribute value separated by semicolons, or include multiple PropsFromEnvSettings elements.

Data Preservation

Once a BAM model has been deployed and live data has been stored in it, very few changes are possible without data loss. For that reason, the Deployment Framework does not automatically un-deploy the model. If you still prefer to have automatic un-deployment, set the following property in your project file:

```
<PropertyGroup>
  ...
  <SkipBamUndeploy>false</SkipBamUndeploy>
  ...
</PropertyGroup>
```

You can also remove the model from the command line using the following command in the deployment project folder:

```
msbuild /t:UndeployBam /p:SkipBamUndeploy=false MyDeploymentProject.btdfproj
```

ESB Toolkit Itineraries

The Deployment Framework can automatically deploy your ESB itineraries during deployment. Before an itinerary can be deployed, you must manually export it to an XML file using the Itinerary Designer's XML Exporter. The Deployment Framework must be directed to the *XML file, not the .itinerary file*.

> **NOTE**: You **MUST** re-export your itinerary any time it is modified. Unfortunately, there is currently no tool in the ESB Toolkit that can automate the export.

To deploy one or more ESB itineraries, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeEsbItineraries property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeEsbItineraries>true</IncludeEsbItineraries>
  ...
</PropertyGroup>
```

2. Add an EsbItineraries ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <EsbItineraries Include="MyItinerary1.xml">
    <LocationPath>..\Itineraries</LocationPath>
  </EsbItineraries>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any EsbItineraries elements in your project file.

```
<ItemGroup>
  <EsbItineraries Include="$(ProjectName).itinerary">
    <LocationPath>..\$(ProjectName).ESB</LocationPath>
  </EsbItineraries>
</ItemGroup>
```

If you have more than one ESB itinerary, then repeat the <EsbItineraries> element for each file (usually within the same ItemGroup).

The Microsoft ESB Toolkit does not include a tool to **un**deploy itineraries, so the Deployment Framework cannot undeploy them. Once deployed, they remain deployed.

Functoids

This topic assumes that you are following a common convention in BizTalk development where each assembly contains a single type of BizTalk artifact (e.g. an assembly for schemas, an assembly for orchestrations, etc.). If you are combining multiple types of BizTalk artifacts into a single assembly, see Assemblies with Multiple Artifact Types.

To deploy one or more assemblies that contain custom functoids, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeCustomFunctoids property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeCustomFunctoids>true</IncludeCustomFunctoids>
  ...
</PropertyGroup>
```

2. Add a CustomFunctoids ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <CustomFunctoids Include="$(ProjectName).CustomFunctoids.dll">
    <LocationPath>..\CustomFunctoids\bin\$(Configuration)</LocationPath>
  </CustomFunctoids>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any CustomFunctoids elements in your project file.

```
<ItemGroup>
  <CustomFunctoids Include="$(ProjectName).CustomFunctoids.dll">
    <LocationPath>..\$(ProjectName).CustomFunctoids\bin\$(Configuration)</LocationPath>
  </CustomFunctoids>
</ItemGroup>
```

If you have more than one assembly that contains custom functoids, then repeat the <CustomFunctoids> element for each assembly (usually within the same ItemGroup).

IIS Virtual Directories

The Deployment Framework for BizTalk includes support for deploying application pools and virtual directories in IIS. Virtual directory and application pool deployment is fully active for server deployments. For deployments from Visual Studio, it is assumed that the developer has manually configured an application pool.

NOTE: On Windows Server 2008 and above, this feature requires installation of these IIS 6 Management Compatibility role services: IIS Metabase and IIS 6 configuration compatibility, IIS 6 WMI Compatibility and IIS 6 Scripting Tools.

To deploy one or more IIS virtual directories, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeVirtualDirectories property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeVirtualDirectories>true</IncludeVirtualDirectories>
  ...
</PropertyGroup>
```

2. Add a VDirList ItemGroup
The following is a typical example:

```
<ItemGroup>
  <VDirList Include="*">
    <Vdir>MyVDirName</Vdir>
    <Physdir>..\MyVDir</Physdir>
    <AppPool>MyAppPool</AppPool>
    <AppPoolNetVersion>v4.0</AppPoolNetVersion>
  </VDirList>
</ItemGroup>
```

If you need more than one virtual directory, then repeat the <VDirList> element for each virtual directory (usually within the same ItemGroup).

The **Vdir** element value is the name given to the virtual directory.

The **Physdir** element value is a relative path from the deployment project folder to the virtual directory physical path. When the project is built into an MSI, the entire contents of that folder will be automatically packaged into the MSI.

The **AppPool** element value is the name given to the application pool (IIS 6+ only). For server deployments, the AppPool is created with a user identity specified by the VDIR_UserName and VDIR_UserPass MSBuild properties. These property values are normally collected by the deployment wizard, which is driven by the project file InstallWizard.xml.

The optional **AppPoolNetVersion** element value configures the .NET Framework version on the application pool (IIS 7+ only). Valid values are v2.0 or v4.0.

Related MSBuild Properties

- AppPoolAccount
- IISMetabasePath
- IISResetTime
- ModifyNTFSPermissionsOnVDirPaths
- SkipIISReset
- UndeployIISArtifacts
- VDIR_UserName
- VDIR_UserPass
- WseExtensionPath

Maps/Transforms

This topic assumes that you are following a common convention in BizTalk development where each assembly contains a single type of BizTalk artifact (e.g. an assembly for schemas, an assembly for orchestrations, etc.). If you are combining multiple types of BizTalk artifacts into a single assembly, see Assemblies with Multiple Artifact Types.

To deploy one or more assemblies that contain maps (a.k.a. transforms), edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeTransforms property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeTransforms>true</IncludeTransforms>
  ...
</PropertyGroup>
```

2. Add a Transforms ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <Transforms Include="$(ProjectName).Transforms.dll">
    <LocationPath>..\Transforms\bin\$(Configuration)</LocationPath>
  </Transforms>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any Transforms elements in your project file.

```
<ItemGroup>
  <Transforms Include="$(ProjectName).Transforms.dll">
    <LocationPath>..\$(ProjectName).Transforms\bin\$(Configuration)</LocationPath>
  </Transforms>
</ItemGroup>
```

If you have more than one assembly that contains maps, then repeat the <Transforms> element for each assembly (usually within the same ItemGroup).

Orchestrations

This topic assumes that you are following a common convention in BizTalk development where each assembly contains a single type of BizTalk artifact (e.g. an assembly for schemas, an assembly for orchestrations, etc.). If you are combining multiple types of BizTalk artifacts into a single assembly, see Assemblies with Multiple Artifact Types.

To deploy one or more assemblies that contain orchestrations, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeOrchestrations property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeOrchestrations>true</IncludeOrchestrations>
  ...
</PropertyGroup>
```

2. Add an Orchestrations ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <Orchestrations Include="$(ProjectName).Orchestrations.dll">
    <LocationPath>..\Orchestrations\bin\$(Configuration)</LocationPath>
  </Orchestrations>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any Orchestrations elements in your project file.

```
<ItemGroup>
  <Orchestrations Include="$(ProjectName).Orchestrations.dll">
    <LocationPath>..\$(ProjectName).Orchestrations\bin\$(Configuration)</LocationPath>
  </Orchestrations>
</ItemGroup>
```

If you have more than one assembly that contains orchestrations, then repeat the <Orchestrations> element for each assembly (usually within the same ItemGroup).

Pipeline Components

This topic assumes that you are following a common convention in BizTalk development where each assembly contains a single type of BizTalk artifact (e.g. an assembly for schemas, an assembly for orchestrations, etc.). If you are combining multiple types of BizTalk artifacts into a single assembly, see Assemblies with Multiple Artifact Types.

To deploy one or more assemblies that contain custom pipeline components, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludePipelineComponents property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludePipelineComponents>true</IncludePipelineComponents>
  ...
</PropertyGroup>
```

2. Add a PipelineComponents ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <PipelineComponents Include="$(ProjectName).PipelineComponents.dll">
    <LocationPath>..\PipelineComponents\bin\$(Configuration)</LocationPath>
  </PipelineComponents>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any PipelineComponents elements in your project file.

```
<ItemGroup>
  <PipelineComponents Include="$(ProjectName).PipelineComponents.dll">
    <LocationPath>..\$(ProjectName).PipelineComponents\bin\$(Configuration)</
LocationPath>
  </PipelineComponents>
</ItemGroup>
```

If you have more than one assembly that contains pipeline components, then repeat the <PipelineComponents> element for each assembly (usually within the same ItemGroup).

Pipelines

This topic assumes that you are following a common convention in BizTalk development where each assembly contains a single type of BizTalk artifact (e.g. an assembly for schemas, an assembly for

orchestrations, etc.). If you are combining multiple types of BizTalk artifacts into a single assembly, see Assemblies with Multiple Artifact Types.

To deploy one or more assemblies that contain pipelines, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludePipelines property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludePipelines>true</IncludePipelines>
  ...
</PropertyGroup>
```

2. Add a Pipelines ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <Pipelines Include="$(ProjectName).Pipelines.dll">
    <LocationPath>..\Pipelines\bin\$(Configuration)</LocationPath>
  </Pipelines>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any Pipelines elements in your project file.

```
<ItemGroup>
  <Pipelines Include="$(ProjectName).Pipelines.dll">
    <LocationPath>..\$(ProjectName).Pipelines\bin\$(Configuration)</LocationPath>
  </Pipelines>
</ItemGroup>
```

If you have more than one assembly that contains pipelines, then repeat the <Pipelines> element for each assembly (usually within the same ItemGroup).

Rule Policies and Vocabularies

The Deployment Framework includes comprehensive support for BizTalk Rules Engine (BRE) policies and vocabularies. Deployments are based on policy and vocabulary XML files exported from the rules engine.

> **NOTE**: You **MUST** re-export your policies and/or vocabularies any time they are modified.

To deploy one or more rule policies and/or vocabularies, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeVocabAndRules property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeVocabAndRules>true</IncludeVocabAndRules>
  ...
</PropertyGroup>
```

2. Add a RulePolicies ItemGroup, a RuleVocabularies ItemGroup or both
The following are typical examples that properly follow the common ItemGroup structure:

```
<ItemGroup>
  <RulePolicies Include="MyFirstPolicy_v1.xml">
    <LocationPath>..\BRE\Policies</LocationPath>
  </RulePolicies>
</ItemGroup>

<ItemGroup>
  <RuleVocabularies Include="MyFirstVocabulary_v1.xml">
    <LocationPath>..\BRE\Vocabularies</LocationPath>
  </RuleVocabularies>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any RulePolicies or RuleVocabularies elements in your project file.

```
<ItemGroup>
  <RulePolicies Include="$(ProjectName).Policies.xml">
    <LocationPath>..\$(ProjectName).BRE</LocationPath>
  </RulePolicies>
</ItemGroup>

<ItemGroup>
  <RuleVocabularies Include="$(ProjectName).Vocabularies.xml">
    <LocationPath>..\$(ProjectName).BRE</LocationPath>
  </RuleVocabularies>
</ItemGroup>
```

If you have more than one policy or vocabulary file, then repeat the corresponding element for each file (usually within the same ItemGroup).

Rule policies and vocabularies are published to BizTalk with the command-line tool DeployBTRules.exe in the Deployment Framework's Framework\DeployTools folder.

Related MSBuild Properties

- ExplicitlyDeployRulePoliciesOnDeploy
- RemoveRulePoliciesFromAppOnUndeploy

Schemas

This topic assumes that you are following a common convention in BizTalk development where each assembly contains a single type of BizTalk artifact (e.g. an assembly for schemas, an assembly for orchestrations, etc.). If you are combining multiple types of BizTalk artifacts into a single assembly, see Assemblies with Multiple Artifact Types.

To deploy one or more assemblies that contain schemas, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeSchemas property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeSchemas>true</IncludeSchemas>
  ...
```

```
</PropertyGroup>
```

2. Add a Schemas ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <Schemas Include="$(ProjectName).Schemas.dll">
    <LocationPath>..\Schemas\bin\$(Configuration)</LocationPath>
  </Schemas>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any Schemas elements in your project file.

```
<ItemGroup>
  <Schemas Include="$(ProjectName).Schemas.dll">
    <LocationPath>..\$(ProjectName).Schemas\bin\$(Configuration)</LocationPath>
  </Schemas>
</ItemGroup>
```

If you have more than one assembly that contains schemas, then repeat the <Schemas> element for each assembly (usually within the same ItemGroup).

Assemblies with Multiple Artifact Types

While the recommended structure for a BizTalk solution is to divide assemblies by artifact type, that is not always desired or practical for small projects. The Deployment Framework for BizTalk supports any method of packaging your BizTalk assemblies.

If you are combining multiple types of BizTalk artifacts into a single assembly, there are a couple of things to know:

- Schemas are critical to so many other BizTalk artifacts that they are always deployed first. If your assembly contains schemas along with any other type of artifact, then *treat the entire assembly as a Schemas assembly*.
- If your entire BizTalk solution is in a single assembly, or if you do not have any separate assemblies that contain orchestrations or transforms, then you must set the IncludeOrchestrations and IncludeTransforms properties to false.

Dynamic Configuration and Bindings

This section describes the Deployment Framework for BizTalk's powerful configuration/settings management features and how to take advantage of them in binding files, custom MSBuild targets and at runtime.

In this section

Overview of Dynamic Configuration

The Deployment Framework for BizTalk makes it easy for you to centrally manage all of the variables (connection strings, user names, passwords, file paths, URL's, etc.) that connect the pieces of your application together within each environment in which it runs. This is one of the most important and powerful features of the Deployment Framework.

Each Deployment Framework for BizTalk project includes an Excel spreadsheet that captures an unlimited number of "settings" (name/value pairs) stored as rows in the spreadsheet. Each setting (row) has a default value in addition to a value for any number of deployment environments (stored in columns). This makes it very easy to manage a large matrix of settings across environments in one place.

The screenshot below shows a sample spreadsheet:

	A	B	C	D	E	F
1	**Settings File Generator**		Press "Ctrl+W" to export settings files			
2	**Environment Name:**	**Default Values**	**Development**	**QA**	**Staging**	**Production**
3	**Generate File?**	-	Yes	Yes	No	Yes
4	**Settings File Name:**	-	DEV_settings.xml	QA_settings.xml	STG_settings.xml	PROD_settings.xml
5						
6	**Settings:**					
8	FileSendLocation	C:\temp\BizTalkSample_OutDir		\\tracesaturn_QA\c$\temp\BizTalkSample_OutDir	\\tracesaturn_Stage\c$\temp\BizTalkSample_OutDir	\\tracesaturn_Prod\c$\temp\BizTalkSample_OutDir
9	POAckHTTP	http://localhost/poack/btahttpreceive.dll				
10	CatalogFTPDrop_Server		devftp.myco.com	qaftp.myco.com	stgftp.myco.com	prodftp.myco.com
11	CatalogFTPDrop_Folder		catalog/drop	catalog/drop	catalog/drop	catalog/drop
12	CatalogDB_Address		SQL://DEVSvr/CatalogDB	SQL://QASvr/CatalogDB	SQL://StgSvr/CatalogDB	SQL://ProdSvr/CatalogDB
13	CatalogDB_ConnString		Persist Security Info=False;Integrated Security=SSPI;database=CatalogDB;server=DEVSvr	Persist Security Info=False;Integrated Security=SSPI;database=CatalogDB;server=QASvr	Persist Security Info=False;Integrated Security=SSPI;database=CatalogDB;server=StgSvr	Persist Security Info=False;Integrated Security=SSPI;database=CatalogDB;server=ProdSvr

These settings are directly supported in many contexts:

- Token-based replacements within XML or other files, including XML port bindings files
- MSBuild properties for use within the deployment project file
- Runtime access from maps, orchestrations, pipeline components and .NET code
- ESB itineraries for dynamic routing

Storing runtime configuration information for a BizTalk solution can be tricky, because traditional .NET configuration files are difficult (but not impossible) to use in this environment. Many people prefer to use the BizTalk SSO for this type of information. With the Deployment Framework, this problem becomes simple to solve.

It is worth noting that the sensitive information in the spreadsheet should (eventually) be managed by the operations team that has access to production environments, and stored on a secured file share that is referenced when deploying the application.

Deploy Configuration Settings into SSO

In order to make the settings in your Excel settings spreadsheet (SettingsFileGenerator.xml) available to your BizTalk application at runtime, they must be deployed into a BizTalk SSO affiliate application. Once deployed into SSO, the settings may be read dynamically at runtime within orchestrations, pipelines, maps and even ESB itineraries. One benefit of deploying your settings into SSO is that they are encrypted at rest, may be easily updated at runtime and are available from anywhere within the BizTalk group.

To deploy configuration settings into SSO, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the IncludeSSO property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeSSO>true</IncludeSSO>
  ...
</PropertyGroup>
```

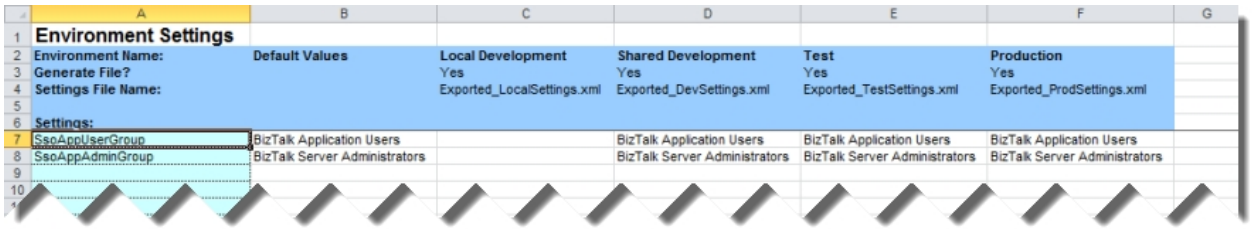
This will cause an XML settings file (generated from the settings spreadsheet) that is appropriate for the environment to be deployed to the BizTalk SSO database. The SSO affiliate application is named after the value in the ProjectName element in the deployment project file.

2. Ensure that you have a PropsFromEnvSettings ItemGroup containing SsoAppUserGroup and SsoAppAdminGroup
The Deployment Framework for BizTalk needs to know the SSO security group names at runtime in order to deploy into SSO. The PropsFromEnvSettings ItemGroup must contain the setting names SsoAppUserGroup and SsoAppAdminGroup (semicolon separated) in order to import the setting values from the settings spreadsheet into MSBuild properties at runtime. The PropsFromEnvSettings Include attribute may also include any other setting names that you wish to bring over into MSBuild properties.

```
<ItemGroup>
  <PropsFromEnvSettings Include="SsoAppUserGroup;SsoAppAdminGroup;AnotherSettingName" />
</ItemGroup>
```

3. Ensure that the Excel settings spreadsheet contains SsoAppUserGroup and SsoAppAdminGroup settings
The settings spreadsheet (normally SettingsFileGenerator.xml) must contain two settings rows, one named SsoAppUserGroup and the other named SsoAppAdminGroup. The Default Values column normally contains the values BizTalk Application Users and BizTalk Server Administrators, respectively. These defaults are appropriate when you are using BizTalk on a development machine where the BizTalk security groups are **local**, not domain-based. In the environment-specific columns, you must enter the **domain-qualified** security group names, such as MYDOMAIN\BizTalk Server Administrators.

Your spreadsheet should look something like this (you will probably have different environment columns):



	A	B	C	D	E	F	G
1	**Environment Settings**						
2	**Environment Name:**	**Default Values**	**Local Development**	**Shared Development**	**Test**	**Production**	
3	**Generate File?**		Yes	Yes	Yes	Yes	
4	**Settings File Name:**		Exported_LocalSettings.xml	Exported_DevSettings.xml	Exported_TestSettings.xml	Exported_ProdSettings.xml	
5							
6	**Settings:**						
7	SsoAppUserGroup	BizTalk Application Users		BizTalk Application Users	BizTalk Application Users	BizTalk Application Users	
8	SsoAppAdminGroup	BizTalk Server Administrators		BizTalk Server Administrators	BizTalk Server Administrators	BizTalk Server Administrators	
9							
10							

If you need runtime configuration with SSO-backed storage *without* putting the values in the settings file (or at least not for all environments) then consider using the UpdateSSOConfigItem MSBuild task to create or

update a particular name-value pair. This can be used in your project file in a Target named CustomSSO, and it can pull values from environment variables established by SetEnvUI or any other MSBuild properties.

Here is an example:

```
<Target Name="CustomSSO" Condition="'$(Configuration)' == 'Server'">
  <UpdateSSOConfigItem Ssoitemname="DatabaseUserName" Ssoitemvalue="$(DatabaseUserName)"/>
  <UpdateSSOConfigItem Ssoitemname="DatabasePassword" Ssoitemvalue="$(DatabasePassword)"/>
</Target>
```

After your application is deployed, you can find your affiliate application in the SSO Config Store using the BizTalk SSO Administration tool. However, you can not view or manipulate the settings with the SSO Administration tool. The Deployment Framework for BizTalk includes its own settings management tool, SSOSettingsEditor.exe, for deploy-time settings management.

Reading Configuration Settings at Runtime

Once you've added a setting to the Excel settings spreadsheet, it's readily accessible to your BizTalk application at runtime.

To read a setting value at runtime from maps, orchestrations, pipeline components or .NET code:

1. Complete the steps in Deploy Configuration Settings into SSO
The steps in Deploy Configuration Settings into SSO are all prerequisites to this topic.

2. Copy SSOSettingsFileReader.dll from the Deployment Framework for BizTalk install folder to a reference location in your BizTalk solution
The settings API is housed in a .NET 2.x assembly named SSOSettingsFileReader.dll. Before you can use it in your application, locate the DLL in <BTDFInstallFolder>\Framework\DeployTools and copy it to a folder within your BizTalk solution folder hierarchy. It's a good idea to check the DLL into your source control system and reference it in that location vs. the Deployment Framework's install folder.

3. Add a reference to SSOSettingsFileReader.dll to your project
In any BizTalk or .NET project, add a reference to the copy of SSOSettingsFileReader.dll in your solution folder structure.

4. Call the settings API
The SSOSettingsFileReader class has static methods for retrieving an individual value as well as for retrieving a Hashtable for the entire set of name-value pairs. This class caches the settings and periodically refreshes them from the SSO server.

The affiliateApplication is the value of the <ProjectName> element in your deployment project file (.btdfproj). The valueName is the name of a setting (row) in the spreadsheet.

```
namespace SSOSettingsFileManager
{
    public class SSOSettingsFileReader
    {
        public static void ClearCache(string affiliateApplication);
        public static Hashtable Read(string affiliateApplication);
        public static int ReadInt32(string affiliateApplication, string valueName);
        public static string ReadString(string affiliateApplication, string valueName);
        public static void Update(string affiliateApplication, Hashtable ht);
    }
}
```

HINT: The Deployment Framework's Advanced sample application uses the settings API.

Within an orchestration, you can either create a variable of type SSOSettingsFileManager.SSOSettingsFileReader **or** directly reference the static methods. Within a map, you may directly reference SSOSettingsFileReader from a Scripting functoid as a component from an external assembly.

Using the BTDF ESB Resolver

The Deployment Framework includes a custom ESB resolver (BTDF-SSO) that can be selected in the Itinerary Designer. It lets you dynamically resolve settings via SSO at runtime that originate from your Excel settings spreadsheet (SettingsFileGenerator.xml).

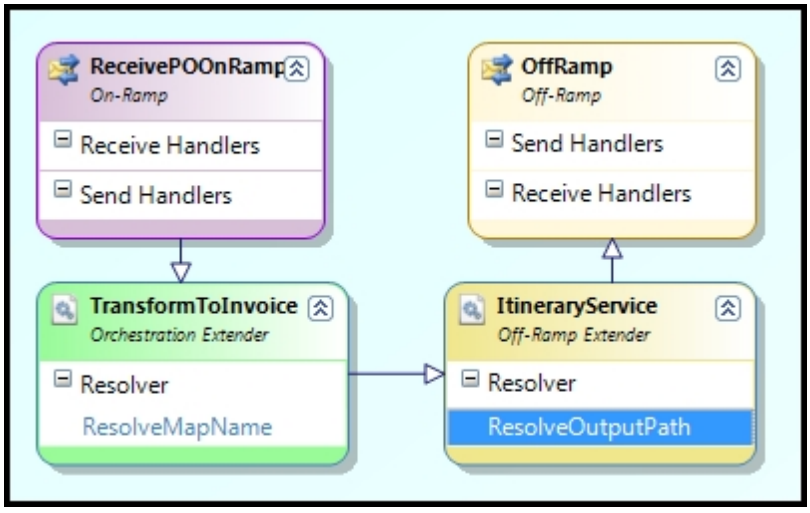
NOTE: If you are using the BTDF-SSO resolver, then you must do a Custom install of the Deployment Framework on your BizTalk servers. This is due to the fact that the resolver DLL must be installed on the servers and registered in the Esb.config file.

NOTE: If you are using BizTalk 2010 or newer, remember to install the BTDF Itinerary Designer Extension. See instructions here.

To use the BTDF ESB Resolver:

1. Complete the steps in Deploy Configuration Settings into SSO
The steps in Deploy Configuration Settings into SSO are all prerequisites to this topic.

2. Create or edit an itinerary in the Itinerary Designer and, on an itinerary service object, add or edit a resolver
Please see the BizTalk ESB Toolkit documentation for details on how to configure an itinerary service with a resolver. An example is shown in the following image:

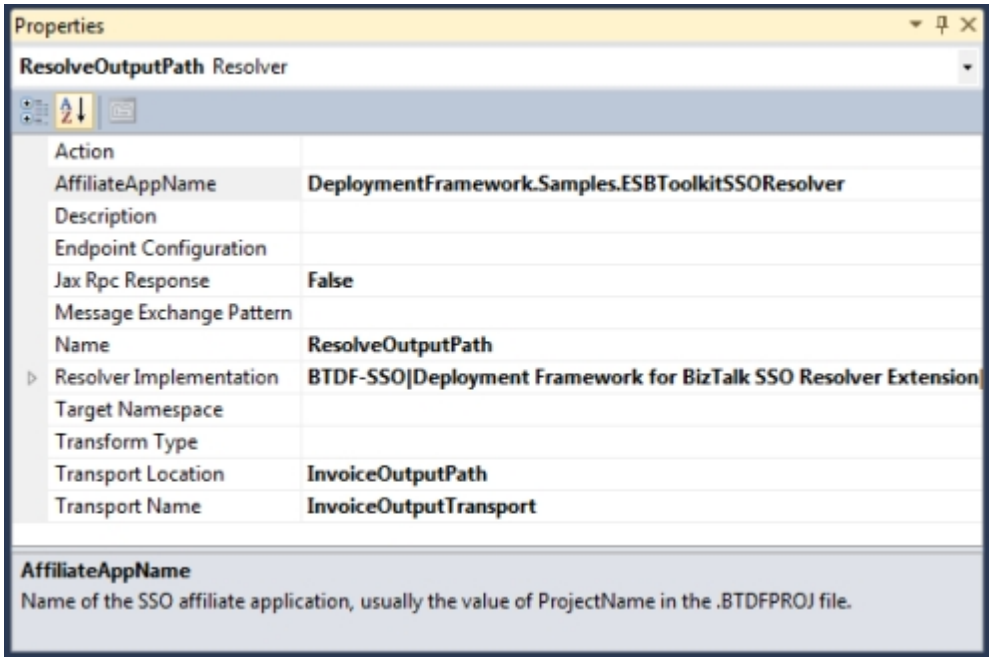


3. Choose "Deployment Framework for BizTalk SSO Resolver Extension" as the Resolver Implementation
This will add the AffiliateAppName property to the properties list.

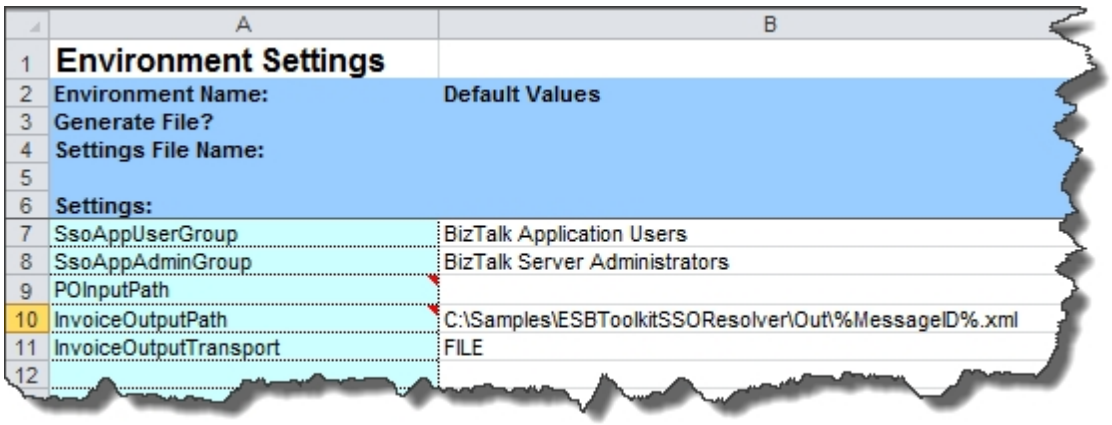
4. Fill in the AffiliateAppName, Transport Location and Transport Type fields
The AffiliateAppName value is the name of the SSO affiliate application into which your project's settings are loaded. This is typically the value of the ProjectName element in your deployment project file.

The values of Transport Location and Transport Name each correspond to the names of settings rows in your settings spreadsheet. The associated values in the spreadsheet should define the destination for the resolver (e.g. C:\MyApp\Output and FILE).

The following image shows a typical resolver configuration:



And the associated configuration in the settings spreadsheet:



	A	B
1	**Environment Settings**	
2	**Environment Name:**	**Default Values**
3	**Generate File?**	
4	**Settings File Name:**	
5		
6	**Settings:**	
7	SsoAppUserGroup	BizTalk Application Users
8	SsoAppAdminGroup	BizTalk Server Administrators
9	POInputPath	
10	InvoiceOutputPath	C:\Samples\ESBToolkitSSOResolver\Out\%MessageID%.xml
11	InvoiceOutputTransport	FILE
12		

Working with Bindings Files

Managing bindings files is one of the more tedious and tricky aspects of BizTalk deployments. There tends to be a lot of XML, some of which is double-encoded in nested XML fragments, and many environment-specific settings are usually present.

The Deployment Framework for BizTalk deploys application bindings from an XML bindings file in your deployment project. By default, you can simply export your application's bindings to a file named PortBindingsMaster.xml in your deployment project folder. That's very easy, but the (huge) downside is that the bindings file is completely specific to the environment from which it was exported. This is the default configuration for new Deployment Framework for BizTalk projects.

To address this problem, the Deployment Framework defines the concept of a "master" bindings file, which is an environment-neutral XML bindings file *template* stored as PortBindingsMaster.xml within your deployment project folder. When using a master bindings file, PortBindings.xml is a tool-generated file. During the deployment process, setting values from your Excel settings spreadsheet are seamlessly merged with the master bindings file to produce an environment-specific PortBindings.xml. To create the master bindings file, you replace environment-specific values with tokens that correspond to setting names in the spreadsheet.

NOTE: The following instructions assume that you are setting up a new Deployment Framework for BizTalk project using the Add New Project wizard. If you take a different approach, then these instructions may not work.

If this is your first time using the Deployment Framework for BizTalk or you just want to get a bindings file going as fast as possible:

1. Create and configure a new Deployment Framework for BizTalk project, then deploy your application from Visual Studio using the Deployment Framework. The script will fail at the point of starting the application. That's OK because you haven't configured the bindings yet.
2. Go into BizTalk Server Administration and manually configure all bindings. Start the application and make sure everything is working.
3. Still in BizTalk Server Administration, export the bindings to <yourBizTalkSolution>\Deployment \PortBindingsMaster.xml. Overwrite the existing file.
4. Once again, deploy your solution from Visual Studio. It should now deploy without errors.

If you want to make your PortBindingsMaster.xml into an environment-neutral template:

5. Follow Steps 1-4 from the previous section.
6. Open PortBindingsMaster.xml in a text editor (see tip about Visual Studio editor below) and EnvironmentSettings\SettingsFileGenerator.xml in MS Excel.
7. In PortBindingsMaster.xml, locate an environment-specific configuration value, for example BindingInfo/ SendPortCollection/SendPort/PrimaryTransport/Address could contain a file path or URL.
8. In Excel, on an empty row enter a descriptive name for the configuration element in column A, like InvoiceOutputPath. In the environment-specific columns or the Default Values column, enter the appropriate value for each environment.
9. In PortBindingsMaster.xml, replace the environment-specific value with a replacement token ${NameFromExcelSpreadsheet}, like ${InvoiceOutputPath}.
10. Repeat these steps until the entire bindings file is environment-neutral and populated with tokens.
11. If you need to do more complex replacements, you can set the property RequireXmlPreprocessDirectives to True in your .btdfproj file and use any of the replacement capabilities of XmlPreprocess.

TIP: When editing a bindings XML file in Visual Studio, beware of auto-formatting. Bindings XML is very touchy, and Visual Studio's auto-formatting can cause invalid XML which will fail upon import into BizTalk.

Adapter, pipeline and party configuration data is often stored within the bindings XML as nested XML strings inside elements. Since that XML is data and not part of the bindings XML itself, it is encoded (< = <, > = >, etc.). The encoding makes it difficult to read and edit that configuration data. The Deployment Framework for BizTalk includes an automatic pre-processor that allows you to keep the nested XML decoded in PortBindingsMaster.xml. At deployment time, the nested XML fragments are re-encoded on the fly.

If you want to store encoded adapter, pipeline and party XML fragments as unencoded XML in PortBindingsMaster.xml for ease of editing:

5. Follow Steps 1-4 from the first section above.
6. Rename PortBindingsMaster.xml to PortBindingsKnownGood.xml

7. Open a Command Prompt. Run ElementTunnel.exe /i:PortBindingsKnownGood.xml /x:adapterXPaths.txt /o:PortBindingsMaster.xml /decode -- adding the appropriate directory paths to the filename parameters. ElementTunnel.exe and AdapterXPaths.txt are in the Deployment Framework install folder under Framework\DeployTools. Your PortBindingsMaster.xml now has selected nested XML fragments decoded into plain, unencoded XML, controlled by the XPath statements in AdapterXPaths.txt.
8. Edit your .btdfproj file and ensure that the property <ApplyXmlEscape> is set to true.
9. In Visual Studio with your solution open, run Tools\Deployment Framework for BizTalk\Preprocess BizTalk Bindings.
10. Use WinMerge or your favorite file compare tool to compare PortBindings.xml (which was automatically generated in the previous step) to PortBindingsKnownGood.xml. Any differences should be analyzed and understood.
11. If there are any differences related to nested XML fragments, you probably need to manually re-encode some nested XML fragments in PortBindingsMaster.xml. Sometimes there is XML nested inside XML nested inside XML. One of those nested XML fragments may not have been processed automatically (by ElementTunnel.exe based on AdapterXPaths.txt). You may need to fix it up manually.
12. Keep adjusting the encoding of nested XML fragments in PortBindingsMaster.xml, re-running Preprocess BizTalk Bindings, and re-comparing PortBindings.xml and PortBindingsKnownGood.xml until all areas including encoded XML are an exact match.
13. Only when you have a successful compare, proceed to Tools\Deploy BizTalk Solution, which should now fully succeed.

Deploying to a BizTalk Server Group

Once you have a Deployment Framework for BizTalk project in hand, it's time to deploy! This section walks you through everything related to deploying your BizTalk application to a single BizTalk server or an entire BizTalk server group. It also describes the process of creating an MSI for your application and how to customize the deployment wizard.

In this section

Understanding Packaging and Deployment

Eventually you will want to take your newly developed BizTalk application and deploy it to a BizTalk server. The Deployment Framework for BizTalk allows you to quickly and easily package your BizTalk application and its dependencies into a standard Windows Installer MSI.

Packaging your application into an MSI may be done from Visual Studio or via command-line execution of MSBuild.

In the Deployment Framework for BizTalk, application deployment on a BizTalk server is a **two-step process**:

1. Install the application on the BizTalk server or servers (interactive Windows Installer wizard or command-line)
2. Deploy the application into BizTalk (MSBuild script execution)

You do not need to install the Deployment Framework for BizTalk on your BizTalk server(s) *unless* you are using the BTDF ESB Resolver. Everything that is needed at runtime is packaged into your application's MSI. See Installation requirements.

Once the MSI is installed on the server, it will appear in Add/Remove Programs just like any other application.

The deployment and undeployment processes work virtually the same as on your development machine. MSBuild executes the steps defined by your project file (.btdfproj), although some steps are modified to be more appropriate for a server environment. One of the more noticeable changes in the process is the appearance of a graphical wizard before deployment and undeployment. This gives your server administrators a friendly interface that collects any information needed to carry out the deployment or undeployment.

The deployment and undeployment processes may be executed interactively (manually) or automated through scripts and command-line tools.

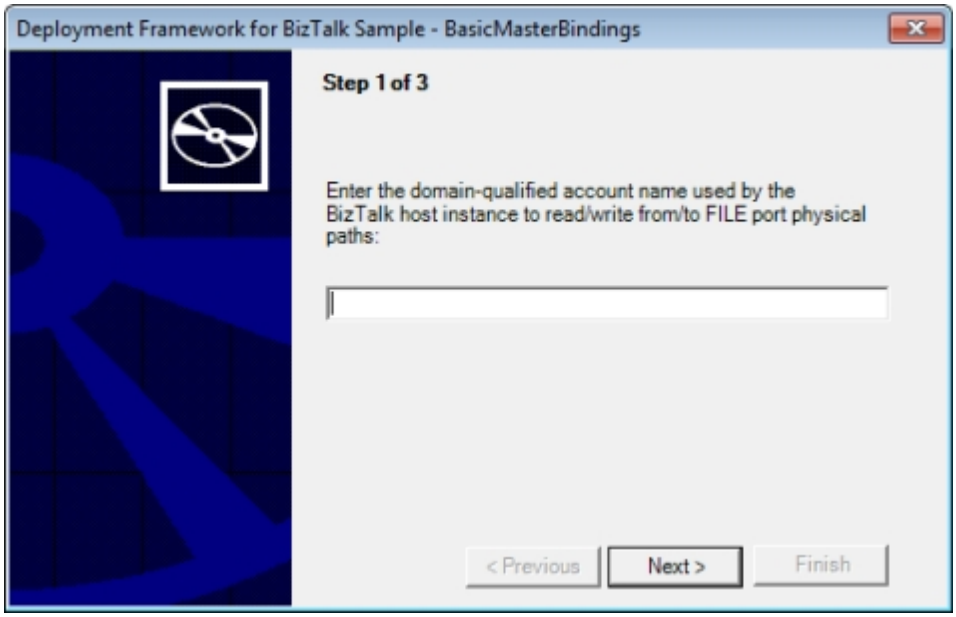
Customizing the Deployment Wizard UI

Deploying your application from Visual Studio is quick and easy, with nothing more than a click of a button required to deploy or undeploy your application. However, when it comes to your BizTalk servers, your BizTalk administrators will want and need more flexibility and hand-holding than you do as a developer. In this topic, we'll look at the graphical wizard that appears during deployments and un-deployments on a BizTalk server.

When you use the Deployment Framework for BizTalk to package your application into an MSI, and then install the MSI on a BizTalk server, you'll notice a couple of differences from your development workstation:

1. A number of menu items for your application appear in the Windows Start menu
2. When you deploy or undeploy your application using the Start menu items, a graphical wizard appears to initialize the process

The typical wizard looks like this:



You have complete control over which pages appear in the wizard, and in what order. If you look in your Deployment Framework for BizTalk project folder, you'll find InstallWizard.xml and UnInstallWizard.xml. These XML files define all of the pages that the wizard will display during, respectively, deployment and undeployment.

The goal of the wizard is to define environment variables before launching MSBuild. Why? All environment variables automatically translate into MSBuild properties that can be referenced like $(EnvVarNameHere). We can use those environment variables/MSBuild properties to control our deployment!

The wizard UI is displayed by a program named SetEnvUI.exe in the Deployment Framework's Framework \DeployTools folder.

Two environment variables **must** be defined when MSBuild is launched on the server to deploy: ENV_SETTINGS and BT_DEPLOY_MGMT_DB. One environment variable **must** be defined to undeploy: BT_DEPLOY_MGMT_DB.

Let's look at a sample InstallWizard.xml:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<SetEnvUIConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  <DialogCaption>Deployment Framework for BizTalk Sample</DialogCaption>
  <SetEnvUIConfigItem>
    <PromptText>Select the XML file that contains configuration information specific to
this environment:</PromptText>
    <PromptValue></PromptValue>
    <ValueType>FileSelect</ValueType>
    <EnvironmentVarName>ENV_SETTINGS</EnvironmentVarName>
  </SetEnvUIConfigItem>
  <SetEnvUIConfigItem>
    <PromptText>Is this the LAST server in the BizTalk Group you are deploying to?</
PromptText>
    <Caption>This is the LAST server in the BizTalk Group</Caption>
    <PromptValue>true</PromptValue>
    <ValueType>Checkbox</ValueType>
    <EnvironmentVarName>BT_DEPLOY_MGMT_DB</EnvironmentVarName>
  </SetEnvUIConfigItem>
</SetEnvUIConfig>
```

Notice the **DialogCaption** element. This element may appear only once, and it defines the text in the title bar of the wizard.

You may include one or more SetEnvUIConfigItem elements, but you will never have fewer than two in InstallWizard.xml and one in UnInstallWizard.xml due to the required environment variables mentioned above. The order of the SetEnvUIConfigItem elements in the XML file is the order in which they will appear as pages in the wizard.

Within a single SetEnvUIConfigItem are the following required elements:

PromptText defines the descriptive text that appears on the top right side of the wizard.
PromptValue defines the default value of the editing control defined by the ValueType element.
ValueType defines the type of editing control that appears on the right side of the wizard, below the text in PromptText. Valid options are Text, Password, FileSelect or Checkbox.
EnvironmentVarName defines the name of the environment variable into which the value from the editing control will be placed.

You may add additional SetEnvUIConfigItem elements in order to capture as many environment variables as you need. Use the environment variables within your .btdfproj file using the $(EnvVarName) syntax to easily customize your server deployments.

Packaging an Application for Deployment

The first step to deploying your application to a BizTalk server is packaging it into an MSI, which can be done either manually in Visual Studio or through a command-line script.

Files Included in the MSI by Default

The Deployment Framework for BizTalk automatically includes all files referenced within your deployment project file. This includes orchestration assemblies, custom .NET assemblies, pipeline component assemblies, rule policy files, ESB itineraries, etc.

To include miscellaneous individual files (vs. an entire folder or a large number of files) in the MSI, you may simply include one or both of the ItemGroup's ExternalAssemblies (.NET DLL's to be installed in the GAC) or AdditionalFiles (miscellaneous files). They will be automatically included in the MSI.

The first packaging step is to copy all necessary files to a staging directory. You'll find this temporary staging directory under <deploymentProjectFolder>\obj\<solutionConfiguration>\redist. For example,

<solutionRoot>\Deployment\obj\Release\redist. If you install your MSI on a server and find that a file is missing, then look to the staging directory on the machine that built the MSI. If a file doesn't get copied there, it will not be part of the MSI.

Including User-Defined Files in the MSI

In your deployment project file, the path to the staging directory is available in an MSBuild property named RedistDir (reference as $(RedistDir)). You may copy any extra files that you wish to include in the MSI inside a target named CustomRedist, placed after all Import elements. *Remember that the Deployment Framework automatically copies all files referenced within the project file*, so you only need to use CustomRedist if you have extra files that are not recognized by the Deployment Framework.

For example, the following target copies everything within the ..\TestFiles folder into the MSI staging folder:

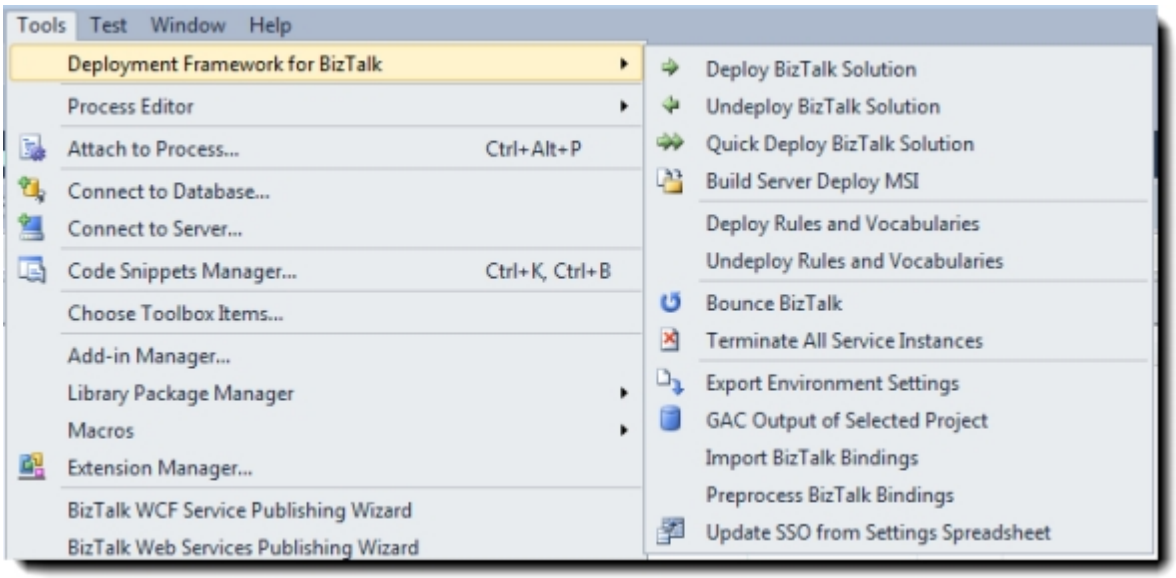
```
<Target Name="CustomRedist">
  <MakeDir Directories="$(RedistDir)\TestFiles" />

  <!-- Force MSBuild to expand the item spec into physical file specs -->
  <CreateItem Include="..\TestFiles\**\*.*">
    <Output TaskParameter="Include" ItemName="TestFilesSourceGroup" />
  </CreateItem>

  <Copy DestinationFolder="$(RedistDir)\TestFiles\%(RecursiveDir)" SourceFiles="@
(TestFilesSourceGroup)"/>
</Target>
```

Build an MSI from Visual Studio

With your BizTalk solution open in Visual Studio, building an MSI is as simple as a menu or toolbar click.



Click the menu item named "Build Server Deploy MSI". You'll also find the icon on the Deployment Framework toolbar. The Deployment Framework for BizTalk will proceed to create the MSI, and you can see the full output from the process in the Visual Studio Output window.

The resulting MSI will be saved to <deploymentProjectFolder>\bin\<solutionConfiguration>. For example, <solutionRoot>\Deployment\bin\Release.

Build an MSI from the Command Line

To build an MSI from the command-line, you need to invoke MSBuild.exe against your deployment project file (.btdfproj) using the following command format:

"<MSBuildPath>\MSBuild.exe" /p:<solutionConfiguration> /t:Installer "<pathToBTDFProj>"

For example:

```
"C:\Windows\Microsoft.NET\Framework\V4.0.30319\MSBuild.exe" /p:Configuration=Release /
t:Installer "C:\TFS\MyBizTalkApp\Deployment\Deployment.btdfproj"
```

The resulting MSI will be saved to <deploymentProjectFolder>\bin\<solutionConfiguration>. For example, <solutionRoot>\Deployment\bin\Release.

Building and Packaging with TFS Team Build

Microsoft Team Foundation Server is often the ALM platform of choice for BizTalk solutions, and you will most likely want to schedule and automate the builds of your BizTalk solutions using TFS Team Build. With the Deployment Framework for BizTalk, it's easy to set up an automated build of your BizTalk solution.

Prerequisites on the Team Build server

The prerequisites vary depending on which version of BizTalk you are using. Here's a list of the software that must be installed **on** the Team Build server:

BizTalk 2006 R2 and earlier:

1. Microsoft Visual Studio (same version required by BizTalk)
2. Microsoft BizTalk Server - Developer Tools and SDK (all Visual Studio integration)

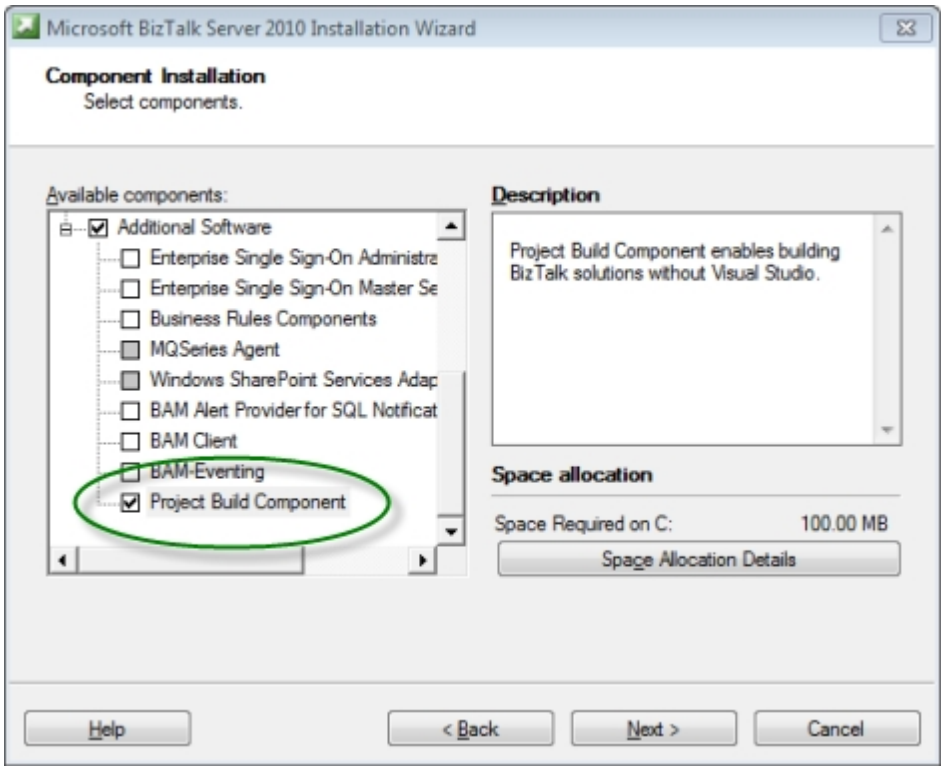
BizTalk 2009:

1. Microsoft .NET Framework 3.5 SP1
2. Microsoft BizTalk Server 2009 - Project Build Component (under Additional Software)

BizTalk 2010:

1. Microsoft .NET Framework 4.0
2. Microsoft BizTalk Server 2010 - Project Build Component (under Additional Software)

The following screenshot shows the Project Build Component in BizTalk 2009 and newer:



Building with BizTalk 2006 R2 and Earlier in Team Build 2005 or 2008

In BizTalk 2006 R2 and earlier, the only way to build a BizTalk solution via command-line (i.e. Team Build) is to run Visual Studio itself using *devenv.exe* and pass it the BizTalk solution file path.

To build a Deployment Framework for BizTalk MSI from Team Build 2005 or 2008:

1. Open or create a TFSBuild.proj Team Build build definition file

Please refer to the Team Build 2005 or 2008 documentation for instructions on creating a build type.

2. Remove and replace the default solution build step(s)

Replace the default solution build process with your own implementation. The following excerpt demonstrates a custom process to build the solution using devenv.exe, build the BTDF MSI and then copy it to the drop folder.

> **Note:** The full TFSBuild.proj may be found in the Samples\TeamBuild folder under the Deployment Framework for BizTalk installation folder.

```
   <Target Name="AfterCompile">
      <!--Call DevEnv to build the BizTalk Solution-->
      <Message Text="Building BizTalk solution using DevEnv"/>
      <Exec
        Command="&quot;c:\Program Files\Microsoft Visual Studio 8\Common7\IDE
\devenv&quot; &quot;$(SolutionPath)&quot; /Build &quot;$(BuildFlavor)&quot;"/>

      <CallTarget Targets="BuildMSI" Condition="'$(DeploymentFramework)' == 'true'" />
   </Target>

   <Target Name="BuildMSI">
      <!--Call batch file to build the MSI (if using Deployment Framework) -->
      <Message Text ="Building MSI using $(DeploymentProjectDir)\BuildReleaseMsi.bat"/>
      <Exec Command="BuildReleaseMsi.bat" WorkingDirectory="$(DeploymentProjectDir)"/>

      <!-- Get the MSI to the appropriate drop location. -->
      <MakeDir Directories="$(DropLocation)\$(BuildNumber)\$(BuildFlavor)"
Condition="!Exists('$(BinariesRoot)\$(BuildFlavor)')" />

      <Message Text="Copying msi from $(DeploymentProjectDir)"/>
      <Copy SourceFiles="$(DeploymentProjectDir)\bin\$(BuildFlavor)\BizTalkSample.msi"
            DestinationFiles="$(DropLocation)\$(BuildNumber)\$(BuildFlavor)
\BizTalkSample.$(BuildNumber).msi"/>
   </Target>
```

Building with BizTalk 2006 R2 and Earlier in Team Build 2010

In BizTalk 2006 R2 and earlier, the only way to build a BizTalk solution via command-line (i.e. Team Build) is to run Visual Studio itself using *devenv.exe* and pass it the BizTalk solution file path.

In Team Build 2010, use the UpgradeTemplate build template. The out-of-the-box WF-based build templates expect to be able to run builds using MSBuild.exe, which is not possible with BizTalk 2006 R2 and earlier. You can also modify the out-of-the-box WF-based build templates or create your own, but the simplest option is to stick with the UpgradeTemplate.

Follow the instructions in "Building with BizTalk 2006 R2 and Earlier in Team Build 2005 or 2008" above. Please refer to the Team Build 2010 documentation for information about changes between Team Build 2005/2008 and 2010.

Building with BizTalk 2009 and Newer in Team Build 2005 or 2008

In BizTalk 2009 and newer, BizTalk solutions may be built via command-line (i.e. Team Build) using

MSBuild.exe against the BizTalk solution file. This means, as identified in the prerequisites above, that Visual Studio is not required on the Team Build server. BizTalk Server includes a special Project Build component for this purpose.

Follow the general instructions in "Building with BizTalk 2006 R2 and Earlier in Team Build 2005 or 2008" above, but run MSBuild.exe instead of devenv.exe. Please refer to the Team Build 2010 documentation for information about changes between Team Build 2005/2008 and 2010.

Building with BizTalk 2009 and Newer in Team Build 2010

In BizTalk 2009 and newer, BizTalk solutions may be built via command-line (i.e. Team Build) using MSBuild.exe against the BizTalk solution file. This is the model used by the out-of-the-box WF-based build template DefaultTemplate.xaml. The Deployment Framework for BizTalk includes a (very) slightly customized version of the Team Build 2010 DefaultTemplate.xaml that you can use to quickly configure your BizTalk solution build.

> **Note:** The BTDFDefaultTemplate.xaml template may be found in the Samples\TeamBuild folder under the Deployment Framework for BizTalk installation folder.

To build a Deployment Framework for BizTalk MSI from Team Build 2010:

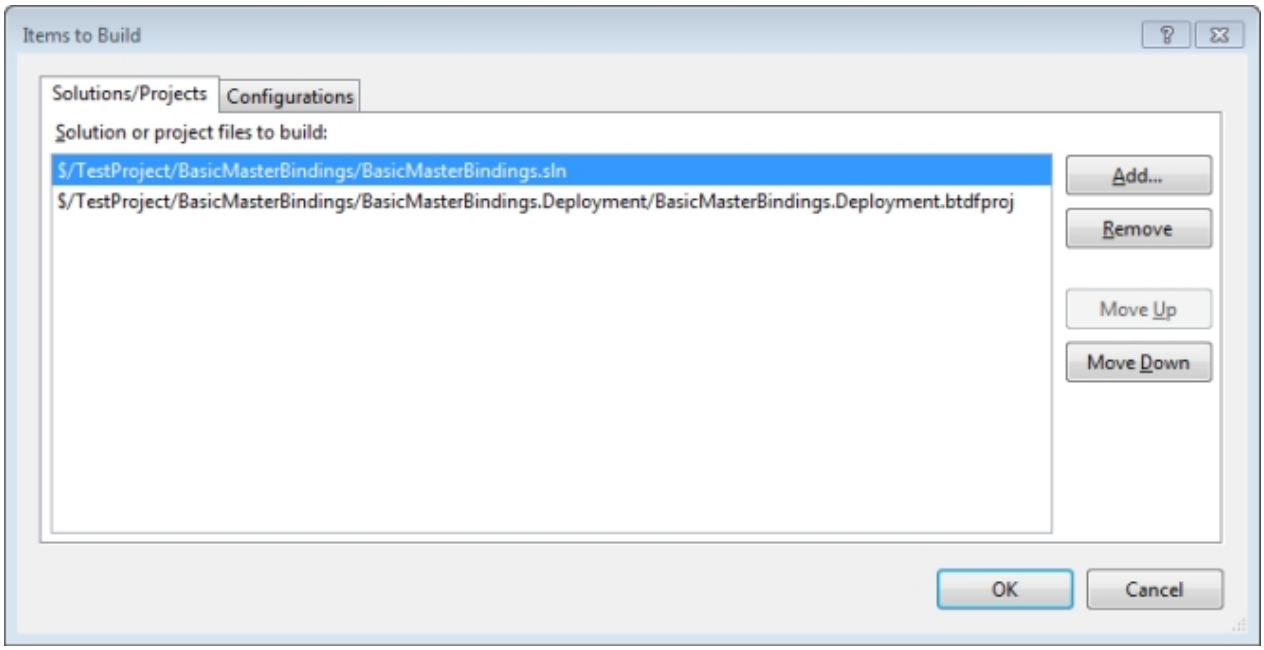
1. Check the BTDFDefaultTemplate.xaml build template into your team project's BuildProcessTemplates folder
Copy the Deployment Framework's BTDFDefaultTemplate.xaml file to your local team project workspace and add it to $/MyTeamProject/BuildProcessTemplates. Remember to add the file **and** check it in.

2. Create a new build definition
To begin, follow the same process that you would use to configure any other .NET solution build. Please refer to the Team Build documentation. Be sure to map your entire BizTalk solution source tree into the build workspace.

3. Configure the build process template (Process node) - Select build template
In the "Build process file" drop-down list, select $/MyTeamProject/BuildProcessTemplates/BTDFDefaultTemplate.xaml. Click "Show details" to expand the build process template pane if necessary. If BTDFDefaultTemplate.xaml is not in the list, click the "New..." button and select "Select an existing XAML file".

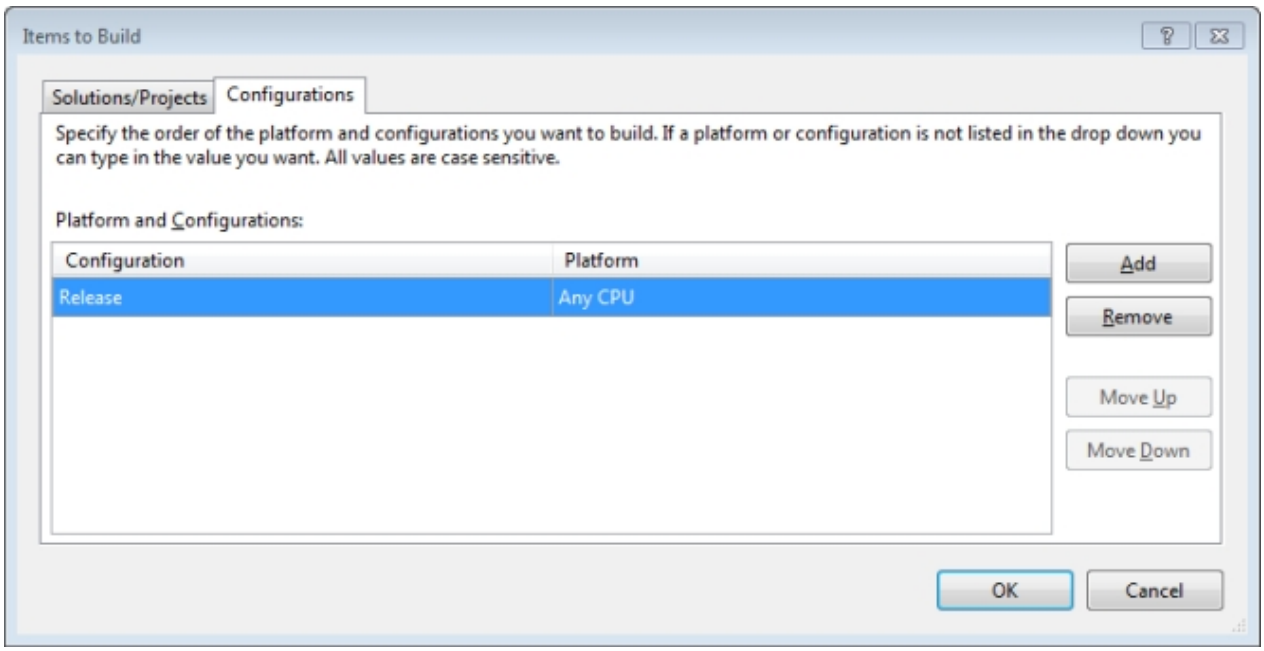
4. Configure the build process template (Process node) - Configure Items to Build
In the Items to Build property, click the "..." button to show the Items to Build dialog box. Add your BizTalk solution (.sln) file to the list on the Solutions/Projects tab. Then, add your BTDF project file (.btdfproj) to the list *after* the solution file.

Your Items to Build dialog should look something like this:



Next, select the Configurations tab. Configure a **single** line to use the Release configuration and Any CPU platform.

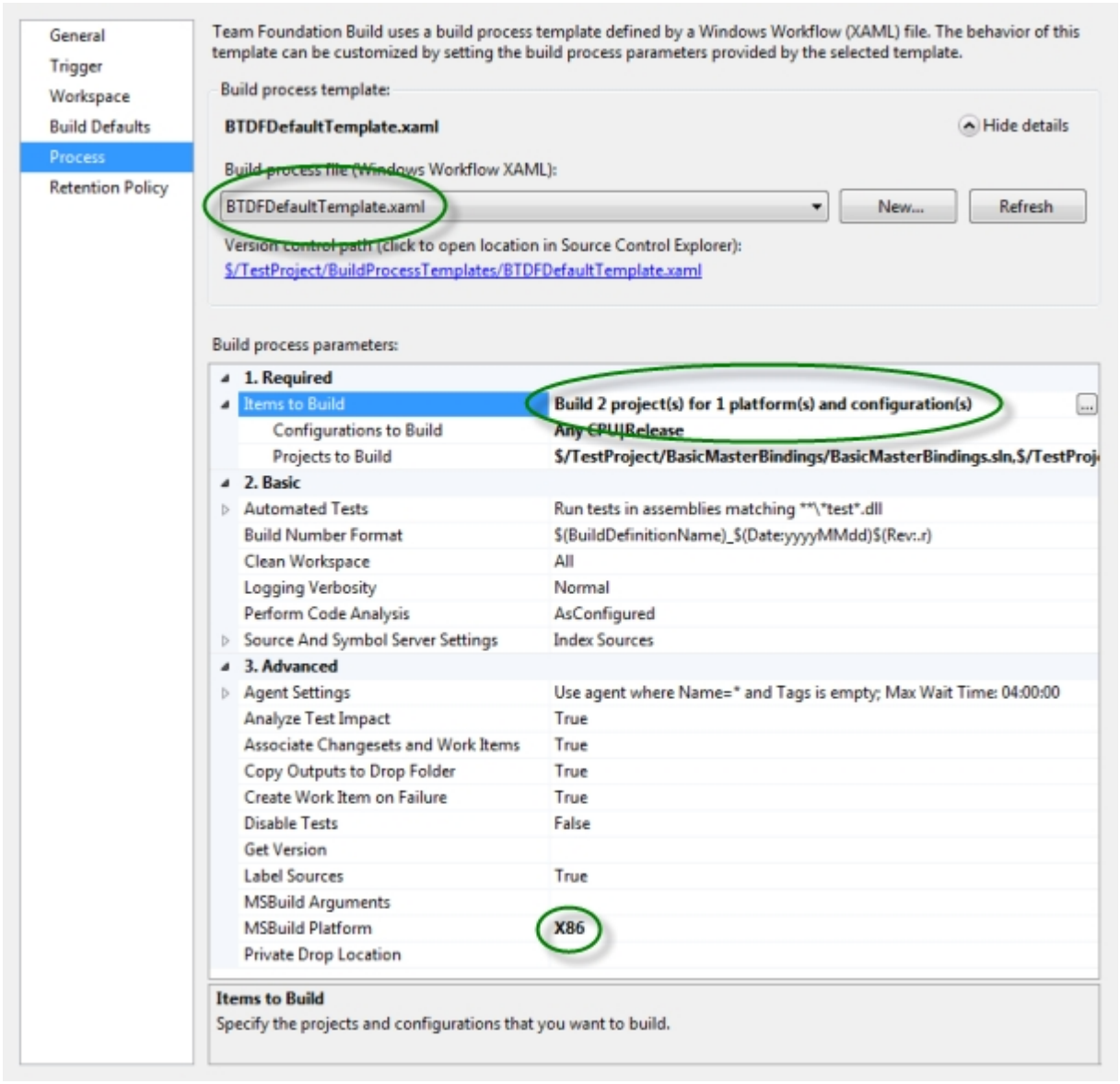
Your Items to Build dialog should now look something like this:



Click OK to return to the main dialog.

5. Configure the build process template (Process node) - Configure MSBuild Platform
Change the MSBuild Platform setting to X86. This is only necessary on x64 operating systems.

6. Verify your Process configuration
Your build definition's Process tab should now look something like this:

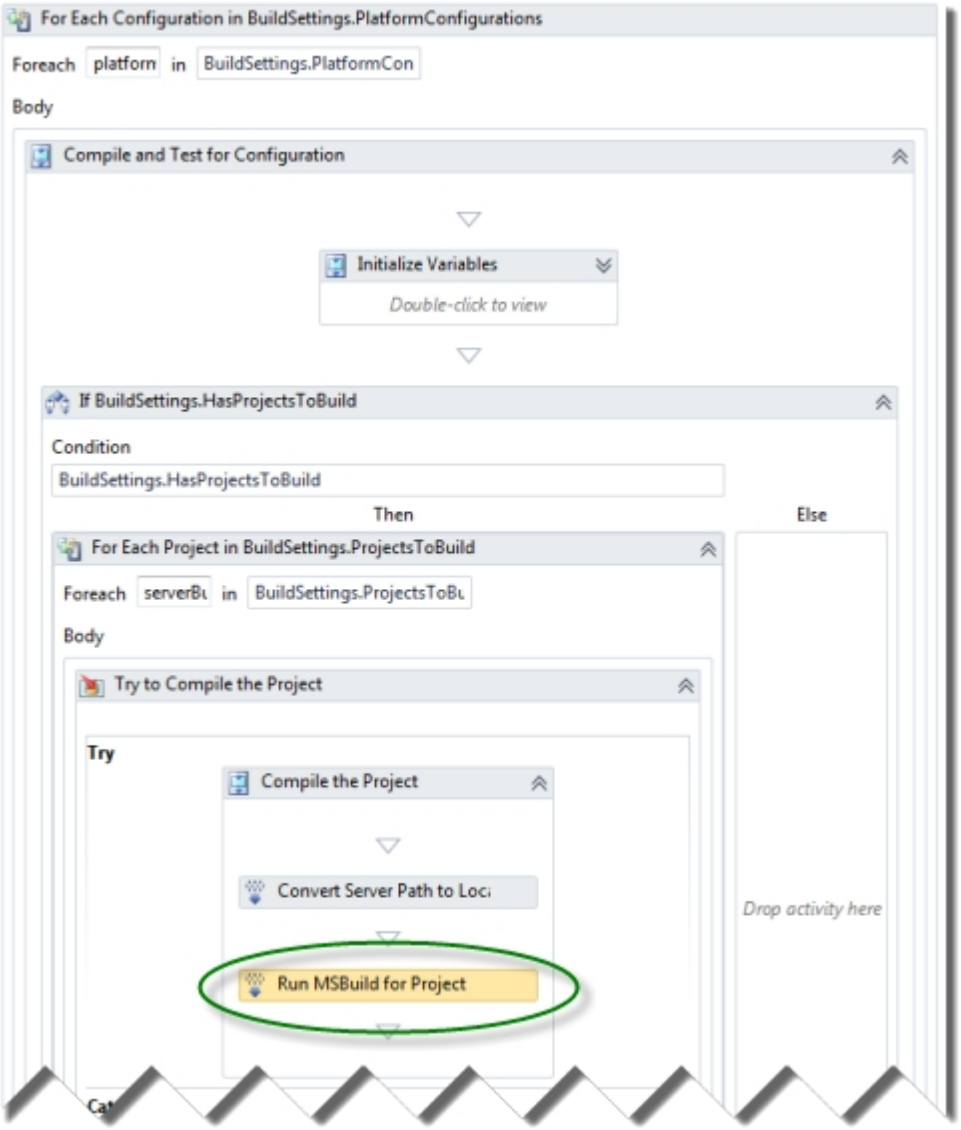


7. Save and test the build definition
Save the build definition, then queue a build. The expected result is a successful build where the drop folder contains the BTDF MSI for the BizTalk solution.

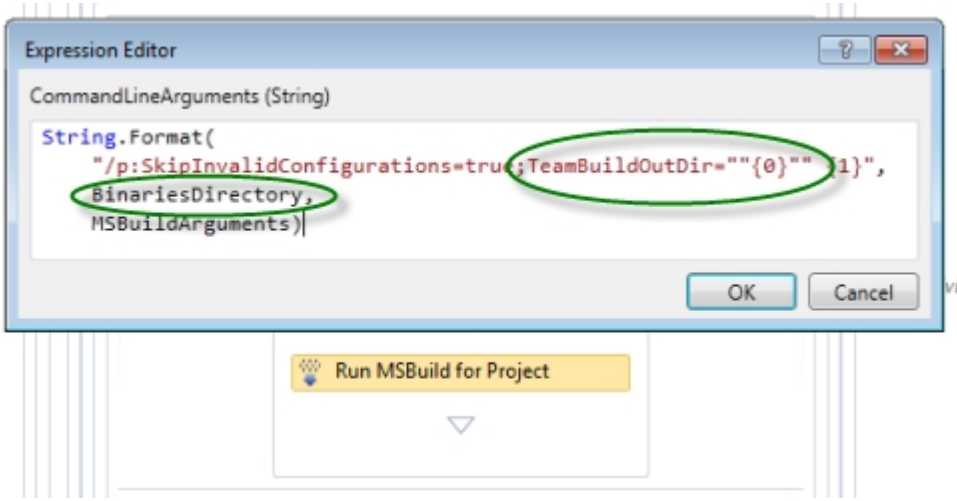
Customizations to the Team Build 2010 DefaultTemplate Build Process Template

The custom version of DefaultTemplate.xaml included with the Deployment Framework for BizTalk as BTDFDefaultTemplate.xaml has only one simple modification: an extra parameter to MSBuild on the project compilation activity. This detail may be useful if you use a custom build workflow and need to add this modification to your own workflow.

The following screenshot indicates the customized activity:



The only modification necessary was to add an MSBuild property named TeamBuildOutDir which receives the value of the BinariesDirectory variable.



Versioning the MSI with Team Build

The deployment project file (.btdfproj) includes a PropertyGroup that contains many MSBuild properties that help define the MSI. They include ProductVersion, ProductId, ProductName, ProductUpgradeCode and

more. The important properties related to Team Build and MSI versioning are *ProductVersion* and *ProductId*.

Deployment projects build in Team Build using MSBuild, just like any other .NET or BizTalk project. That means you can directly reference *any* MSBuild property that is defined by Team Build. You can also modify your build process to pass a property value into MSBuild.exe with the /p switch.

If you generate or maintain a build version number formatted like X.Y.Z, you can pass it to MSBuild.exe with the parameter /p:ProductVersion=X.Y.Z to override the explicit value of ProductVersion in the .btdfproj file with the value provided during the build. You could also replace the explicit value in the .btdfproj file with an MSBuild property reference, using a property name that will be defined during the Team Build process: <ProductVersion>$(TeamBuildProductVersion)</ProductVersion>.

Any blog or other reference that discusses generating and maintaining version numbers in Team Build should directly translate to your Deployment Framework projects.

For Windows Installer to be able to automatically upgrade your application MSI's (see Enabling Automatic Upgrade in the MSI), you must increment the ProductVersion **and** replace the ProductId. The ProductId is just a GUID, so you can follow steps similar to those discussed above to override the explicit ProductId in the deployment project file.

> **NOTE**: Windows Installer only considers Major.Minor.Revision when automatically upgrading an MSI, so for auto-upgrade to work you must increment one of those three components.

Updating Prerelease Deployment Project Files for Team Build Support

Native support for Team Build was added **after** publication of the final release candidate of Deployment Framework for BizTalk 5.0. Deployment project files created in any version of the Deployment Framework prior to V5.0 RTW require a few modifications:

1. Update the DefaultTargets attribute
In the Project element, change the DefaultTargets attribute value to Installer.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003"
DefaultTargets="Installer">
```

2. Update the OutputPath elements in the default Debug and Release configuration PropertyGroup's
In the two PropertyGroup elements that are conditional on the Debug and Release configurations, replace the OutputPath element with two lines that are conditional on the property TeamBuildOutDir. The following example includes the new OutputPath elements. You may already have other elements mixed in with them, so just replace your two existing OutputPath elements with the four shown below.

```
  <PropertyGroup Condition="'$(Configuration)' == 'Debug'">
    <DeploymentFrameworkTargetsPath>$(MSBuildExtensionsPath)
\DeploymentFrameworkForBizTalk\5.0\</DeploymentFrameworkTargetsPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' == ''">bin\Debug\</OutputPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' != ''">$(TeamBuildOutDir)</OutputPath>
    <!-- Get our PDBs into the GAC so we get file/line number information in stack
traces. -->
    <DeployPDBsToGac>false</DeployPDBsToGac>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)' == 'Release'">
    <DeploymentFrameworkTargetsPath>$(MSBuildExtensionsPath)
\DeploymentFrameworkForBizTalk\5.0\</DeploymentFrameworkTargetsPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' == ''">bin\Release\</OutputPath>
    <OutputPath Condition="'$(TeamBuildOutDir)' != ''">$(TeamBuildOutDir)</OutputPath>
    <!-- Get our PDBs into the GAC so we get file/line number information in stack
traces. -->
    <DeployPDBsToGac>false</DeployPDBsToGac>
  </PropertyGroup>
```

For further reference, the Deployment Framework for BizTalk sample applications already include the correct configuration for Team Build.

Deploying an Application Interactively

Deploying a BizTalk application to a BizTalk server or server group using the Deployment Framework for BizTalk is a two-step process:

1. Install the application MSI on the server
2. Launch the deployment process

This topic will describe how to deploy an application interactively (user-assisted).

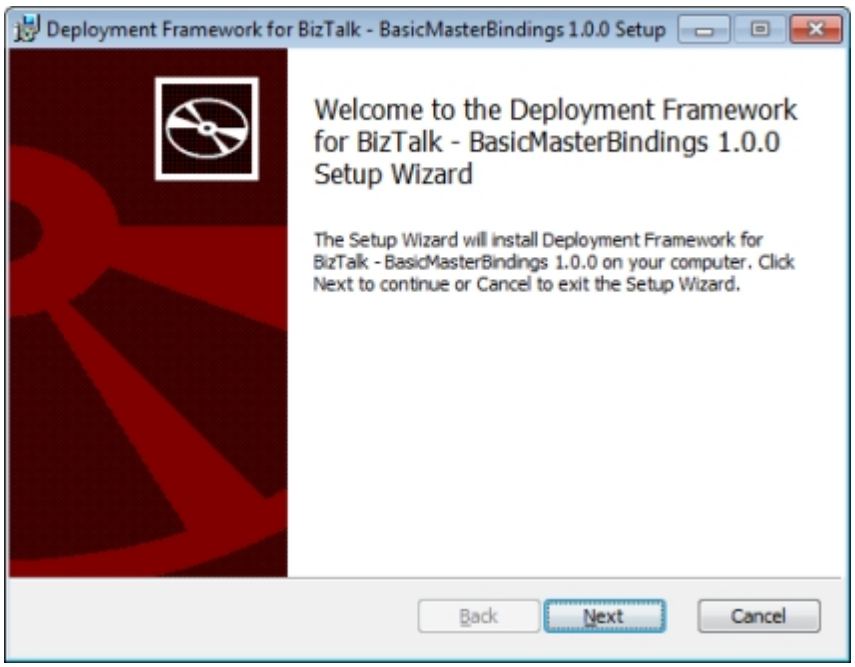
Single Server Deployment

Before you begin, you must have a copy of the BizTalk application's MSI created through the previously described steps. Follow this process to install and deploy a BizTalk application to a single BizTalk server.

Step 1 of 2: Install the MSI

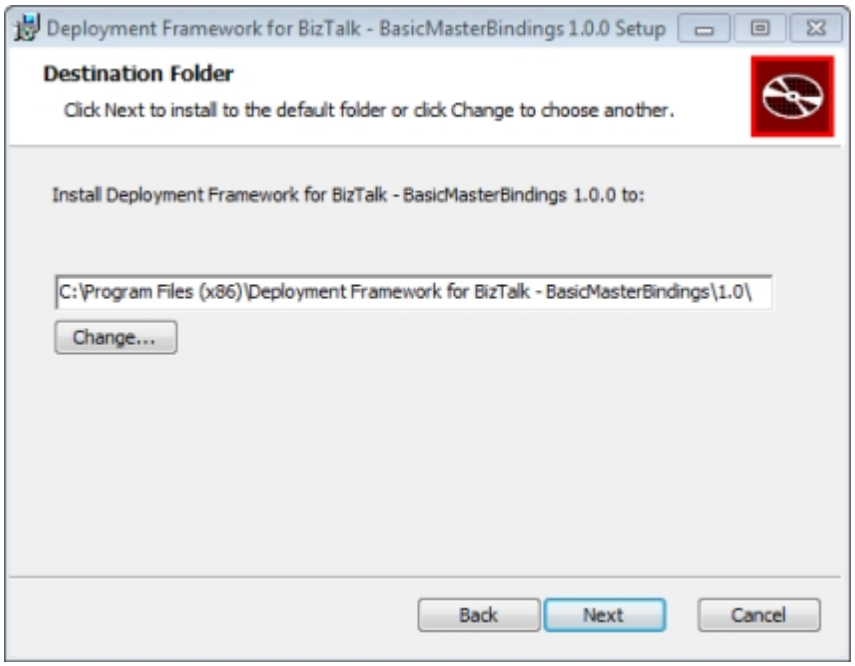
Copy the MSI to the BizTalk server, then double-click the MSI in Windows Explorer to launch the install wizard. The following is a sample of an application installation wizard.

The initial page of the installation wizard displays the name of the application that is about to be installed. Click Next.

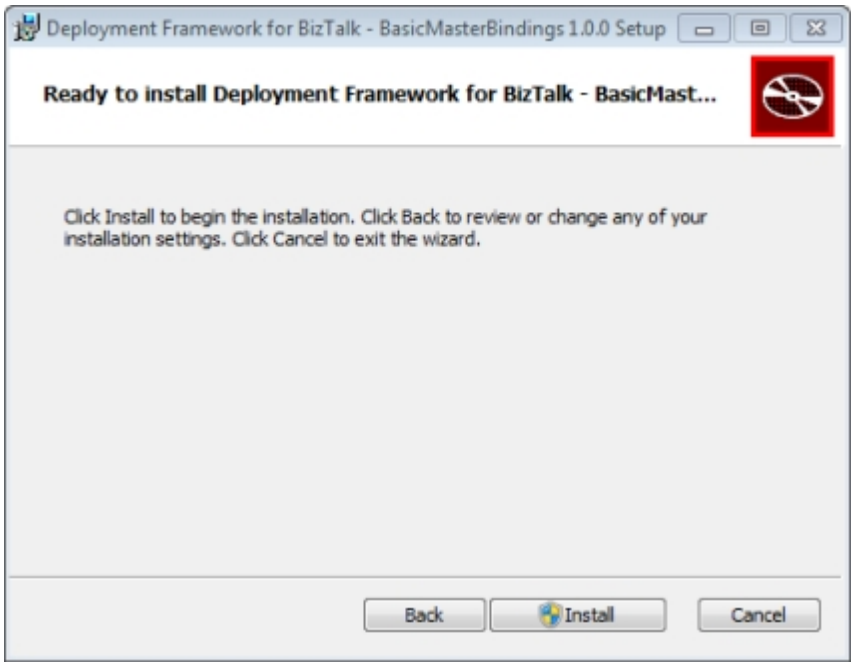


The next page displays the software license agreement. Click Next.

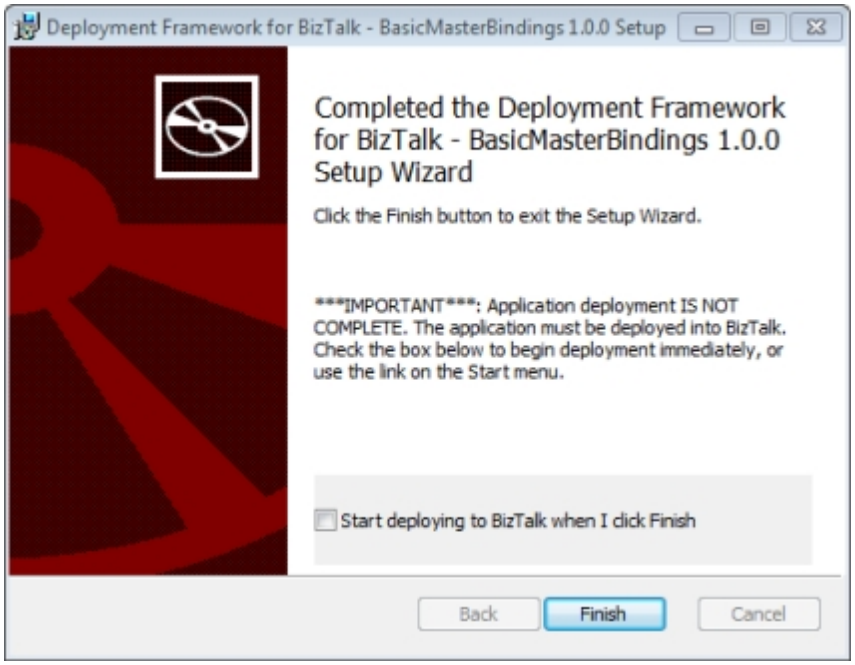
The next page displays the default path where the application files will be installed. Use the Change button to change the path, if desired. Click Next.



The installation is now ready to begin. Click Install to begin the installation.



Once the installation completes, the final page appears.



The application files are now **installed** on the machine, but the application has not been **deployed** into BizTalk. You must make a decision: check the checkbox to deploy the application into BizTalk immediately upon clicking Finish, or skip the checkbox and click Finish without deploying the application into BizTalk. If you choose to finish without deploying the application, then you must initiate the deployment process via the application's program group under the Start menu.

NOTE: The application now appears in Add/Remove Programs like any other installed Windows application.

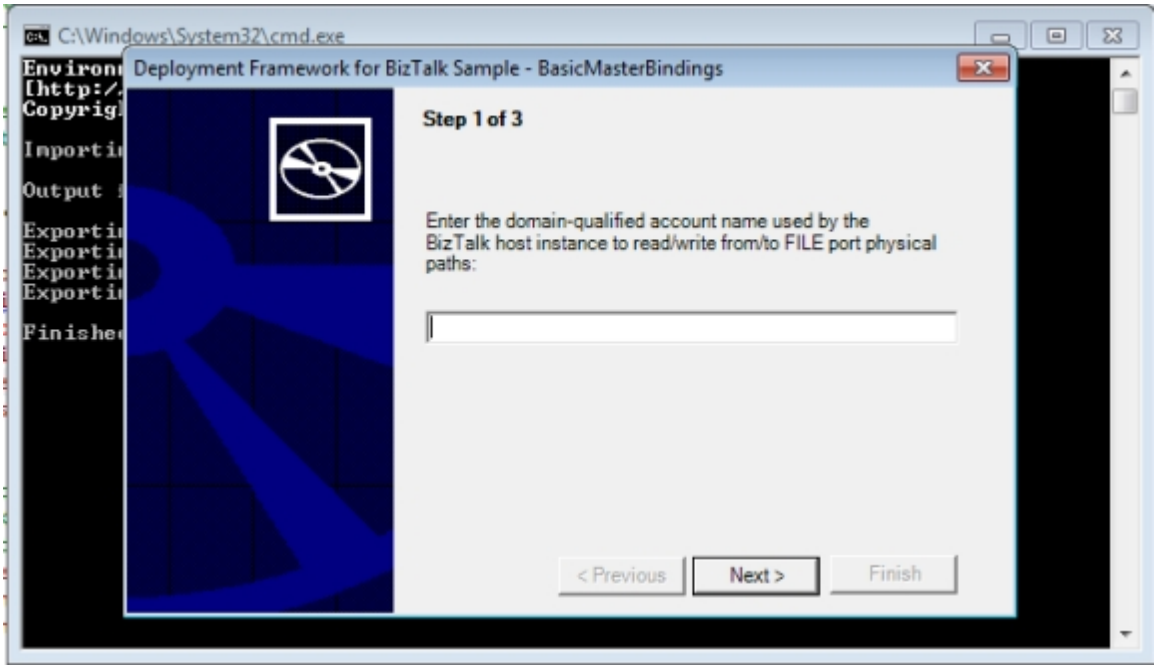
Step 2 of 2: Deploy the Application into BizTalk

Now that the BizTalk application's files have been installed on the BizTalk server, the application is ready to be deployed into BizTalk.

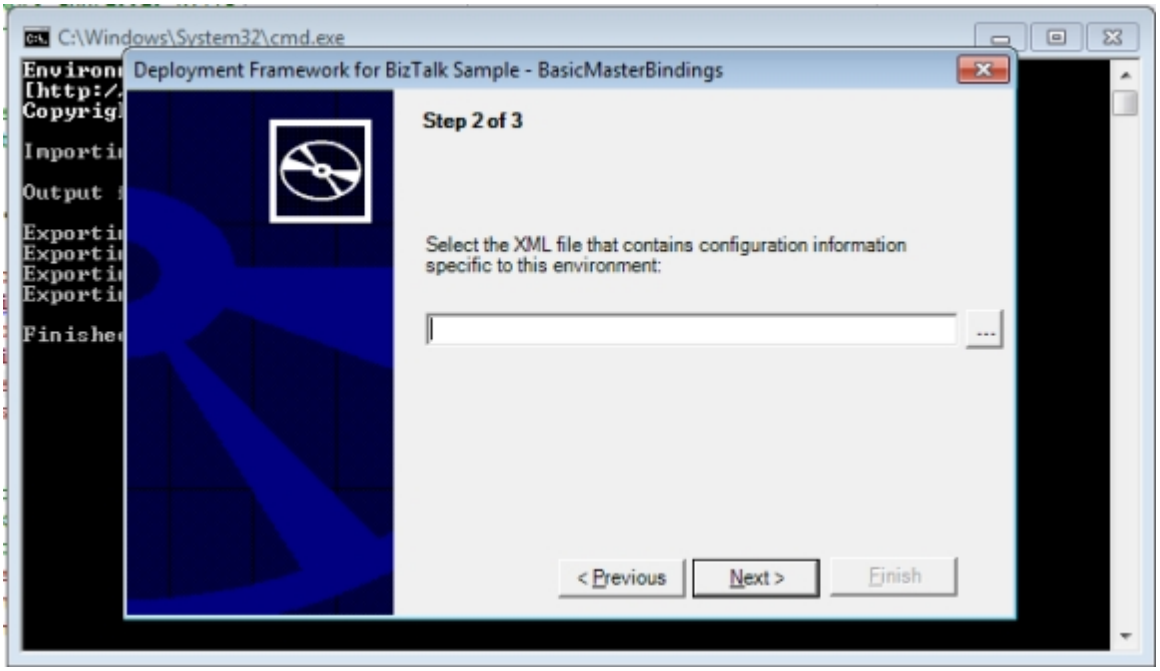
If you checked the checkbox on the final page of the installation wizard, then the deployment wizard should already be visible on the screen. If you did not check the checkbox, then open the Start menu and locate a program group named the same as the BizTalk application. Click the "Deploy <BizTalk app name>" link to launch the deployment wizard.

Regardless of which of the two methods you chose to begin the deployment, you should now see the deployment wizard. An example is shown below. The exact screen will vary depending on how the developer customized the wizard for this particular application.

In this example, the Deployment Framework will grant NTFS permissions to file folders referenced by BizTalk FILE adapters. When asked to provide a username, always use the format DOMAIN\USERNAME. For example, MYDOMAIN\Joe or MYCOMPUTER\Jane.



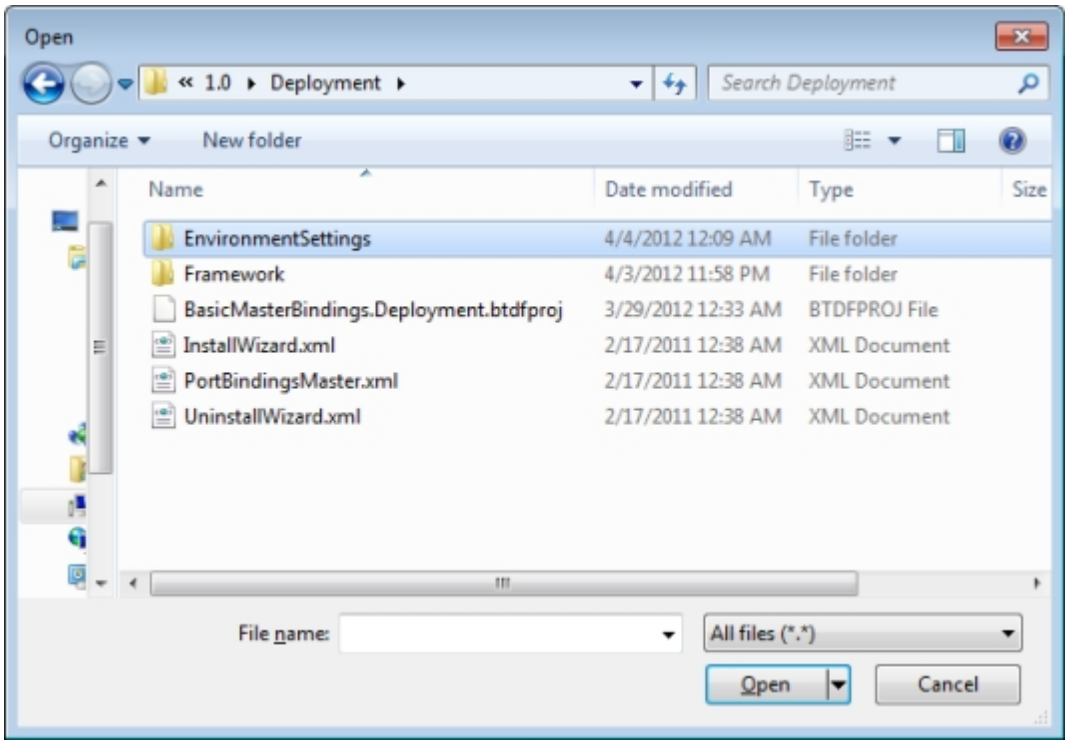
To continue the example, after filling in a username and clicking Next, the following screen appears:



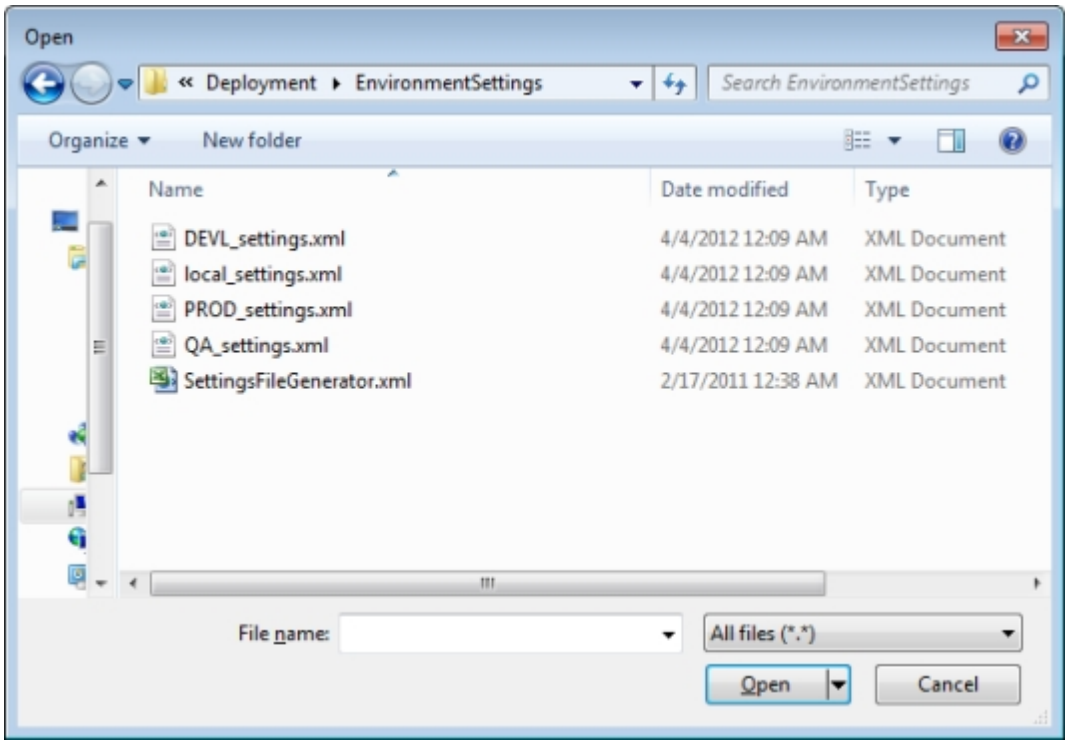
This screen will appear in virtually every deployment wizard, regardless of the application. The Deployment

Framework for BizTalk uses an Excel spreadsheet to maintain an assortment of environment-specific settings (for dev, test, prod, etc.). At deployment time (now), the settings associated with one of those environments are used to dynamically build a BizTalk bindings XML file specific to the current environment.

Click the ellipsis (...) button to bring up a standard Windows file browser dialog. The dialog will open with the current directory set to the application's installation folder (typically under Program Files). Here's an example:

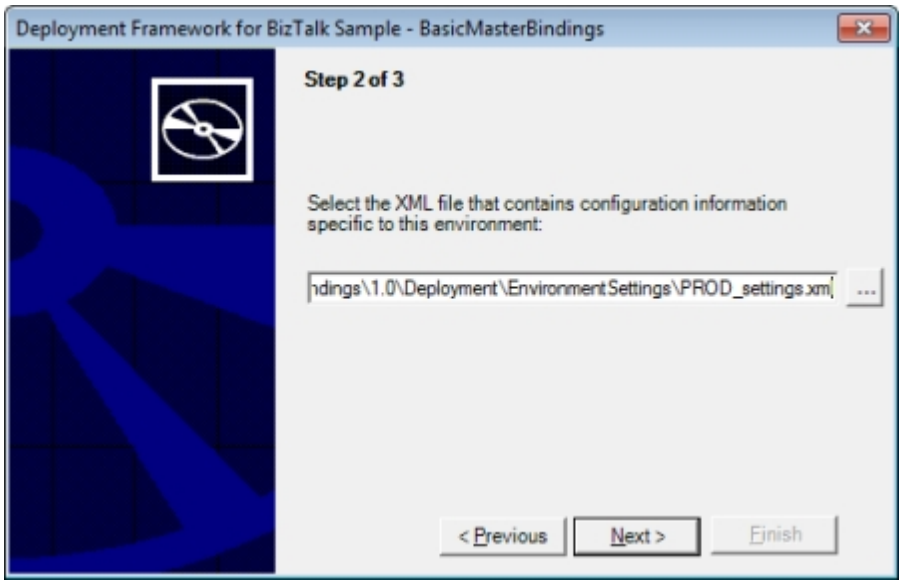


Notice that there is a folder named "EnvironmentSettings". By default, this folder holds an assortment of environment-specific settings XML files that were previously exported from the Excel spreadsheet. Double-click to open the EnvironmentSettings folder. Here's an example of the typical contents:

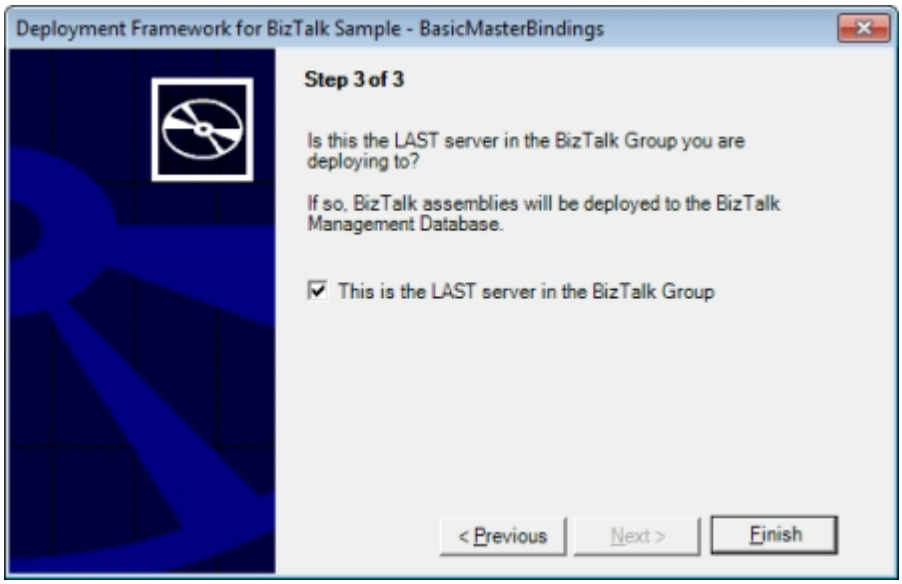


The file names will vary based on your specific environment, but the application developer should have configured them to make sense in your particular environment. Choose the file that corresponds to the current environment in which you are deploying the application and click Open.

The wizard should now look something like this:

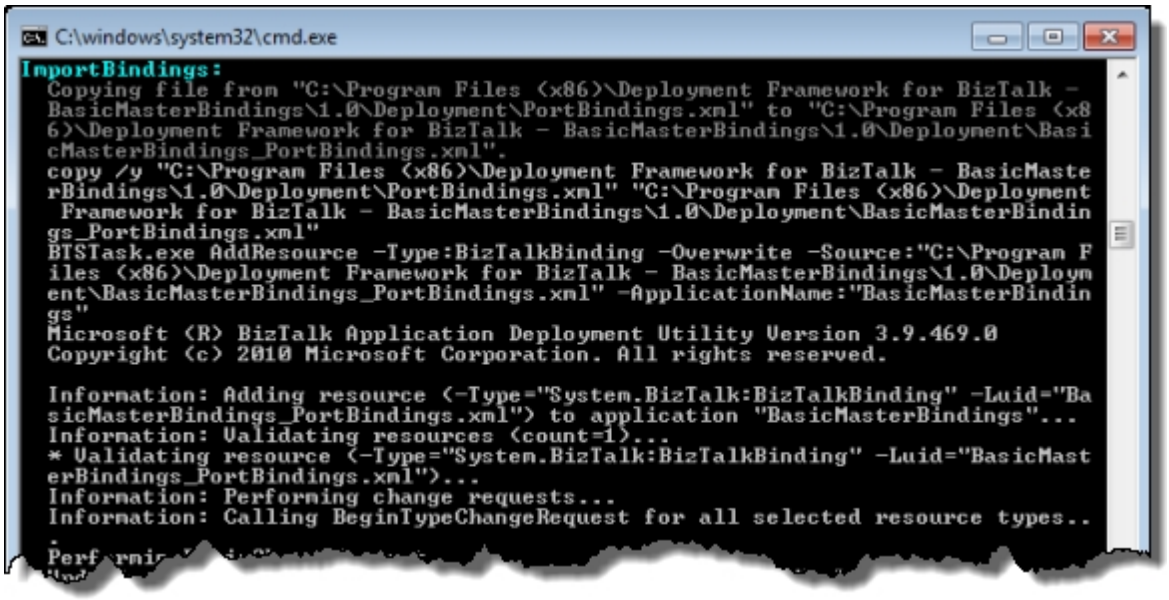


Click Next to continue. Once you have walked through all of the steps configured by the application developer, you will arrive at the final step:



This is a very important step. When deploying to a single BizTalk server, there is only one server in the BizTalk group, so you must check the checkbox before clicking Finish to start the deployment script. When deploying to a BizTalk group, pay careful attention to the sequence in which you deploy to each server and choose the appropriate state for the checkbox.

After clicking Finish, the actual deployment script will begin (driven by MSBuild.exe):



When the deployment finishes, it will display a message indicating success or failure. It's possible that some warnings will appear. They are worth reviewing but generally don't indicate a significant problem.

TIP: The complete output from the deployment script (the text displayed in the command window) is saved to a subfolder named DeployResults in the application install folder (usually under Program Files).

This completes the application deployment to a single BizTalk server.

Multi-Server Deployment (BizTalk Group)

Once you understand how to deploy your application to a single BizTalk server, it's easy to extend that knowledge to an entire BizTalk group.

The multi-server deployment process is virtually identical to the single-server process. The key *differences* are:

1. The application MSI must be copied to and installed on every server in your BizTalk group
2. The deployment script must be executed on every server in your BizTalk group
3. The checkbox on the last page of the deployment wizard, which asks "is this the LAST server in the BizTalk group," must be set correctly on each server

BizTalk requires application binaries and certain other artifacts to be installed on every BizTalk server that runs the application. However, the BizTalk application, its port bindings, rule policies and more must be registered in the BizTalk databases **only once within the group**. This is the reason why the checkbox asks if this is the last server in the group.

The recommended process is:

1. Identify the complete set of BizTalk servers within the BizTalk group that will run the application
2. Walking through the servers one at a time:
3. Copy the application MSI to the server
4. Install the MSI
5. Deploy the application, ***unchecking*** the checkbox on the final page of the deployment wizard
6. On the final server in the list, follow steps 3-5 but ***check*** the checkbox on the final page of the deployment wizard

The deployment process on each individual server is the same as the Single Server Deployment process described above.

Deploying an Application via Script

Deploying a BizTalk application to a BizTalk server or server group using the Deployment Framework for BizTalk is a two-step process:

1. Install the application MSI on the server
2. Launch the deployment process

This topic will describe how to deploy an application via script (non-interactive). Since there are countless scripting tools available (batch files, PowerShell, TFS workflows, etc.) this discussion will convey the general process without going into the details of a particular tool.

Single Server Deployment

Before you begin, you must have a copy of the BizTalk application's MSI created through the previously described steps. Follow this process to install and deploy a BizTalk application to a single BizTalk server.

Step 1 of 2: Install the MSI

Before the BizTalk application can be deployed to BizTalk, it must be installed on the server:

1. Copy the application's MSI to the BizTalk server
2. Install the MSI

In general, you'll install the MSI using the Windows Installer command-line tool msiexec.exe:

```
msiexec.exe /i MyBizTalkApp.msi /passive /log MyBizTalkAppInstall.log INSTALLDIR="C:
\Program Files\MyBizTalkApp\1.0"
```

The /i parameter specifies the MSI name, /passive activates unattended mode to show only a progress bar and /log creates a log file. The final, and optional, parameter INSTALLDIR overrides a Windows Installer property that is predefined within the MSI. INSTALLDIR specifies the file system path at which the application files will be installed.

A good option for remote execution of msiexec.exe across the network on the BizTalk server is Microsoft SysInternals PsExec. WinRS.exe is another possible option.

Step 2 of 2: Deploy the Application into BizTalk

Now that the BizTalk application's files have been installed on the BizTalk server, the application is ready to be deployed into BizTalk.

The deployment script is executed by MSBuild.exe. Many servers will have several versions of MSBuild.exe installed: one from .NET Framework 2.0, one from .NET Framework 3.5 and another from .NET Framework 4.0. Any version should work. The important thing is to run the 32-bit MSBuild.exe, even on Windows x64.

For this example, I'll assume that .NET Framework 4.0 is installed.

The first command, shown below, exports the environment-specific settings XML files from the settings file spreadsheet (usually named SettingsFileGenerator.xml).

```
"C:\Program Files\MyBizTalkApp\1.0\Deployment\Framework\DeployTools
\EnvironmentSettingsExporter.exe" "C:\Program Files\MyBizTalkApp\1.0\Deployment
\EnvironmentSettings\SettingsFileGenerator.xml" "C:\Program Files\MyBizTalkApp\1.0
\Deployment\EnvironmentSettings"
```

The second command, shown below, executes the deployment process.

```
"%windir%\Microsoft.NET\Framework\v4.0.30319\MSBuild.exe" /
p:DeployBizTalkMgmtDB=true;Configuration=Server;SkipUndeploy=true /target:Deploy /
```

```
l:FileLogger,Microsoft.Build.Engine;logfile="C:\Program Files\MyBizTalkApp\1.0
\DeployResults\DeployResults.txt" "C:\Program Files\MyBizTalkApp\1.0\Deployment
\MyBizTalkApp.btdfproj" /p:ENV_SETTINGS="C:\Program Files\MyBizTalkApp\1.0\Deployment
\EnvironmentSettings\Exported_ProdSettings.xml"
```

The /p parameter includes multiple Property=Value pairs separated by semicolons. This allows MSBuild properties to be defined directly from the command-line, where they will take precedence over default values set within the deployment project.

Pay particular attention to the ENV_SETTINGS property, which points to one of the settings XML files exported from the settings spreadsheet. In an interactive deployment, this path is collected from the user in the deployment wizard. With a scripted deployment, you must provide the path yourself.

Multi-Server Deployment (BizTalk Group)

Once you understand how to deploy your application to a single BizTalk server via script, it's easy to extend that knowledge to an entire BizTalk group.

The multi-server deployment process is virtually identical to the single-server process. The key *differences* are:

1. The application MSI must be copied to and installed on every server in your BizTalk group
2. The deployment script must be executed on every server in your BizTalk group
3. The DeployBizTalkMgmtDB property must be correctly set on each server

BizTalk requires application binaries and certain other artifacts to be installed on every BizTalk server that runs the application. However, the BizTalk application, its port bindings, rule policies and more must be registered in the BizTalk databases **only once within the group**. The DeployBizTalkMgmtDB property controls whether to deploy application files on the server AND deploy configuration data into the BizTalk databases, or to simply deploy application files on the server.

The recommended process is:

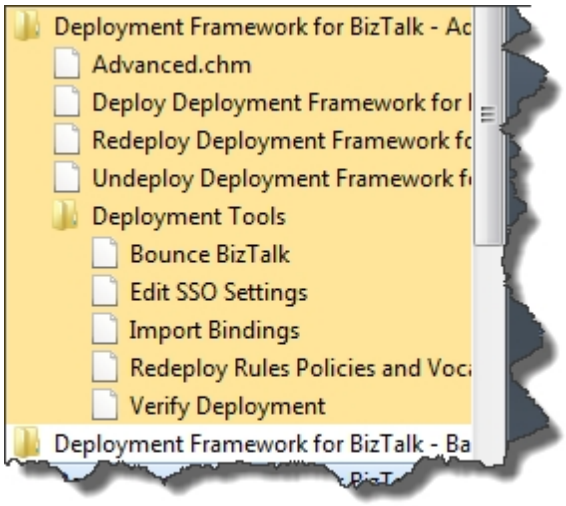
1. Identify the complete set of BizTalk servers within the BizTalk group that will run the application
2. Walking through the servers one at a time:
3. Copy the application MSI to the server
4. Install the MSI
5. Deploy the application with DeployBizTalkMgmtDB=**false**
6. On the final server in the list, follow steps 3-5 but use DeployBizTalkMgmtDB=**true**

The deployment process on each individual server is the same as the Single Server Deployment process described above.

Working with a Deployed Application

Once you've deployed your application to one or more BizTalk servers, you'll manage it mostly through the BizTalk Server Administration tool. However, you'll want to be aware of a number of actions that are provided by the Deployment Framework for BizTalk. They're all found in the Start menu under your application's name.

The specific list of options depends on whether your application uses the BRE and/or SSO, but here is a representative sample (captured from a Windows 7 Start menu):



The **Deploy <Application Name>** command initiates the deployment of your BizTalk application. Unlike deployments from Visual Studio, this action will present a wizard UI to collect certain data needed for the deployment. (Keep in mind that installing your MSI is a separate step from deploying your application into BizTalk.)

The **Undeploy <Application Name>** command initiates the undeployment of your BizTalk application. You must do this before you uninstall your application's MSI.

The **Redeploy <Application Name>** command undeploys then deploys your BizTalk application in one step.

The **Bounce BizTalk** command will restart all BizTalk hosts or those defined in your deployment project's BizTalkHosts ItemGroup.

The **Edit SSO Settings** command launches the SSO Settings Editor tool that allows you to edit any application settings stored in SSO. This command appears when the deployment project has the IncludeSSO property set to True.

The **Import Bindings** command will re-import your application's PortBindings.xml file.

The **Redeploy Rules Policies and Vocabularies** command redeploys all BRE vocabularies and rules associated with your application. This command appears when the deployment project has the IncludeVocabAndRules property set to True.

The **Verify Deployment** command launches the NUnit GUI and auto-loads your application's NUnit test assembly. This command appears when the deployment project has the IncludeDeploymentTest property set to True.

The **"Advanced.chm"** command is a link to a CHM Help file which is optionally included in your MSI. (The name will vary based on your application.) Typically this CHM file would be generated by the BizTalk Documenter tool. The CHM file must be located in the **parent** folder of your deployment project, which is often the same folder as the solution file.

Testing a Deployed Application

More often than not, BizTalk administrators and/or operations staff find themselves in the position of having just deployed a BizTalk application to production -- yet having no idea if the application actually works. An extremely useful and elegant solution for this problem is to bundle a handful of verification tests with your application. The Deployment Framework for BizTalk uses NUnit as the foundation for your verification tests.

To include verification tests with your application:

1. Add a .NET Class Library project to your BizTalk solution to house your NUnit tests
If you are using BizTalk 2010 or newer then it's best to target .NET 4.0. Otherwise, target .NET 3.5 or lower. Please reference the NUnit documentation for details on how to implement one or more unit test methods within the project.

2. Set the IncludeDeploymentTest property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <IncludeDeploymentTest>true</IncludeDeploymentTest>
  ...
</PropertyGroup>
```

3. Add a DeploymentTest ItemGroup
The following is a typical example that properly follows the common ItemGroup structure:

```
<ItemGroup>
  <DeploymentTest Include="$(ProjectName).DeploymentTest.dll">
    <LocationPath>..\DeploymentTest\bin\$(Configuration)</LocationPath>
  </DeploymentTest>
</ItemGroup>
```

The following is the default configuration, which corresponds to the optional naming conventions. This default is in effect if you do not include any DeploymentTest elements in your project file.

```
<ItemGroup>
  <DeploymentTest Include="$(ProjectName).DeploymentTest.dll">
    <LocationPath>..\$(ProjectName).DeploymentTest\bin\$(Configuration)</LocationPath>
  </DeploymentTest>
</ItemGroup>
```

Your NUnit test assembly will automatically be included when you generate an MSI for deployment to a BizTalk server or group. After the application is installed on the server, the Start menu includes a shortcut to launch the NUnit GUI with the test assembly loaded and ready to run.

Upgrading a Deployed Application

Recall that the Deployment Framework for BizTalk uses a two-step process to deploy a BizTalk application to a server: 1) install the MSI, then 2) deploy the application.

The application upgrade process is (in order):
1. Undeploy the existing version of the application (scripted or through the Start menu shortcut)
2. Install the new version of the application (MSI installer)
3. Deploy the new version of the application (scripted, through the Start menu shortcut or by checking the box on the last page of the install wizard)

If other BizTalk applications are dependent on the application via app-to-app references, then the other applications must be undeployed first.

The Deployment Framework allows a BizTalk application's files (upgrade step #2) to be quickly upgraded by simply installing the new version over the old version. The MSI installer will detect the old version and silently uninstall it, then install the new version.

NOTE: You **MUST UNDEPLOY** the old version of the application **before** installing the new MSI. The MSI will upgrade the application files, but the old application must be undeployed first, and the new application must be deployed when the installer is finished. If you forget this step, you must reinstall the old version's MSI and complete the undeploy process.

Enabling Automatic Upgrade in the MSI

In order for Windows Installer to perform an automatic upgrade of your application, you must do two things before (or while) building the new version of your application's MSI:

1. Replace the ProductId GUID in the .btdfproj with a new GUID
2. Increment the ProductVersion in the .btdfproj.

Windows Installer only looks at the first three digits of the version number, so you must increment one of the first three digits: 1.1.0.0 to 1.1.1.0 will work, but 1.1.0.1 to 1.1.0.2 will **not** work. Do **not** change the ProductUpgradeCode, as it must be consistent across all versions of the MSI.

When using an automated build system, it's common to override the explicit property values in the .btdfproj by passing new values in from the build system via the MSBuild command line (like /p:ProductVersion=1.0.5).

Walkthroughs

This section contains step-by-step walkthroughs to help you get up to speed quickly with the Deployment Framework for BizTalk. They are a supplement to the sample applications included with the Deployment Framework installation.

In this section

- Create a Deployment Project for the HelloWorld Application

Create a Deployment Project for the HelloWorld Application

Overview

The goal of this walkthrough is to demonstrate how to take a simple, functional BizTalk Server application and build a Deployment Framework for BizTalk project for it. Rather than create a new application from scratch, the BizTalk Server SDK HelloWorld sample application will serve as the application.

Topics

During this walkthrough, you will:

- Learn to use the Deployment Framework for BizTalk's Add New Project wizard
- Customize a Deployment Framework project file in the Visual Studio editor using IntelliSense
- Configure an XML bindings file for use with the Deployment Framework project
- Deploy the application as a developer using the Deployment Framework's Visual Studio integration
- Build a Deployment Framework MSI for deployment to a BizTalk server
- Deploy the MSI to the local machine as if it were a BizTalk server

Step 1: Develop the Application

Rather than develop an application from scratch, we'll start with a simple application from the BizTalk Server SDK: the Orchestrations HelloWorld sample.

TIP: You'll normally want to create a deployment project for your application very early in your development process. It will not only speed your development cycle, but also offer additional flexibility in your application development through features like the SSO Resolver and SSO Settings API.

Step 1: Copy the HelloWorld sample application to a working folder

1a. Create a new folder to hold the walkthrough application files at C:\BTDF\Walkthrough1.

1b. Copy the folder <BizTalkInstallationPath>\SDK\Samples\Orchestrations\HelloWorld into C:\BTDF\Walkthrough1. You should now have the entire sample application in C:\BTDF\Walkthrough1\HelloWorld.

Step 2: Delete unnecessary files

Delete the files Setup.bat and Cleanup.bat from the HelloWorld folder.

Step 3: Move the solution file

Move HelloWorld.sln from the HelloWorld folder to the Walkthrough1 folder. We want the solution file to be

at the top of our folder structure since we will be adding a deployment project folder next to the HelloWorld application folder.

Step 4: Create a strong-name key (SNK) file

4a. Open a Visual Studio Command Prompt from the Start/Microsoft Visual Studio <version> folder that corresponds to your BizTalk version. Run the following two commands:

```
cd "C:\BTDF\Walkthrough1\HelloWorld"
sn -k HelloWorld.snk
```

4b. Close the Command Prompt window.

Step 5: Create file input and output folders and grant permissions

5a. Create a new folder to hold input XML files at C:\BTDF\Walkthrough1\In. Create another new folder to hold output XML files at C:\BTDF\Walkthrough1\Out.

5b. In Windows Explorer, open the Properties page for C:\BTDF\Walkthrough1\In and switch to the Security tab. Click the Edit... button, then the Add... button. Enter the name of the user account used by your BizTalk host instances, then click OK. Check the Allow > Full Control box and click OK twice to return to Explorer. The BizTalk host instance user account should now have Full Control access to the C:\BTDF \Walkthrough1\In folder.

5c. Repeat the previous steps to grant permissions on the C:\BTDF\Walkthrough1\Out folder.

Step 6: Open and build the HelloWorld sample application

6a. Run the version of Visual Studio that corresponds to your BizTalk version and, on Windows Vista or above, ensure that you run Visual Studio "as Administrator".

6b. Open the HelloWorld solution file (C:\BTDF\Walkthrough1\HelloWorld.sln). You will see an error that one or more projects did not load correctly -- just click OK.

6c. In the Solution Explorer window, select the HelloWorld project (marked as unavailable). With the project selected, press Alt-Enter or use the View menu to display the Properties window. Update the "File path" property to reflect the new path to the project file: C:\BTDF\Walkthrough1\HelloWorld\HelloWorld.btproj.

6d. Back in the Solution Explorer, right-click the HelloWorld project and choose the Reload Project command.

6e. The project should now be loaded. Rebuild the solution using the Build\Rebuild Solution command and ensure that the build succeeded.

Step 7: Disable the built-in deployment option

In recent versions of BizTalk, every BizTalk project has a "Deploy" option in addition to the standard "Build" option. When using the Deployment Framework for BizTalk, you always want to disable the built-in Deploy option.

Right-click the HelloWorld *solution* and choose the Configuration Manager command. In the Deploy column, un-check the Deploy checkbox. Change the "Active solution configuration" to Release and again un-check the Deploy checkbox. Switch the Active configuration back to Debug and click the Close button to close the dialog.

Step 2: Create a BTDF Project

You should now have the HelloWorld solution from C:\BTDF\Walkthrough1 open in Visual Studio and successfully built. The next step is to create a new Deployment Framework for BizTalk project.

A best practice for organization of BizTalk solutions is to divide the artifacts into separate assemblies. The HelloWorld sample instead has a single BizTalk project that contains schemas, a map and an orchestration, so we will make a couple of extra settings in the deployment project to account for that configuration.

Step 1: Create a deployment project with the Add New Project wizard

The first step with every new BizTalk application is to use the Add New Project wizard to create a Deployment Framework for BizTalk project.

1a. With the HelloWorld solution and HelloWorld BizTalk project visible in Solution Explorer, right-click the HelloWorld *solution* and choose the Add > New Project... command. The Add New Project dialog box will appear.

1b. Select the category named BizTalk Projects and locate the Deployment Framework for BizTalk Project template.

1c. Select the Deployment Framework for BizTalk Project template and, in the Name textbox, enter the value HelloWorld.Deployment. Ensure that the Location textbox contains the path C:\BTDF\Walkthrough1.

1d. Click OK to create the deployment project.

Step 2: Configure the initial deployment project settings

The next screen you see is the Deployment Framework for BizTalk Project Options dialog box. This dialog contains the most commonly used -- but not all -- Deployment Framework options. All of the options that you see here can also be changed later in the project file.

Since the HelloWorld project combines schemas, a map and an orchestration into one assembly, and the Deployment Framework normally has separate deployment options for those three artifact types, we must choose how to deploy them as one unit. Whenever a BizTalk assembly has schemas in it, the best option is to treat the entire assembly as a "schemas" assembly. That's because maps, orchestrations and most other artifacts in BizTalk are all dependent on schemas.

2a. In the Project Options dialog, set "Deploy orchestrations?" to False and set "Deploy transforms/maps?" to False. "Deploy schemas?" is set to True by default. Also set "Skip IIS/AppPool reset during deploy?" to True since this application has nothing to do with IIS.

2b. Click the Create Project button to complete the creation of the deployment project. A dialog box will appear confirming the project creation, and the deployment project file (Deployment.btdfproj) will open in the text editor. Click OK to close the confirmation dialog.

Step 3: Add the deployment project files to the solution

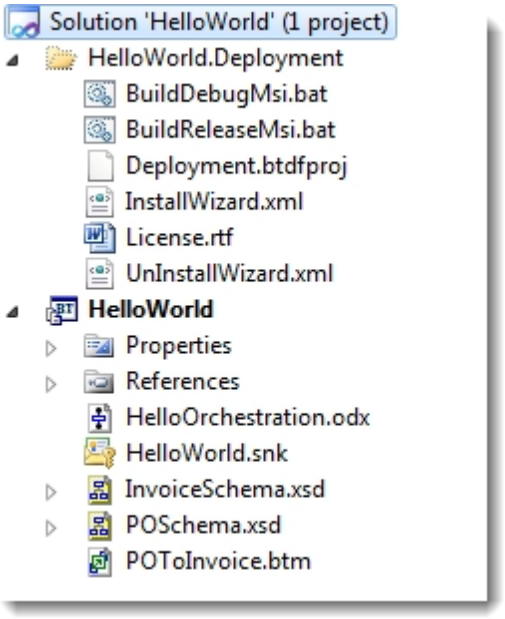
The Deployment Framework for BizTalk does not include a full-blown custom Visual Studio project type (like C#, VB.NET, etc.), so its project files do not appear as a project in Solution Explorer. However, it is easy to add the individual files to the solution for ease of access and source control integration.

3a. In Solution Explorer, right-click the HelloWorld *solution* and choose the Add > New Solution Folder command. Replace the default folder name with HelloWorld.Deployment.

3b. Next, right click the new HelloWorld.Deployment solution folder and choose the Add > Existing Item... command. Browse to the C:\BTDF\Walkthrough1\HelloWorld.Deployment folder and select all seven files (two .bat files, one .btdfproj, three .xml and one .rtf). Click the Add button to add the files to Solution Explorer.

3c. Optionally, repeat the same Add > Existing Item... process to add the file HelloWorld.Deployment \EnvironmentSettings\SettingsFileGenerator.xml. When a solution is bound to source control, you may find that this file randomly disappears from the solution (which is automatically checked out), so sometimes it is easier to leave this file out of the solution and just ensure that it is added to source control.

Your solution should now look like this:



Step 3: Customize the BTDF Project File

You should now have the HelloWorld solution from C:\BTDF\Walkthrough1 open in Visual Studio and successfully built, with the newly created Deployment Framework for BizTalk project files added to Solution Explorer. The next step is to customize the Deployment Framework for BizTalk project file.

Step 1: Configure the Schemas ItemGroup to point to the HelloWorld assembly

In order to deploy a BizTalk assembly (and any other files that you may include in a deployment), you must tell the Deployment Framework for BizTalk where to find your application's files.

1a. In Solution Explorer, double-click Deployment.btdfproj to open it in the text/XML editor.

1b. Locate the XML element <Schemas Include="YourSchemas.dll">. This is a placeholder added by the Add New Project wizard.

1c. In this application, our BizTalk assembly is simply named HelloWorld.dll. Replace the default attribute value for Include from YourSchemas.dll to HelloWorld.dll.

File paths are always *relative to the deployment project folder*. In this case, the path to HelloWorld.dll is ..\HelloWorld\bin\Debug. The default path in LocationPath is ..\$(ProjectName)\bin\$(Configuration). The two $() references are MSBuild properties. $(Configuration) is automatically set to Debug or Release based on the active solution configuration. $(ProjectName) already has a value of HelloWorld -- you'll find the element <ProjectName>HelloWorld</ProjectName> near the top of the project file. In this case, the default path is perfect and does not need to be modified.

You should now have this XML:

```
<ItemGroup>
  <Schemas Include="HelloWorld.dll">
    <LocationPath>..\$(ProjectName)\bin\$(Configuration)</LocationPath>
  </Schemas>
</ItemGroup>
```

Save the changes to Deployment.btdfproj.

Step 2: Customize other project settings

At this point you would add additional ItemGroup elements to point to your other artifacts such as schemas assemblies, BRE policy or vocabulary XML files, etc. IntelliSense is readily available to help you create the correct XML structure. The HelloWorld application is so simple that there is nothing else to add, so we do not need to do additional configuration for this walkthrough.

Step 4: Configure Bindings File

You should now have the HelloWorld solution from C:\BTDF\Walkthrough1 open in Visual Studio and successfully built, with the Deployment Framework for BizTalk project files added to Solution Explorer. Your project file should now be customized to point to HelloWorld.dll. The next step is to deploy the application for the first time and set up a port bindings XML file.

Step 1: Deploy the application for the first time

Now that you have indicated to the Deployment Framework what you want to deploy (properties in the PropertyGroup) and where your application file(s) are located (the Schemas ItemGroup), you are ready to deploy your application for the first time. Since you have not yet defined the bindings for the application, the application *cannot start*, but it can be deployed.

1a. On the Visual Studio main menu bar, open Tools > Deployment Framework for BizTalk and choose the "Deploy BizTalk Solution" command.

1b. Watch the deployment script execute by opening the Visual Studio Output window. (You can find a link in the main menu bar's View menu.)

1c. The build **will fail** due to the incomplete port bindings. You should see the message:

```
Starting HelloWorld application...
C:\Program Files\MSBuild\DeploymentFrameworkForBizTalk\5.0
\BizTalkDeploymentFramework.targets(1831,5): error : Could not enlist orchestration
'Microsoft.Samples.BizTalk.HelloWorld.HelloSchedule,HelloWorld, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=cd2d694226bed292'. Could not enlist orchestration
'Microsoft.Samples.BizTalk.HelloWorld.HelloSchedule'. All orchestration ports must be
bound and the host must be set. [C:\BTDF\Walkthrough1\HelloWorld.Deployment
\Deployment.btdfproj]
Done Building Project "C:\BTDF\Walkthrough1\HelloWorld.Deployment
\Deployment.btdfproj" (Deploy target(s)) -- FAILED.
```

Step 2: Configure bindings in BizTalk Server Administration tool

The application is now fully deployed into BizTalk and is almost ready to go. Before the deployment can be fully automated, you need to manually configure all of the port bindings using the BizTalk Server Administration tool, then export the completed bindings into your deployment project folder.

2a. Open the BizTalk Server Administration Console, then expand the BizTalk Group and Applications nodes. Locate an application named HelloWorld.

2b. Right-click the HelloWorld application and select the Configure... command. In the Configure Application dialog, select the HelloSchedule node.

2c. In the Host drop-down, choose BizTalkServerApplication (assuming a default BizTalk configuration).

2d. Create and configure a receive port and location:

1. Under Receive Ports, choose <New receive port...> from the drop-down. In the receive port dialog, change the Name to HWReceivePort.
2. Select the "Receive Locations" node from the left pane. Click the New... button to create a receive location. In the receive location dialog, change the name to HWReceiveLocation.

3. Choose the FILE adapter from the Type drop-down list. Click the Configure... button, then enter "C:\BTDF\Walkthrough1\In" as the Receive Folder. Click OK.
4. In the Receive Pipeline drop-down, choose the XmlReceive pipeline. Click OK.
5. Click OK again to complete the creation of the receive port and location.

2e. Create and configure a send port:
1. Under Send Ports, choose <New send port...> from the drop-down. In the send port dialog, change the Name to HWSendPort.
2. Choose the FILE adapter from the Type drop-down list. Click the Configure... button, then enter "C:\BTDF\Walkthrough1\Out" as the Destination Folder. Click OK.
3. Click OK again to complete the creation of the send port.

2f. Click the OK button to complete configuration of the application.

2g. Right-click the HelloWorld application and select the Start... command. Click the Start button to start the application. The application should now be fully deployed, started and functional.

Step 3: Test the application

Now that the application is started, we can try sending a PO XML file through it by dropping a file into the In folder. We should see an invoice XML appear in the Out folder.

3a. Copy the file C:\BTDF\Walkthrough1\HelloWorld\SamplePOInput.xml into C:\BTDF\Walkthrough1\In.

3b. Verify that a new XML file named {SomeGUID}.xml appears in C:\BTDF\Walkthrough1\Out.

3c. If the input file did not disappear or the output file did not appear, then follow standard BizTalk debugging practices to solve the problem before continuing. Check the Windows Application Event Log first.

Step 4: Export the complete port bindings to an XML file

Back in the BizTalk Server Administration Console, right-click the HelloWorld application and choose the Export > Bindings... command. Enter the file path C:\BTDF\Walkthrough1\HelloWorld.Deployment\PortBindingsMaster.xml and click the OK button. When prompted whether to overwrite the existing file, click the Yes button.

Step 5: Undeploy the application

We have completed our goal of manually configuring the port bindings and exporting them to a file within the deployment project. Next, we can undeploy the application and then perform a full deployment to test the new bindings file.

5a. On the Visual Studio main menu bar, open Tools > Deployment Framework for BizTalk and choose the "Undeploy BizTalk Solution" command.

5b. Watch the deployment script execute by opening the Visual Studio Output window. The script should complete successfully:

```
Done Building Project "C:\BTDF\Walkthrough1\HelloWorld.Deployment
\Deployment.btdfproj" (Undeploy target(s)).

Build succeeded.
    0 Warning(s)
    0 Error(s)
```


Step 5: Test Local Deployment

You should now have the HelloWorld solution from C:\BTDF\Walkthrough1 open in Visual Studio and successfully built, with a fully configured port bindings XML file. The next step is to deploy the application and verify the success of a fully automated deployment.

Step 1: Deploy the fully-configured application

Now that your Deployment Framework for BizTalk project is fully configured, you are ready to test a completely automated deployment. The application should now deploy and start successfully.

1a. On the Visual Studio main menu bar, open Tools > Deployment Framework for BizTalk and choose the "Deploy BizTalk Solution" command.

1b. Watch the deployment script execute by opening the Visual Studio Output window.

1c. The build should succeed with the message:

```
Build succeeded.

"C:\BTDF\Walkthrough1\HelloWorld.Deployment\Deployment.btdfproj" (Deploy target) (1) ->
(DeploySchemas target) ->
  EXEC : warning : If any of the assemblies were previously loaded by a Host Instance, it
may be necessary to restart the Host Instance for changes to take effect. [C:\BTDF
\Walkthrough1\HelloWorld.Deployment\Deployment.btdfproj]

    1 Warning(s)
    0 Error(s)
```

1d. Disregard the warning because the deployment process already restarted the host instances.

Step 2: Test the application

Try sending a PO XML file through the application by dropping a file into the In folder. We should see an invoice XML appear in the Out folder.

2a. Copy the file C:\BTDF\Walkthrough1\HelloWorld\SamplePOInput.xml into C:\BTDF\Walkthrough1\In.

2b. Verify that a new XML file named {SomeGUID}.xml appears in C:\BTDF\Walkthrough1\Out.

2c. If the input file did not disappear or the output file did not appear, then follow standard BizTalk debugging practices to solve the problem before continuing. Check the Windows Application Event Log first.

Congratulations! You've successfully created and configured your first Deployment Framework for BizTalk project!

Step 6: Build MSI for Server Deployment

You should now have the HelloWorld solution from C:\BTDF\Walkthrough1 open in Visual Studio and successfully built. The next step is to package the application for deployment to a BizTalk server.

Package the application into an MSI

Before deploying your application to a BizTalk server, you need to package it up into an MSI.

a. On the Visual Studio main menu bar, open Tools > Deployment Framework for BizTalk and choose the "Build Server Deploy MSI" command.

b. Watch the deployment script execute by opening the Visual Studio Output window.

c. The build should succeed with the message:

```
Build succeeded.
    0 Warning(s)
    0 Error(s)
```

d. In Windows Explorer, open C:\BTDF\Walkthrough1\HelloWorld.Deployment\bin\Debug. You will find two files: Install-HelloWorld-1.0.0.bat and HelloWorld-1.0.0.msi. These files can be copied directly to a BizTalk server for installation and deployment.

The .bat file is not necessary unless you need to pre-populate Windows Installer property values (like the installation path) from the command-line.

TIP: You can build the solution and MSI under TFS Team Build.

Step 7: Test MSI Deployment

You should now have the HelloWorld solution packaged into an MSI in C:\BTDF\Walkthrough1 \HelloWorld.Deployment\bin\Debug. The next step is to test the MSI installation and deployment.

Step 1: Undeploy the application

We can test the MSI installation and deployment on the local machine even though it is intended for a true BizTalk server. If you have not yet undeployed the application, it must be removed before we can test the MSI.

1a. On the Visual Studio main menu bar, open Tools > Deployment Framework for BizTalk and choose the "Undeploy BizTalk Solution" command.

1b. Watch the deployment script execute by opening the Visual Studio Output window. The script should complete successfully.

Step 2: Install the MSI

Deployment using an MSI is a two-step process -- install the MSI, then execute the deployment script.

2a. Double-click C:\BTDF\Walkthrough1\HelloWorld.Deployment\bin\Debug\HelloWorld-1.0.0.msi to start the installation process.

2b. Click Next through the wizard, accepting the default values, then click the Install button.

2c. Before clicking Finish to close the installation wizard, proceed to the next step.

Step 3: Deploy the application

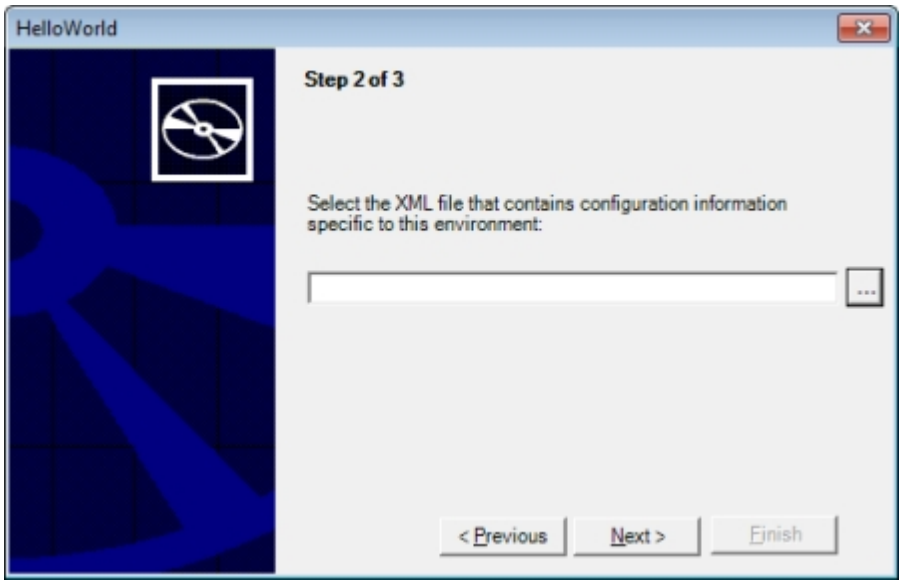
Now that the application files have been physically delivered to the machine with the MSI, the application must be deployed into BizTalk.

3a. On the final page of the installation wizard, check the checkbox labeled "Start deploying to BizTalk when I click Finish," then click the Finish button.

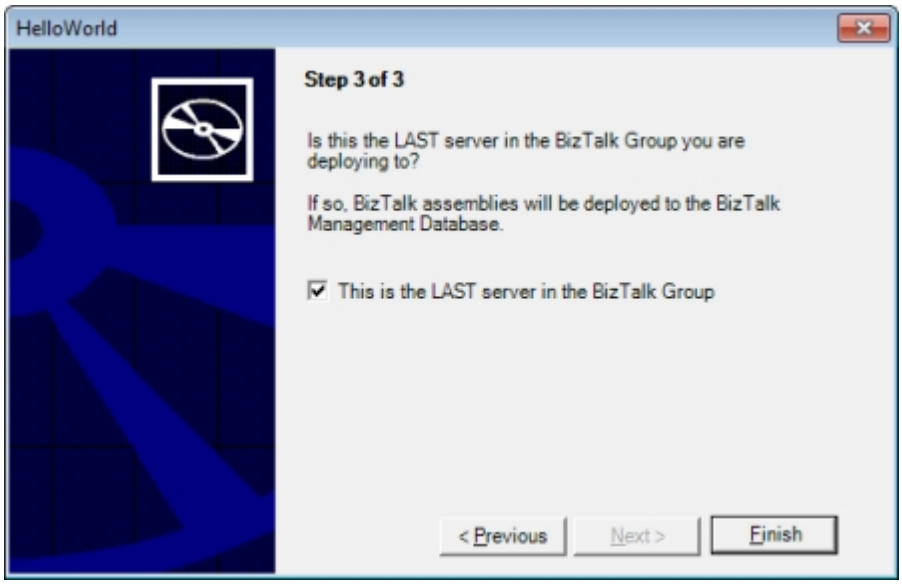
3b. On Windows Vista and newer, you may be prompted for UAC elevation to Administrator privileges. Click Yes to allow.

3c. The deployment wizard, below, appears to begin the deployment. This wizard is driven by C:\BTDF \Walkthrough1\HelloWorld.Deployment\InstallWizard.xml. The first default step asks for a domain- or computer-qualified user name (MACHINEorDOMAIN\Username). This user account will be granted full control permissions on any paths associated with the FILE adapter in the port bindings. Enter the user name that is used by your host instance(s) and click the Next button.

3d. The next page, below, asks for the path to an XML configuration file. This will be one of the environment-specific XML files exported from the Excel settings spreadsheet EnvironmentSettings \SettingsFileGenerator.xml. In this case, we are deploying to your development (local) machine, so select C:\BTDF\Walkthrough1\HelloWorld.Deployment\EnvironmentSettings\Exported_LocalSettings.xml and click the Next button.



3e. The final page, below, asks if this is the last server in the BizTalk group. Since this is just your development machine, it's the last server in the group. Click the Finish button to start the deployment script. You will see the MSBuild script output appear in a Command Prompt window.



3f. The build should succeed with the message:

```
Build succeeded.

"C:\Program Files\HelloWorld for BizTalk\1.0\Deployment\Deployment.btdfproj" (Deploy
target) (1) ->
(DeploySchemas target) ->
  EXEC : warning : If any of the assemblies were previously loaded by a Host Instance, it
may be necessary to restart the Host Instance for changes to take effect. ["C:\Program
Files\HelloWorld for BizTalk\1.0\Deployment\Deployment.btdfproj]

    1 Warning(s)
    0 Error(s)
```

Disregard the warning because the deployment process already restarted the host instances.

Step 4: Test the application

Try sending a PO XML file through the application by dropping a file into the In folder. We should see an invoice XML appear in the Out folder.

4a. Copy the file C:\BTDF\Walkthrough1\HelloWorld\SamplePOInput.xml into C:\BTDF\Walkthrough1\In.

4b. Verify that a new XML file named {SomeGUID}.xml appears in C:\BTDF\Walkthrough1\Out.

4c. If the input file did not disappear or the output file did not appear, then follow standard BizTalk debugging practices to solve the problem before continuing. Check the Windows Application Event Log first.

Step 5: Undeploy the application

To undeploy the application, open the Start > Programs menu and locate HelloWorld for BizTalk 1.0. Choose the command "Undeploy HelloWorld" to launch the deployment wizard in undeploy mode. Click the Finish button to undeploy the application.

After the script completes, use the Windows Add/Remove Programs Control Panel to uninstall the "HelloWorld for BizTalk 1.0.0" application.

This concludes the walkthrough! You have learned how to create and configure a basic Deployment Framework for BizTalk project, configure a bindings XML file, deploy, test and undeploy the application both through Visual Studio and an MSI.

Sample Applications

The Deployment Framework for BizTalk includes a number of sample applications that can help you get up to speed quickly. They focus on subsets of functionality to help you understand how to configure various types of applications.

In this section

- HelloWorld
- BasicMasterBindings
- BAM
- Advanced
- ESBToolkitSSOResolver

HelloWorld

The HelloWorld sample demonstrates the most basic implementation of the Deployment Framework for BizTalk on a very simple, single-assembly BizTalk application.

What this Sample Does

The HelloWorld sample is based on the Orchestrations HelloWorld sample included in the BizTalk Server SDK. The documentation for the BizTalk Server 2010 version of the sample application (it hasn't changed through many BizTalk releases) may be found here. It is a simple application that picks up a file from a file system folder, runs it through an orchestration and writes it out to another file system folder.

How this Sample is Designed and Why

This sample demonstrates a very stripped-down and compact implementation of the Deployment Framework on a BizTalk application that consists of one assembly. It uses a standard BizTalk XML bindings file, so it does not take advantage of the Deployment Framework's environment-specific configuration features. All non-essential features are disabled in this sample. Even so, the sample still demonstrates the ease with which you can deploy and test a BizTalk project from Visual Studio, then package it as an MSI for deployment to your BizTalk servers.

One extensibility feature demonstrated in the sample's HelloWorld.Deployment.btdfproj project file is how to package additional files into the MSI so that they are deployed to the server along with the application binaries and other project files. The CustomRedist target is called during MSI packaging, and in the sample it copies the files in the TestFiles folder into the MSI staging folder (defined by the $(RedistDir) MSBuild property).

Where to Find this Sample

<BTDFInstallFolder>\Samples\BizTalk2006\HelloWorld or <BTDFInstallFolder>\Samples\BizTalk2009\HelloWorld

NOTE: The BizTalk 2006/2006 R2 version of the sample must be prepared by running PrepareSample.bat in the Samples\BizTalk2006\HelloWorld folder.

Building and Deploying this Sample

1. Open the solution file HelloWorld.sln in Visual Studio. If necessary, upgrade the solution and projects to Visual Studio 2010 format.
2. Build the solution and ensure that the build succeeded
3. Deploy the application using the Deployment Framework's Visual Studio Deploy command
4. The Deployment Framework will create an input folder at C:\DeploymentFrameworkForBizTalk\Samples

\HelloWorld\In
5. Confirm that the deployment process succeeded and the application is now running in BizTalk (see the Visual Studio Output window and BizTalk Admin Console)
6. If desired, package the application into an MSI using the Build Server MSI command

Running this Sample

1. Copy the file SamplePOInput.xml from TestFiles into C:\DeploymentFrameworkForBizTalk\Samples\HelloWorld\In
2. Watch for an output XML file to appear in C:\DeploymentFrameworkForBizTalk\Samples\HelloWorld\Out

Removing this Sample

1. With the HelloWorld.sln open in Visual Studio, use the Deployment Framework's Visual Studio Undeploy command
2. Delete the folder C:\DeploymentFrameworkForBizTalk\Samples\HelloWorld

BasicMasterBindings

The BasicMasterBindings sample demonstrates a typical starter implementation of the Deployment Framework for BizTalk on a simple, single-assembly BizTalk application.

What this Sample Does

The BasicMasterBindings sample is based on the Orchestrations HelloWorld sample included in the BizTalk Server SDK. The documentation for the BizTalk Server 2010 version of the sample application (it hasn't changed through many BizTalk releases) may be found here. It is a simple application that picks up a file from a file system folder, runs it through an orchestration and writes it out to another file system folder.

How this Sample is Designed and Why

This sample demonstrates a typical entry-level implementation of the Deployment Framework on a BizTalk application that consists of one assembly. It uses the Deployment Framework's environment-specific configuration features to dynamically transform a template bindings XML file into an environment-specific bindings file. Most non-essential features are disabled in this sample. The sample demonstrates the ease with which you can deploy and test a BizTalk project from Visual Studio, then package it as an MSI for deployment to your BizTalk servers.

The BasicMasterBindings.Deployment\EnvironmentSettings folder contains a file named SettingsFileGenerator.xml. This is actually an Excel 2003+ spreadsheet saved in XML format. This spreadsheet gives you a single place to store environment-specific configuration information. During deployment, the environment-specific values are exported from the spreadsheet to a set of XML files (their filenames appear at the top of the spreadsheet). When you deploy from Visual Studio, the Local settings are always used. When you deploy to a server using an MSI, you can select any of the exported, environment-specific XML files.

The bindings file is named PortBindingsMaster.xml. It is a standard BizTalk bindings file with a minor twist -- it's generic vs. specific to a particular environment. Inside the file, you'll find tokens such as ${InvoiceOutputPath}. Referring to the Excel spreadsheet, you'll find a corresponding row named InvoiceOutputPath. At deployment time, a tool named XmlPreprocess will dynamically combine PortBindingsMaster.xml with the selected environment-specific XML file to produce an environment-specific bindings file named PortBindings.xml.

One extensibility feature demonstrated in the sample's BasicMasterBindings.Deployment.btdfproj project file is how to package additional files into the MSI so that they are deployed to the server along with the application binaries and other project files. The CustomRedist target is called during MSI packaging, and in the sample it copies the files in the TestFiles folder into the MSI staging folder (defined by the $(RedistDir) MSBuild property).

Where to Find this Sample

<BTDFInstallFolder>\Samples\BizTalk2006\BasicMasterBindings or <BTDFInstallFolder>\Samples\BizTalk2009\BasicMasterBindings

NOTE: The BizTalk 2006/2006 R2 version of the sample must be prepared by running PrepareSample.bat in the Samples\BizTalk2006\BasicMasterBindings folder.

Building and Deploying this Sample

1. Open the solution file BasicMasterBindings.sln in Visual Studio. If necessary, upgrade the solution and projects to Visual Studio 2010 format.
2. Build the solution and ensure that the build succeeded
3. Deploy the application using the Deployment Framework's Visual Studio Deploy command
4. The Deployment Framework will create an input folder at C:\DeploymentFrameworkForBizTalk\Samples\BasicMasterBindings\In
5. Confirm that the deployment process succeeded and the application is now running in BizTalk (see the Visual Studio Output window and BizTalk Admin Console)
6. If desired, package the application into an MSI using the Build Server MSI command

Running this Sample

1. Copy the file SamplePOInput.xml from TestFiles into C:\DeploymentFrameworkForBizTalk\Samples\BasicMasterBindings\In
2. Watch for an output XML file to appear in C:\DeploymentFrameworkForBizTalk\Samples\BasicMasterBindings\Out

Removing this Sample

1. With the BasicMasterBindings.sln open in Visual Studio, use the Deployment Framework's Visual Studio Undeploy command
2. Delete the folder C:\DeploymentFrameworkForBizTalk\Samples\BasicMasterBindings

BAM

The BAM sample demonstrates how to use the Deployment Framework for BizTalk with a BAM model, tracking profile, custom pipeline and custom pipeline component.

What this Sample Does

The BAM sample is a simplified version of the BamEndToEnd sample included in the BizTalk Server SDK. The documentation for the BizTalk Server 2010 version of the sample application (it hasn't changed through many BizTalk releases) may be found here. This application picks up a file from a file system folder, runs it through a custom pipeline and an orchestration and writes it out to another file system folder. Along the way, data is fed into a BAM model via the BAM API and a tracking profile.

How this Sample is Designed and Why

This sample demonstrates deployment of a custom pipeline, a custom pipeline component, an assembly containing both schemas and orchestrations, a BAM model and a BAM tracking profile. It uses the Deployment Framework for BizTalk's environment-specific configuration features to dynamically transform a template bindings XML file into an environment-specific bindings file (see BasicMasterBindings sample).

The Excel settings spreadsheet, SettingsFileGenerator.xml, contains an extra setting used for BAM deployments: BAMViewsAndAccounts. This setting allows you to specify the users and/or groups that will be granted access to the BAM view(s). The setting value is translated into an MSBuild property for use during the build by including it in a PropsFromEnvSettings ItemGroup in Deployment.btdfproj.

One extensibility feature demonstrated in the sample's Deployment.btdfproj project file is how to package additional files into the MSI so that they are deployed to the server along with the application binaries and other project files. The CustomRedist target is called during MSI packaging, and in the sample it copies the files in the TestFiles folder into the MSI staging folder (defined by the $(RedistDir) MSBuild property).

Where to Find this Sample

<BTDFInstallFolder>\Samples\BizTalk2006\BAM or <BTDFInstallFolder>\Samples\BizTalk2009\BAM

NOTE: The BizTalk 2006/2006 R2 version of the sample must be prepared by running PrepareSample.bat in the Samples\BizTalk2006\BAM folder.

NOTE: After upgrading the solution to BizTalk 2010 or newer, you must upgrade the .NET class library projects to target .NET Framework 4.0.

Building and Deploying this Sample

1. Open the solution file BAM.sln in Visual Studio. If necessary, upgrade the solution and projects to Visual Studio 2010 format.
2. Build the solution and ensure that the build succeeded
3. Deploy the application using the Deployment Framework's Visual Studio Deploy command
4. The Deployment Framework will create an input folder at C:\DeploymentFrameworkForBizTalk\Samples\BAM\In
5. Confirm that the deployment process succeeded and the application is now running in BizTalk (see the Visual Studio Output window and BizTalk Admin Console)
6. If desired, package the application into an MSI using the Build Server MSI command

Running this Sample

1. Copy one or more of the InputMessageXX.xml files from TestFiles into C:\DeploymentFrameworkForBizTalk\Samples\BAM\In
2. Watch for an output XML file to appear in C:\DeploymentFrameworkForBizTalk\Samples\BAM\Out
3. The captured BAM data should be visible in the BAM Portal. If you don't see the activity, you may need to grant yourself rights to the view.

Removing this Sample

1. With the BAM.sln open in Visual Studio, use the Deployment Framework's Visual Studio Undeploy command
2. Delete the folder C:\DeploymentFrameworkForBizTalk\Samples\BAM

ESBToolkitSSOResolver

The ESBToolkitSSOResolver sample demonstrates how to use the Deployment Framework for BizTalk with an ESB Toolkit itinerary that resolves adapter information at runtime from settings stored in SSO.

NOTE: This sample requires BizTalk 2009 or newer with the ESB Toolkit installed **and configured**. On BizTalk 2010 or newer, you must also install the ESB Toolkit Itinerary Designer Extension.

What this Sample Does

The ESBToolkitSSOResolver sample picks up a file from a file system folder and runs it through the ESB Toolkit's ItinerarySelectReceiveXml pipeline in order to assign an itinerary named ESBToolkitSSOResolver. The itinerary calls the POToInvoice map to transform the input message, then uses the Deployment Framework for BizTalk SSO Resolver to dynamically configure an off-ramp, which then writes the transformed file to another file system folder.

How this Sample is Designed and Why

This sample demonstrates deployment of a single BizTalk assembly containing both schemas and a map, a custom ESB Toolkit itinerary and the Deployment Framework for BizTalk SSO Resolver. It uses the Deployment Framework for BizTalk's environment-specific configuration features to dynamically transform a template bindings XML file into an environment-specific bindings file (see BasicMasterBindings sample). Building on the same configuration features, it dynamically reads settings from SSO at runtime to configure

the ESB off-ramp.

The Excel settings spreadsheet, SettingsFileGenerator.xml, contains a couple of settings used to configure the off-ramp: InvoiceOutputPath and InvoiceOutputTransport. During deployment a particular environment's setting values are loaded into SSO, and the ESB Resolver reads them at runtime using the application's name (the value of the ProjectName property in the deployment project file).

One extensibility feature demonstrated in the sample's Deployment.btdfproj project file is how to package additional files into the MSI so that they are deployed to the server along with the application binaries and other project files. The CustomRedist target is called during MSI packaging, and in the sample it copies the files in the TestFiles folder into the MSI staging folder (defined by the $(RedistDir) MSBuild property).

Another extensibility feature demonstrated in the project file is how to create a file system folder at deployment time -- in this case the output folder. The CustomDeployTarget is called during application deployment, and in the sample it creates the output folder. The folder path is automatically pulled from the settings spreadsheet (named InvoiceOutputPath) into an MSBuild property via the PropsFromEnvSettings ItemGroup.

Where to Find this Sample

<BTDFInstallFolder>\Samples\BizTalk2009\ESBToolkitSSOResolver

NOTE: After upgrading the solution to BizTalk 2010 or newer, you must upgrade the ESBToolkitSSOResolver.Itineraries project to target .NET Framework 4.0.

Building and Deploying this Sample

1. Open the solution file ESBToolkitSSOResolver.sln in Visual Studio. If necessary, upgrade the solution and projects to Visual Studio 2010 format.
2. Build the solution and ensure that the build succeeded
3. On BizTalk 2010, open the file Deployment\PortBindingsMaster.xml in Notepad, replace Version=2.0.0.0 with Version=2.1.0.0 and save the file.
4. Deploy the application using the Deployment Framework's Visual Studio Deploy command
5. The Deployment Framework will create an input folder at C:\DeploymentFrameworkForBizTalk\Samples\ESBToolkitSSOResolver\In
6. Confirm that the deployment process succeeded and the application is now running in BizTalk (see the Visual Studio Output window and BizTalk Admin Console)
7. If desired, package the application into an MSI using the Build Server MSI command

Running this Sample

1. Copy the file SamplePOInput.xml from TestFiles into C:\DeploymentFrameworkForBizTalk\Samples\ESBToolkitSSOResolver\In
2. Watch for an output XML file to appear in C:\DeploymentFrameworkForBizTalk\Samples\ESBToolkitSSOResolver\Out

Removing this Sample

1. With the ESBToolkitSSOResolver.sln open in Visual Studio, use the Deployment Framework's Visual Studio Undeploy command
2. Delete the folder C:\DeploymentFrameworkForBizTalk\Samples\ESBToolkitSSOResolver

Advanced

The Advanced sample demonstrates how to use the Deployment Framework for BizTalk with multiple orchestrations, C# helper classes, runtime access to SSO settings, virtual directory deployment, the BizTalk HTTP ISAPI filter, log4net logging, pre-processing an XML file and NUnit deployment tests.

What this Sample Does

This application contains two separate functions: one that receives a message through the HTTP ISAPI filter and returns the same message in the response, and a second that takes in a file and writes out another file while processing it through orchestrations and maps.

How this Sample is Designed and Why

The first function of this sample is a message echo via the HTTP adapter (an IIS ISAPI filter). An NUnit test posts an XML message to the HTTP adapter and an orchestration named Echo.odx receives it and sends back the same message. The NUnit test validates that the message was echoed back. This part of the sample demonstrates deployment of an IIS virtual directory (the VDir_Advanced folder in the sample directory) and an IIS ISAPI filter. In the deployment project (.btdfproj) file, the properties IncludeVirtualDirectories, WseExtensionPath and WseExtensionName and the ItemGroup VDirList are all related to these functions. The two Wse* properties are only used to deploy the BizTalk HTTP adapter ISAPI filter, so they are not required when deploying a virtual directory with a .NET web service.

The sample uses the Deployment Framework for BizTalk's environment-specific configuration features to dynamically transform a template bindings XML file into an environment-specific bindings file (see BasicMasterBindings sample). The Excel settings spreadsheet, SettingsFileGenerator.xml, also contains a number of settings used to demonstrate reading values from SSO at runtime.

The second function of this sample is to read in a file, then pass it through an orchestration while writing to Log4Net, calling out to a C# helper component to construct a message and dynamically reading settings from SSO, and finally writing a file to an output folder. An NUnit test copies the input file TestFiles \S1_output.xml to the InDir folder within the sample solution folder (which you are free to do manually). The file is handled by the orchestration TopLevelOrch.odx, which configures and writes to a Log4Net logger and reads some settings from the SSO database (which originated in the Excel settings spreadsheet SettingsFileGenerator.xml). The orchestration calls CalledOrch.odx to transform the message, and the final message is written out to a file at C:\DeploymentFrameworkForBizTalk\Samples\Advanced\OutDir. The NUnit test waits for the output file to appear and fails if it does not.

One extensibility feature demonstrated in the sample's Deployment.btdfproj project file is how to package additional files into the MSI so that they are deployed to the server along with the application binaries and other project files. The CustomRedist target is called during MSI packaging, and in the sample it copies the files in the TestFiles folder into the MSI staging folder (defined by the $(RedistDir) MSBuild property).

Where to Find this Sample

<BTDFInstallFolder>\Samples\BizTalk2006\Advanced or <BTDFInstallFolder>\Samples\BizTalk2009 \Advanced

NOTE: The BizTalk 2006/2006 R2 version of the sample must be prepared by running PrepareSample.bat in the Samples\BizTalk2006\Advanced folder.
NOTE: After upgrading the solution to BizTalk 2010 or newer, you must upgrade the .NET class library project to target .NET Framework 4.0.

Building and Deploying this Sample

1. Open the solution file Advanced.sln in Visual Studio. If necessary, upgrade the solution and projects to Visual Studio 2010 format.
2. Build the solution and ensure that the build succeeded
3. On a 32-bit system, copy BTSHTTPReceive.dll from <BizTalkInstallFolder>\HttpReceive to the Advanced sample's VDir_Advanced folder and overwrite the existing file. On a 64-bit system, copy from HttpReceive64.
4. Deploy the application using the Deployment Framework's Visual Studio Deploy command
5. On a 64-bit system, in IIS Manager, configure the BTDFAdvancedSample application pool to run in 64-bit mode (Enable 32-bit Applications = False).
6. In IIS Manager, configure the BTDFAdvancedSample application pool identity to run as your BizTalk service account (same as your host instances).
7. On a 64-bit system, add a 64-bit copy of the log4net registry entry (see NOTE below).
8. The Deployment Framework will create an output folder at C:\DeploymentFrameworkForBizTalk

\Samples\Advanced\OutDir
9. Confirm that the deployment process succeeded and the application is now running in BizTalk (see the Visual Studio Output window and BizTalk Admin Console)
10. If desired, package the application into an MSI using the Build Server MSI command

NOTE: If you are running on x64 Windows with 64-bit hosts, see the Log4Net registry key note here. The orchestrations will fail unless the 64-bit registry key is created.

Running this Sample

1. Run the deployment tests by starting NUnit from <BTDFInstallFolder>\Framework\DeployTools \NUnitSubset\nunit-gui.exe, then load the test assembly <ProjectFolder>\DeploymentTest\bin\Debug \DeploymentFramework.Samples.Advanced.DeploymentTest.dll.
2. Verify that the NUnit tests ran successfully.

Removing this Sample

1. With the Advanced.sln open in Visual Studio, use the Deployment Framework's Visual Studio Undeploy command
2. Delete the folder C:\DeploymentFrameworkForBizTalk\Samples\Advanced
3. Delete the IIS application pool BTDFAdvancedSample.

Advanced Topics

This section covers less common but still important features of the Deployment Framework for BizTalk, including how to customize your deployment process by adding your own MSBuild targets.

In this section

- Customizing Deployments with Custom Targets
- Using Log4Net
- Modifying BizTalk Server Configuration
- Deploying Multiple Application Versions Side-by-Side
- Controlling .NET App Domains

Customizing Deployments with Custom Targets

The Deployment Framework for BizTalk contains a huge amount of functionality out of the box, but there are always cases where you need to modify or add your own customizations to the process. There are many extensibility points built in to the Deployment Framework for just this reason.

NOTE: This topic applies mainly to BizTalk 2009 and earlier. In MSBuild 4.0 (BizTalk 2010+), the Target element has new attributes BeforeTargets and AfterTargets that let you very precisely hook into the execution order. See the MSBuild Target Build Order documentation for more.

Customizations are implemented via MSBuild targets in your deployment project file (.btdfproj). Below you'll find a list of all possible target names and when they execute. You must use the exact name specified here for your target to be recognized. Place the Target element **after** the Import element.

Name	When	Description
CustomPreInitialize	Deploy; Undeploy	Runs at the absolute beginning of the process, before the Deployment Framework has done any initialization
CustomPostInitialize	Deploy; Undeploy	Runs just after the Deployment Framework has initialized itself by loading required MSBuild properties and exporting settings files from the settings spreadsheet
CustomPreExportSettings	Deploy; Undeploy	Runs just before the Deployment Framework exports settings files from the settings spreadsheet
CustomPostExportSettings	Deploy; Undeploy	Runs just after the Deployment Framework exports settings files from the settings spreadsheet
CustomDeployTarget	Deploy	Runs just after the Deployment Framework has initialized itself, preprocessed files with XmlPreprocess and prepared the bindings file and just before the empty BizTalk application is created
CustomPostDeployTarget	Deploy	Runs just after the Deployment Framework has deployed all artifacts and started the BizTalk application and just before the BizTalk hosts are restarted
CustomUndeployTarget	Undeploy	Runs just after the Deployment Framework has initialized itself and just before any undeployment steps occur
CustomPostUndeployTarget	Undeploy	Runs just after the Deployment Framework has undeployed all artifacts and just before the BizTalk hosts are restarted
CustomSSO	Deploy	Runs during SSO deployment, just after the settings file is imported into SSO

Name	When	Description
CustomPreRedist	MSI Build	Runs just after the Deployment Framework has initialized itself by loading required MSBuild properties
CustomRedist	MSI Build	Runs just after the Deployment Framework has finished copying files to the MSI staging folder ($(RedistDir)) and just before the MSI is built
CustomPostInstaller	MSI Build	Runs just after the MSI build has completed

Here's an example extension target that copies an entire folder of test files for inclusion into the MSI:

```
<Target Name="CustomRedist">
  <MakeDir Directories="$(RedistDir)\TestFiles" />

  <!-- Force MSBuild to expand the item spec into physical file specs -->
  <CreateItem Include="..\TestFiles\**\*.*">
    <Output TaskParameter="Include" ItemName="TestFilesSourceGroup" />
  </CreateItem>

<!-- Copy all of the files and subfolders from ..\TestFiles to $(RedistDir)\TestFiles -->
  <Copy DestinationFolder="$(RedistDir)\TestFiles\%(RecursiveDir)" SourceFiles="@
(TestFilesSourceGroup)"/>
</Target>
```

Keep in mind that your deployment project file uses the MSBuild <Import> element to import the (large) BizTalkDeploymentFramework.targets file. That file contains all of the "code" that makes the deployment process work. If you are trying to understand exactly how the Deployment Framework for BizTalk implements a specific feature, or if you are looking for every available MSBuild property, that file should be your first stop. It may be found in <ProgramFiles>\MSBuild\DeploymentFrameworkForBizTalk\5.0.

Using Log4Net

To assist in debugging your BizTalk application during development and even into production, it is often critical to have a mechanism for saving messages to one or more easily-accessible locations. The open-source logging framework Log4Net is one good solution, and the Deployment Framework for BizTalk offers built-in support for it.

To enable Log4Net support, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the Includelog4net property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <Includelog4net>true</Includelog4net>
  ...
</PropertyGroup>
```

2. Add and configure a log4net configuration file
The Log4Net configuration is loaded from an XML configuration file that is deployed with your BizTalk application. In the **parent** folder of your deployment project, which is often the same folder as the solution file, add an XML file named $(ProjectName).log4net. $(ProjectName) is the value of the ProjectName MSBuild property in your .btdfproj file. Remember to add this file to your source control system.

Here's an example of a .log4net config file to write to a text file:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<log4net debug="true">
  <appender name="File"
     type="log4net.Appender.FileAppender, log4net, Version=1.2.9.0, Culture=Neutral,
PublicKeyToken=b32731d11ce58905">
    <layout type="log4net.Layout.SimpleLayout, log4net,
Version=1.2.9.0,Culture=Neutral,PublicKeyToken=b32731d11ce58905"/>
  </appender>
  <root>
    <level value="ERROR" />
    <appender-ref ref="File" />
  </root>
  <logger name="SampleApp">
    <level value="ALL" />
    <appender-ref ref="File"/>
  </logger>
</log4net>
```

3. Copy Log4Net.dll and Log4Net.Ext.Serializable.dll from the Deployment Framework for BizTalk install folder to a reference location in your BizTalk solution
Before you can use Log4Net in your application, locate the two DLL's in <BTDFInstallFolder>\Framework \DeployTools and copy them to a folder within your BizTalk solution folder hierarchy. It's a good idea to check the DLL's into your source control system and reference them in that location vs. the Deployment Framework's install folder.

4. Add references to Log4Net.dll and Log4Net.Ext.Serializable.dll to your project
In every BizTalk or .NET project where you want to write log entries, add a reference to the copies of Log4Net.dll and Log4Net.Ext.Serializable.dll stored within your solution folder structure.

5. Write log entries from an orchestration
The Deployment Framework for BizTalk includes two special serializable helper classes for Log4Net that can be used with orchestration variables. Create two variables in your orchestration: one of type *log4net.Ext.Serializable.SLog* and another of type *log4net.helpers.PropertiesCollectionEx*.

You can then initialize and write to the log within an Expression shape using code like the following example:

```
// Create the logger using the ProjectName property value from the .btdfproj
logger = log4net.Ext.Serializable.SLogManager.GetLogger("MyProjectName",
log4net.helpers.CallersTypeName.Name);
// Configure the logger by referencing the registry key written during the deployment
process
logger.RegistryConfigurator();
// Set some properties in the PropertiesCollectionEx object
logProps.Set("InstanceId", MyOrchestration(Microsoft.XLANGs.BaseTypes.InstanceId));
// Write a debug level entry to Log4Net including the properties object
logger.Debug(logProps, "Received top level request...");

// We can even pass the logger to C# code via a param typed as log4net.ILog
BizTalkSample.Components.MyClass.Execute(sampleRequest, logger);
```

6. Write log entries from a C# helper class
As the sample code above demonstrates, you can pass the SLog object to a C# method as type log4net.ILog. You're free to use the entire ILog interface to write log entries. You can also create and initialize the logger directly from C# code if you don't want or need to do it in an orchestration. That may include helper classes, pipeline components, map helper classes, etc.

When Log4Net deployment is enabled, the deployment process will automatically add log4net.dll and log4net.Ext.Serializable.dll to the GAC.

NOTE: On Windows x64, you might find issues with the registry configurator. The deployment script

typically runs as a 32-bit process, which means that it sees and works with the 32-bit "view" of the registry (SysWow6432). If your host(s) are 64-bit, they will see the 64-bit view of the registry. That might mean that they will not see the log4net registry entry because it only exists in the 32-bit registry.

On a 64-bit system the registry key will appear at HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\<YourProjectName>. For 64-bit hosts, the key needs to be duplicated at HKEY_LOCAL_MACHINE\SOFTWARE\<YourProjectName>. You can do this in a CustomPostDeploy target in your deployment project file.

Modifying BizTalk Server Configuration

The BizTalk orchestration engine includes a number of lesser-known options for debugging and developer assistance, all of which are disabled by default. They can be very useful during the application development and testing cycle. The Deployment Framework for BizTalk can automatically configure these features in the BizTalk service configuration file (BTSNTSvc.exe.config).

Extended Logging

To enable extended logging, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

Set the EnableBizTalkExtendedLogging property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <EnableBizTalkExtendedLogging>true</EnableBizTalkExtendedLogging>
  ...
</PropertyGroup>
```

Assembly Validation

To enable assembly validation, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

Set the EnableBizTalkAssemblyValidation property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <EnableBizTalkAssemblyValidation>true</EnableBizTalkAssemblyValidation>
  ...
</PropertyGroup>
```

Correlation Validation

To enable correlation validation, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

Set the EnableBizTalkCorrelationValidation property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <EnableBizTalkCorrelationValidation>true</EnableBizTalkCorrelationValidation>
  ...
</PropertyGroup>
```

Schema Validation

To enable schema validation, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

Set the EnableBizTalkSchemaValidation property to true

The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <EnableBizTalkSchemaValidation>true</EnableBizTalkSchemaValidation>
  ...
</PropertyGroup>
```


Deploying Multiple Application Versions Side-by-Side

Sometimes one must deploy multiple versions of the same BizTalk application to the same BizTalk server. To make this easier, the Deployment Framework for BizTalk can auto-append a version number to the BizTalk application name, SSO affiliate app name, port names, MSI product name and install path to avoid conflicts.

There are three important version numbers:

1. The assembly version numbers
2. The "project version" number
3. The "product version" number

BizTalk will look at the assembly version numbers to decide if an assembly has changed, but it will only look at Major.Minor and ignore the remainder of the version number [please verify]. The assembly version numbers must be deliberately changed in order to deploy SxS to the GAC since the strong names include the version numbers.

The deployment project file contains a property named "ProjectVersion". This is the version number that is used in the install directory structure, BizTalk application name, SSO app name, etc. This version should generally match the Major.Minor version of your assemblies.

The project file also contains a property named "ProductVersion". This version number represents the specific build version of the code. For instance, you may be working on BTApp 1.1 (ProjectVersion), but you are creating an installer for the specific build number BTApp 1.1.2358.2 (ProductVersion). ProductVersion is used in the generated MSI filename and displays in Add/Remove Programs in the "support information" dialog. The ProductVersion is a good choice to receive your automated build's version number.

Another way to look at these properties is that ProductVersion will change often and perhaps automatically (maybe on every rebuild), but ProjectVersion will change infrequently.

To enable side-by-side deployment features for your application, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

1. Set the EnableSideBySide property to true

The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <EnableSideBySide>true</EnableSideBySide>
  ...
</PropertyGroup>
```

2. Update the ProjectVersion property value
Set the value of the ProjectVersion property to the same Major.Minor version of your BizTalk assemblies.

```
<PropertyGroup>
  ...
  <ProjectVersion>1.0</ProjectVersion>
  ...
</PropertyGroup>
```

3. If you do not want the version number added to port names, then set the DisableAutomaticPortNameVersioning to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <DisableAutomaticPortNameVersioning>true</DisableAutomaticPortNameVersioning>
  ...
</PropertyGroup>
```


Controlling .NET App Domains

.NET application domains are a CLR (Common Language Runtime) feature that can be likened to lightweight processes within a managed process. Assemblies are loaded into and types are created in a particular AppDomain. Why is this important? Static objects are scoped by AppDomain. In addition, an AppDomain may be configured by a specific .NET configuration XML file.

Intuitively, one would guess that each BizTalk application is isolated from every other application. This, unfortunately, is not the case. By default, BizTalk loads and executes assemblies from many BizTalk applications in a single AppDomain. As a result, if your code uses a static/singleton object that is implemented in a common assembly, each running application will share the same static/singleton object state. This may lead to unanticipated data sharing or conflicts between BizTalk apps.

To avoid this problem, BizTalk can be configured to run a particular set of assemblies in a distinct AppDomain to provide true isolation from other BizTalk applications. Each AppDomain will have a distinct instance of shared static/singleton objects. Alternatively, the application might require configuration data from an XML configuration file, like that of Microsoft Enterprise Library. A separate AppDomain can be configured to use a specific application-defined XML configuration file path.

These settings are stored in the BizTalk BTSNTSvc.exe.config file. The Framework can automatically update the configuration file while deploying and undeploying the application.

To instruct BizTalk to run your application in an isolated AppDomain, edit your Deployment Framework for BizTalk project file (.btdfproj) as follows:

Set the UseIsolatedAppDomain property to true
The property may be included in any PropertyGroup, but is commonly placed in the first PropertyGroup in the project file.

```
<PropertyGroup>
  ...
  <UseIsolatedAppDomain>true</UseIsolatedAppDomain>
  ...
</PropertyGroup>
```

This feature utilizes a custom MSBuild task called UpdateBizTalkAppDomainConfig, which can also be used to specify a configuration file path and other options.

Technical Reference

This section dives deeper into the details of the tools, MSBuild properties and item groups that make up the Deployment Framework for BizTalk. You'll find a complete reference to every available BTDF property and item group.

In this section

- MSBuild Tasks
- MSBuild Properties
- MSBuild ItemGroups
- Deployment Tools

MSBuild Tasks

The Deployment Framework for BizTalk uses many custom MSBuild tasks (special .NET classes) to implement its extensive feature set. Many of the tasks are owned by the Deployment Framework itself, and their source code is included in the BTDF codebase.

Other tasks are imported from the SDC Tasks Library open-source project on CodePlex. These tasks are housed in Microsoft.Sdc.Tasks.dll.

The entire set of tasks (BTDF + SDC Tasks) is available for your use in custom MSBuild targets defined in your Deployment Framework for BizTalk project file (.btdfproj).

Please download the SDC Tasks Library help file (a .CHM) to see the complete set of tasks and related usage instructions.

MSBuild Properties

The Deployment Framework for BizTalk uses MSBuild properties to define and customize the deployment process.

Typical property usage

MSBuild properties are defined within a PropertyGroup element:

```
<PropertyGroup>
  <PropertyName>MyValue</PropertyName>
  <PropertyName2>MyValue2</PropertyName2>
  ...
</PropertyGroup>
```

PropertyGroup's may include an *optional* Condition attribute that causes the group of properties to be defined at runtime only when a certain boolean statement is true. A common use case is to define properties differently depending on whether the script is running on a development machine vs. a server. In this case, the Condition attribute should test against the Configuration property as follows: `Condition="'$(Configuration)' == 'Debug'"`. Possible values of Configuration are 'Debug', 'Release' (not a BizTalk server) and 'Server' (a BizTalk server).

Properties: Include/Exclude Artifacts

These properties control which BizTalk artifacts are included in or excluded from the deployment.

Name	Default	Description
IncludeBAM	False	Set to True to deploy one or more BAM models, specified in a BamDefinitions ItemGroup.
IncludeComponents	False	Set to True to deploy one or more assemblies containing .NET classes, specified in a Components ItemGroup.
IncludeCustomFunctoids	False	Set to True to deploy one or more assemblies containing custom functoids, specified in a CustomFunctoids ItemGroup.
IncludeDeploymentTest	False	Set to True to deploy an assembly containing NUnit tests, specified in a DeploymentTest ItemGroup.
IncludeEsbItineraries	False	Set to True to deploy one or more ESB Toolkit itinerary XML files, specified in an EsbItineraries ItemGroup.
IncludeLog4net	False	Set to True to deploy a Log4Net configuration file named $(ProjectName).log4net and located in the parent directory of the deployment project folder.
IncludeMessagingBindings	True	Set to True to deploy an XML bindings file, specified in a PortBindings property.
IncludeOrchestrations	True	Set to True to deploy one or more assemblies containing orchestrations, specified in an Orchestrations ItemGroup.
IncludePipelineComponents	False	Set to True to deploy one or more assemblies containing custom pipeline components, specified in a PipelineComponents ItemGroup.
IncludePipelines	False	Set to True to deploy one or more assemblies containing pipelines, specified in a Pipelines ItemGroup.
IncludeSchemas	True	Set to True to deploy one or more assemblies containing schemas, specified in a Schemas ItemGroup. If an assembly contains schemas **and** another type of BizTalk artifact, then treat it as a schemas assembly.
IncludeSSO	False	Set to True to deploy the environment-specific settings named in $(SettingsFilePath) [usually exported from the SettingsFileGenerator.xml Excel spreadsheet] into an SSO affiliate app for dynamic access at runtime.
IncludeTransforms	True	Set to True to deploy one or more assemblies containing transforms (maps), specified in a Transforms ItemGroup.
IncludeVirtualDirectories	False	Set to True to deploy one or more IIS virtual directories, specified in a VDirList ItemGroup.
IncludeVocabAndRules	False	Set to True to deploy one or more BRE policies and/or vocabularies, specified in RulePolicies and/or RuleVocabularies ItemGroup(s).

Properties: General Settings

These properties control various aspects of the deployment process.

Name	Default	Description
ApplyXmlEscape	False	Set to True to re-encode nested XML fragments within a bindings XML file prior to deployment. This requires a bindings file that has previously been processed by a decode operation in ElementTunnel.exe.
BizTalkAppDescription	$(ProjectName)	Specifies the description that appears in the application properties in BizTalk Server Administration.
DeleteFileAdapterPhysicalPath	Never	Indicates whether all physical file paths in the bindings file

Name	Default	Description
sOnUndeploy		that are associated with FILE adapters should be deleted when the application is undeployed. Acceptable values: Never, DeleteIfEmpty, DeleteRecursive.
DeployPDBsToGac	True	Set to True to deploy PDB files to the GAC next to the actual assemblies. Deploying PDBs to the GAC provides more meaningful stack traces. Enabling this option implies a host stop prior to deployment/undeployment since loading PDBs causes the BizTalk host process to hold on to assemblies even after orchestration unenlistment.
DisableAutomaticPortNameVersioning	False	Set to True to disable automatic insertion of the application's $(ProjectVersion) into all port names defined in the bindings file. Only applicable when EnableSideBySide is True.
EnableAllReceiveLocationsOnDeploy	True	Set to True to enable all receive locations during application startup.
EnableSideBySide	False	Set to True to enable side-by-side deployment assistance. This causes the application's $(ProjectVersion) to be appended to the BizTalk app name, SSO app name and port names.
EnableXmlPreprocess	True	Set to True to enable the use of XMLPreprocess to modify XML files on the fly during deployment, most commonly to merge environment settings into a bindings file.
ExplicitlyDeployRulePoliciesOnDeploy	False	Set to True to explicitly deploy BRE policies during deployment. Regardless of this setting, the policies will always be published, but are normally deployed automatically by BizTalk when the application starts. If you need policies to be deployed without having to start the BizTalk application, then set this to True.
IISMetabasePath	IIS://localhost/w3svc/1/Root	Specifies the IIS metabase path in WMI (IIS 5/6) format to be used to create IIS vdirs and app pools. The default path is that of the Default Web Site.
IncludeCompsAndVDirsAsResources	False	Set to True to include components and virtual directories as resources in the BizTalk application. Only useful if you plan to export the BizTalk app to an MSI from BizTalk Server Administration (not very useful since you're using the BTDF).
IncludeInstallUtilForComponents	False	Set to True to run the .NET Framework InstallUtil.exe tool on each assembly specified in the Components ItemGroup.
IncludeSettingsSpreadsheetInMsi	True	Set to True to include the environment settings spreadsheet (usually EnvironmentSettings\SettingsFileGenerator.xml) in generated MSI files. Since the spreadsheet may contain sensitive information like passwords and database connection strings, you may elect to leave it under the control of your operations or security staff.
ManageFileAdapterPhysicalPaths	True	Set to True to automatically create and assign file system permissions to any folders associated with FILE adapter ports in the bindings file.
ModifyNTFSPermissionsOnVDirPaths	True	Set to True to edit the NTFS permissions of IIS virtual directory paths to grant Read rights to the AspNet/AppPool identity account.
MsiName	$(ProjectName)-$(ProductVersion)	Specifies the name of the generated MSI file.

Name	Default	Description
RemoveRulePoliciesFromAppOnUndeploy	False	Set to True to remove rule policies from the BizTalk application (visible in Policies node in BizTalk Server Administration) when the application is undeployed. Regardless of this setting value, the policies will always be removed from the rule store. This option may be useful if you are undeploying rules separately from an application undeployment.
RequireXmlPreprocessDirectives	True	Set to True to require XMLPreprocess to look for *ifdef* preprocessor sections and replace macros (${id}) only within those sections, or whether (when set to False) it should simply replace all macros in the file.
SettingsFilesExportPath	$(MSBuildProjectDirectory)\EnvironmentSettings	Path that will store settings files exported from the environment settings spreadsheet. This only applies to developer deployments due to a hard-coded path in the batch files used on server deployments.
SettingsSpreadsheetPath	$(MSBuildProjectDirectory)\EnvironmentSettings\SettingsFileGenerator.xml	Path to the environment settings spreadsheet. This only applies to developer deployments due to a hard-coded path in the batch files used on server deployments.
SkipBamUndeploy	False (Local); True (Server)	Set to True to ignore BAM models during undeployment. Undeploying BAM models will cause data loss, so this defaults to True on server deployments.
SkipHostInstancesRestart	False	Set to True to skip restarting BizTalk host instances during deployment/undeployment.
SkipIISReset	False	Set to True to skip restarting IIS (or IIS AppPools on IIS 7.x) during deployment/undeployment. Recommended value is true if you are not using HTTP/WCF adapters, or if you are using them but have no custom pipeline components.
StartApplicationOnDeploy	True	Set to True to start the BizTalk application at the end of the deployment process.
StartReferencedApplicationsOnDeploy	True	Set to True to start all BizTalk applications referenced by the current BizTalk app.
UndeployIISArtifacts	False (Local); True (Server)	Set to True to undeploy IIS virtual directories during undeployment.
UsingMasterBindings	False	Set to True to enable use of a master (templated) bindings XML file that will be transformed at deploy time to a standard BizTalk bindings XML file.
XmlEscapeXPathsFile	$(DeployTools)\adapterXPaths.txt	Filename of and optional relative path to a text file containing XPath statements, one per line, to be processed by ElementTunnel.exe during bindings file preprocessing. Only applies when ApplyXmlEscape is True. The Deployment Framework for BizTalk includes a default file under the Framework\DeployTools folder.

Properties: BizTalk Host Control (Host Config Files)

These properties cause modifications to the BizTalk host config file(s) [BTSNTSvc.exe.config/

BTSNTSvc64.exe.config]. Please see the BizTalk product documentation for more information.

Name	Default	Description
EnableBizTalkAssemblyValidation	False	Set to True to cause the BizTalk engine to load all assemblies referenced by immediate dependent assemblies of an orchestration, and upon failure throw an AssemblyValidationException.
EnableBizTalkCorrelationValidation	False	Set to True to cause the BizTalk engine to verify that in a parallel convoy all messages that match one of the convoy receives have the same correlation property values, and upon failure throw a CorrelationValidationException.
EnableBizTalkExtendedLogging	False	Set to True to cause the BizTalk engine to display the promoted properties in the information for PublishMessageException for the message that failed to publish.
EnableBizTalkSchemaValidation	False	Set to True to cause the BizTalk engine to use XmlValidatingReader to validate every schema representing a message part type, and upon failure throw an XmlException.
UseIsolatedAppDomain	False	Set to True to cause BizTalk to load this application's assemblies into a separate .NET AppDomain.

Properties: MSI Generation

These properties control generation of your application's MSI.

Name	Default	Description
ProductName	[ProjectName] for BizTalk	Product name shown in the title of the installer
ProductId	GUID	GUID that changes for each new version of the application. This is used to determine whether the same application is already installed.
ProductVersion	1.0.0	Version number used only by the installer, displays in Add/ Remove Programs
Manufacturer	Deployment Framework User	Manufacturer name shown in the installer properties
PackageDescription	[ProjectName]	Description shown in the installer properties
PackageComments	[ProjectName]	Comments shown in the installer properties
ProductUpgradeCode	GUID	GUID that remains the same across versions of the application. This is used to determine when to upgrade an existing installed app. If you want to have two versions of the app installed side-by-side, then you need to change this GUID. Otherwise, keep it the same but update the ProductId and ProductVersion.
DefaultInstallDir	ProgramFilesFolder\[ProductName]\[ProjectVersion]	If present, this path will be written into a batch file generated next to the MSI. The batch file simply passes this path into the msiexec command-line to override the default property value.

MSBuild ItemGroups

The Deployment Framework for BizTalk uses MSBuild ItemGroup elements to identify and locate all of the

files that make up your BizTalk solution. Many of the common artifact types use the same structure, including those for orchestrations, schemas, transforms/maps, pipelines and more.

Element values within an ItemGroup are free to use any MSBuild property in the standard $(propertyName) format. The most commonly used property is $(Configuration), which contains the name of the active solution configuration (usually Debug or Release).

Typical ItemGroup structure

Most artifact types use a standard ItemGroup structure, shown below:

```
<ItemGroup>
  <ArtifactName Include="MyFilename.ext">
    <LocationPath>..\ProjectName\bin\$(Configuration)</LocationPath>
  </ArtifactName>
  <ArtifactName Include="MyFilename2.ext">
    <LocationPath>..\ProjectName2\bin\$(Configuration)</LocationPath>
  </ArtifactName>
  ...
</ItemGroup>
```

ArtifactName is a placeholder for the actual artifact name, such as Orchestrations, Schemas, Pipelines, etc. One *ArtifactName* element corresponds to exactly one physical file. The *Include* attribute specifies the file name without path information. The *LocationPath* element text specifies the path to the file named in the *Include* attribute, and is **always** resolved starting in the directory containing the .btdfproj file. Most of the time the *LocationPath* text should be a relative path within the BizTalk solution structure, but that is **not** a restriction.

While most ItemGroups reference files in the file system, some do not. Those include VDirList, BizTalkHosts, AppsToReference, PropsFromEnvSettings and IisAppPools.

Assemblies containing schemas

The Schemas ItemGroup is appropriate for any BizTalk assembly that contains schemas. The assembly may contain only schemas, or it may contain a mixture of schemas and other BizTalk artifacts. Schemas are always deployed early in the deployment cycle because so many other artifacts depend on them. This is the appropriate choice if your entire BizTalk application is contained in a single assembly.

```
<ItemGroup>
  <Schemas Include="MySchemas.dll">
    <LocationPath>..\$(ProjectName).Schemas\bin\$(Configuration)</LocationPath>
  </Schemas>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Assemblies containing pipelines

The Pipelines ItemGroup is appropriate for any BizTalk assembly that contains pipelines (not pipeline components). The assembly must contain only pipelines.

```
<ItemGroup>
  <Pipelines Include="MyPipelines.dll">
    <LocationPath>..\$(ProjectName).Pipelines\bin\$(Configuration)</LocationPath>
  </Pipelines>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Assemblies containing transforms (a.k.a. maps)

The Transforms ItemGroup is appropriate for any BizTalk assembly that contains transforms (a.k.a. maps). The assembly must contain only transforms.

```
<ItemGroup>
  <Transforms Include="MyTransforms.dll">
    <LocationPath>..\$(ProjectName).Transforms\bin\$(Configuration)</LocationPath>
  </Transforms>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Assemblies containing orchestrations

The Orchestrations ItemGroup is appropriate for any BizTalk assembly that contains orchestrations. The assembly must contain only orchestrations.

```
<ItemGroup>
  <Orchestrations Include="MyOrchestrations.dll">
    <LocationPath>..\$(ProjectName).Orchestrations\bin\$(Configuration)</LocationPath>
  </Orchestrations>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Assemblies containing custom functoids

The CustomFunctoids ItemGroup is appropriate for any BizTalk assembly that contains custom functoids. The assembly must contain only functoids.

```
<ItemGroup>
  <CustomFunctoids Include="MyFunctoids.dll">
    <LocationPath>..\$(ProjectName).CustomFunctoids\bin\$(Configuration)</LocationPath>
  </CustomFunctoids>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Assemblies containing custom .NET components

The Components ItemGroup is appropriate for any assembly that contains custom .NET components (classes). The assembly must contain only .NET artifacts.

```
<ItemGroup>
  <Components Include="MyComponents.dll">
    <LocationPath>..\$(ProjectName).Components\bin\$(Configuration)</LocationPath>
  </Components>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Assemblies containing custom pipeline components

The PipelineComponents ItemGroup is appropriate for any assembly that contains custom pipeline components. The assembly must contain only pipeline components.

```
<ItemGroup>
  <PipelineComponents Include="MyPipelineComponents.dll">
    <LocationPath>..\$(ProjectName).PipelineComponents\bin\$(Configuration)</
LocationPath>
  </PipelineComponents>
</ItemGroup>
```

Include attribute: DLL file name

Repeating: 1 or more elements

BAM model Excel spreadsheets

The BamDefinitions ItemGroup is appropriate for any Excel spreadsheet that defines a BAM model.

```
<ItemGroup>
  <BamDefinitions Include="MyBAMModel.xls">
    <LocationPath>..\BRE</LocationPath>
  </BamDefinitions>
</ItemGroup>
```

Include attribute: Excel file name (xls/xlsx)
Repeating: 1 or more elements

BAM tracking profile definition files

The BamTrackingProfiles ItemGroup is appropriate for any BAM tracking profile definition file.

```
<ItemGroup>
  <BamTrackingProfiles Include="MyTrackingProfile.btt">
    <LocationPath>..\BRE</LocationPath>
  </BamTrackingProfiles>
</ItemGroup>
```

Include attribute: Tracking profile file name (btt)
Repeating: 1 or more elements

Assembly containing NUnit test(s)

The DeploymentTest ItemGroup is appropriate for any .NET assembly that contains one or more NUnit test methods.

```
<ItemGroup>
  <DeploymentTest Include="MyDeploymentTests.dll">
    <LocationPath>..\$(ProjectName).Tests\bin\$(Configuration)</LocationPath>
  </DeploymentTest>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: Exactly 1 element

XML files requiring preprocessing

The FilesToXmlPreprocess ItemGroup is appropriate for any XML files that require preprocessing with XMLPreprocess.exe.

```
<ItemGroup>
  <FilesToXmlPreprocess Include="MyFileForPreprocessing.xml">
    <LocationPath>..\SupportingFiles</LocationPath>
  </FilesToXmlPreprocess>
</ItemGroup>
```

Include attribute: XML file name
Repeating: 1 or more elements

Assemblies requiring GAC installation

The ExternalAssemblies ItemGroup is appropriate for any .NET assemblies that need to be installed in the GAC.

```
<ItemGroup>
  <ExternalAssemblies Include="MyDLLForTheGAC.dll">
    <LocationPath>..\ExternalAssemblies</LocationPath>
  </ExternalAssemblies>
</ItemGroup>
```

Include attribute: DLL file name
Repeating: 1 or more elements

Miscellaneous files required on the BizTalk server

The AdditionalFiles ItemGroup is appropriate for any files that do not fall into a more specific ItemGroup that need to be packaged in the MSI and installed on the server along with your application.

```
<ItemGroup>
  <AdditionalFiles Include="MyAdditionalFile.reg">
    <LocationPath>..\SupportingFiles</LocationPath>
  </AdditionalFiles>
</ItemGroup>
```

Include attribute: File name (any extension)
Repeating: 1 or more elements

BRE rule vocabulary XML files

The RuleVocabularies ItemGroup is appropriate for any BRE rule vocabulary XML files.

```
<ItemGroup>
  <RuleVocabularies Include="MyBREVocabulary.xml">
    <LocationPath>..\BRE\Vocabularies</LocationPath>
  </BamTrackingProfiles>
</ItemGroup>
```

Include attribute: Vocabulary file name (xml)
Repeating: 1 or more elements

BRE rule policy XML files

The RulePolicies ItemGroup is appropriate for any BRE rule policy XML files.

```
<ItemGroup>
  <RulePolicies Include="MyBREPolicy.xml">
    <LocationPath>..\BRE\Policies</LocationPath>
  </RulePolicies>
</ItemGroup>
```

Include attribute: Policy file name (xml)
Repeating: 1 or more elements

ESB Toolkit itinerary XML files

The EsbItineraries ItemGroup is appropriate for any ESB Toolkit itinerary XML files (exported from the Itinerary Designer to XML).

```
<ItemGroup>
  <EsbItineraries Include="MyItinerary.xml">
    <LocationPath>..\$(ProjectName).Itineraries\bin\$(Configuration)</LocationPath>
  </EsbItineraries>
</ItemGroup>
```

Include attribute: Exported itinerary XML file name (xml)
Repeating: 1 or more elements

IIS Virtual Directory definitions

The VDirList ItemGroup defines one or more IIS virtual directories. It's unusual in that it doesn't point to a particular file, but instead defines metadata and folder paths that make up a virtual directory.

```
<ItemGroup>
  <VDirList Include="*">
    <Vdir>MyWebService</Vdir>
```

```
    <Physdir>..\MyWebService</Physdir>
    <AppPool>MyAppPool</AppPool>
    <AppPoolNetVersion>v4.0</AppPoolNetVersion>
  </VDirList>
</ItemGroup>
```

Include attribute: Always * (a single asterisk)
Repeating: 1 or more elements

BizTalk Host names to restart

The BizTalkHosts ItemGroup is appropriate for the names of one or more BizTalk hosts that should be restarted during the deployment and undeployment processes.

```
<ItemGroup>
  <BizTalkHosts Include="MyHostName" />
</ItemGroup>
```

Include attribute: BizTalk host name
Repeating: 1 or more elements

BizTalk application names to reference

The AppsToReference ItemGroup is appropriate for the names of one or more BizTalk applications that should be referenced by the current application.

```
<ItemGroup>
  <AppsToReference Include="AnotherBizTalkAppName" />
</ItemGroup>
```

Include attribute: Name of BizTalk application to reference
Repeating: 1 or more elements

BizTalk Host names

The PropsFromEnvSettings ItemGroup is appropriate for the names of one or more settings listed in the SettingsFileGenerator.xml Excel spreadsheet that should be converted to MSBuild properties. This allows you to reference any of the setting values in the spreadsheet within the deployment project file.

```
<ItemGroup>
  <PropsFromEnvSettings Include="MySettingName" />
</ItemGroup>
```

Include attribute: Name of a setting in the SettingsFileGenerator.xml Excel spreadsheet
Repeating: 1 or more elements

IIS Application Pool names to restart

The IisAppPools ItemGroup is appropriate for the names of one or more IIS application pools that should be restarted during the deployment and undeployment processes. This applies to IIS 6.0 and newer.

```
<ItemGroup>
  <IisAppPools Include="MyAppPoolName" />
</ItemGroup>
```

Include attribute: IIS application pool name
Repeating: 1 or more elements

Deployment Tools

The Deployment Framework for BizTalk builds upon a number of external programs: some from BizTalk Server itself, some housed within the Deployment Framework project and some from other open-source

projects.

In this section

- BAM Definition XML Exporter (ExportBamDefinitionXml.exe)
- Environment Settings Exporter (EnvironmentSettingsExporter.exe)
- Environment Variables Wizard GUI (SetEnvUI.exe)
- Nested XML Encoder/Decoder (ElementTunnel.exe)
- BRE Deployment Tool (DeployBTRules.exe)
- SSO Settings Editor GUI (SSOSettingsEditor.exe)
- SSO Settings Importer (SSOSettingsFileImport.exe)
- XML File Preprocessor (XmlPreprocess.exe)

BAM Definition XML Exporter (ExportBamDefinitionXml.exe)

The **BAM Definition XML Exporter (ExportBamDefinitionXml.exe)** is a tool that exports the XML definition of a BAM model from inside an Excel workbook to an XML file. The XML file, which is always present inside a BAM workbook, may be used as input to the BizTalk BAM Utility (bm.exe) in order to deploy or manage a BAM model without having Microsoft Excel installed -- typical for a BizTalk server.

```
ExportBamDefinitionXml <ExcelFile> <XMLFile> [UseLegacyExport]
```

Arguments

Argument	Description
ExcelFile	The path to a BAM Excel workbook file (.xls)
XMLFile	The path to an XML file that will contain the exported XML data
UseLegacyExport	An optional (and rarely necessary) option that tells the tool to use Excel Automation, which requires Excel to be installed. This option should be used only as a last resort if exports are not working for a problematic workbook. To activate this option, pass the value "True" (without the quotes).

Remarks

Working with Excel 2003 files requires the Microsoft JET 4.0 OLE DB driver, which is typically present on machines that have the .NET Framework 2.0 or newer installed.

Working with Excel 2007 or newer files requires the Microsoft Office 2007 Data Connectivity Components or Access Database Engine 2010 Redistributable, both of which can be obtained from Microsoft.com's Download Center. If you're working on a machine that has Excel 2007 or 2010 installed, the components are already present (usually the case for a development machine).

It is not currently possible to install both the 32-bit and 64-bit versions of certain data access components, including the Access Database Engine 2010 Redistributable. As a result, if you have the 64-bit data access components installed (or 64-bit Office), then this tool must also run as 64-bit, and likewise for 32-bit. The Deployment Framework for BizTalk includes only the 32-bit version of the tool, so please install the 32-bit version of data access components.

See Also

This tool is part of the *BizTalk BAM Typed API Generator & BAM XML Exporter* project on CodePlex. The project website contains complete documentation and much more detail. The tool is Copyright © 2007-08 Darren Jefford and © 2008-11 Thomas F. Abraham. All Rights Reserved. Subject to the Microsoft Public License (Ms-PL).

Environment Settings Exporter (EnvironmentSettingsExporter.exe)

The **Environment Settings Exporter (EnvironmentSettingsExporter.exe)** is a tool that reads a specially formatted Excel spreadsheet in either Excel Binary or Excel SpreadsheetML XML format and exports the configuration settings in the workbook to multiple XML files, one per defined environment (development, test, production, etc.).

The exporter can export to three different XML formats, but the Deployment Framework for BizTalk uses only the first:
1. XmlPreprocess format
2. .NET <appSettings> format
3. WiX include format with a <CustomTable> definition

```
EnvironmentSettingsExporter <File.xls/x or File.xml> <OutputPath> [/
F:<XmlPreprocess/AppSettings/WixCustomTable>]
```

Arguments

Argument	Description
File.xls/x or File.xml	The path to a configuration settings spreadsheet, either in binary (.xls or .xlsx) or Excel SpreadsheetML XML format. Surround the path in double-quotes if it contains spaces.
OutputPath	The path to a folder that will contain the exported XML files, one for each defined environment and named per the filenames defined in the spreadsheet.
F (Format)	An optional parameter that specifies the output XML format: XmlPreprocess, AppSettings or WixCustomTable. The default, and the only option used by the Deployment Framework for BizTalk, is XmlPreprocess.

Remarks

Newly created Deployment Framework for BizTalk projects contain a default settings spreadsheet in the file EnvironmentSettings\SettingsFileGenerator.xml.

See Also

This tool is part of the *Environment Settings Manager* project on CodePlex. The project website contains complete documentation and much more detail. The tool is Copyright © 2007-10 Thomas F. Abraham. All Rights Reserved. Subject to the Microsoft Public License (Ms-PL).

Environment Variables Wizard GUI (SetEnvUI.exe)

The **Environment Variables Wizard GUI (SetEnvUI.exe)** is a tool that sets one or more environment variables via a wizard GUI created from an XML definition file. It is used by the Deployment Framework for BizTalk to present the deployment wizard GUI during server (MSI) deployment and un-deployment.

```
SetEnvUI.exe [@argFile] <configFile> <postProcess> <postProcessArgs> [/Help|?|
h] [/Version|v]
```

Arguments

Argument	Description
ConfigFile	Path to an XML file that defines the wizard pages and environment variable names. Must validate against SetEnvUIConfig.xsd. Corresponds to InstallWizard.xml and UnInstallWizard.xml in a Deployment Framework for BizTalk project.
PostProcess	Path to a program to be executed after the user clicks Finish in the wizard

PostProcessArgs	Command line arguments for the program specified in PostProcess
Help OR ? or h	Displays command-line help text.
Version OR v	Displays program version information

XML Schema

The following XSD schema may be used to validate XML files intended for use with the Environment Variables Wizard GUI. The schema may also be found in the file SetEnvUIConfig.xsd in <BTDFInstall>\Framework\DeployTools.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- V1.0a -->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SetEnvUIConfig" nillable="true" type="SetEnvUIConfig" />
  <xs:complexType name="SetEnvUIConfig">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="DialogCaption" type="xs:string" />
      <xs:element minOccurs="1" maxOccurs="unbounded" name="SetEnvUIConfigItem"
type="SetEnvUIConfigItem" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="SetEnvUIConfigItem">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="PromptText" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" name="Caption" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" name="PromptValue" type="xs:string" />
      <xs:element minOccurs="0" maxOccurs="1" default="true" name="PersistValue"
type="xs:boolean" />
      <xs:element minOccurs="1" maxOccurs="1" name="ValueType" type="SetEnvUIValueType" /
>
      <xs:element minOccurs="1" maxOccurs="1" name="EnvironmentVarName"
type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="SetEnvUIValueType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Text" />
      <xs:enumeration value="Password" />
      <xs:enumeration value="FileSelect" />
      <xs:enumeration value="Checkbox" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

This is a common example of a default server deployment definition file (InstallWizard.xml). This instance defines two wizard steps (two SetEnvUIConfigItem elements), each of which sets the value of a single environment variable (named in EnvironmentVarName). The wizard caption/title is defined in DialogCaption.

```
<?xml version="1.0" encoding="utf-8" ?>
<SetEnvUIConfig xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <DialogCaption>Deployment Framework for BizTalk Sample</DialogCaption>
  <SetEnvUIConfigItem>
    <PromptText>Select the XML file that contains configuration information:</PromptText>
    <PromptValue></PromptValue>
    <ValueType>FileSelect</ValueType>
    <EnvironmentVarName>ENV_SETTINGS</EnvironmentVarName>
```

```
  </SetEnvUIConfigItem>
  <SetEnvUIConfigItem>
      <PromptText>Is this the LAST server in the BizTalk Group you are deploying to?</
PromptText>
      <Caption>This is the LAST server in the BizTalk Group</Caption>
      <PromptValue>true</PromptValue>
      <ValueType>Checkbox</ValueType>
      <EnvironmentVarName>BT_DEPLOY_MGMT_DB</EnvironmentVarName>
   </SetEnvUIConfigItem>
</SetEnvUIConfig>
```


Nested XML Encoder/Decoder (ElementTunnel.exe)

The **Nested XML Encoder/Decoder for BizTalk XML Binding Files (ElementTunnel.exe)** is a tool that facilitates working with a (more) human-friendly BizTalk XML binding file. Many elements within a BizTalk binding file, such as adapter configurations, contain nested XML fragments that are encoded (& to ", > to >, etc.). The encoding makes it difficult to read and edit the nested XML, so the tool creates a copy of the bindings file with decoded XML fragments, then re-encodes them so that BizTalk can once again understand them.

```
ElementTunnel.exe  [@argfile] /I:<inputFile> /X:<fileWithXPaths> /
O:<outputFile> [/Encode[+|-]] [/Decode[+|-]] [/Help|?|h] [/Version|v]
```

Arguments

Argument	Description
I (Input File)	Path to input file, typically (when decoding) a BizTalk XML bindings file freshly exported from BizTalk Server Administration tool
X (XPaths File)	Path to text file containing a list of XPath queries that indicate the locations requiring encoding or decoding. In a Deployment Framework for BizTalk project, this normally points to <BTDFInstallPath>\Framework\DeployTools\AdapterXPaths.txt.
O (Output File)	Path to output file, typically (when decoding) a Deployment Framework for BizTalk project's PortBindingsMaster.xml file.
Encode	Direct the tool to encode the XML fragments indicated by each input XPath query. Cannot be specified with Decode.
Decode	Direct the tool to decode the XML fragments indicated by each input XPath query. Cannot be specified with Encode.
Help OR ? or h	Displays command-line help text.
Version OR v	Displays program version information

BRE Deployment Tool (DeployBTRules.exe)

The **BRE Policy/Vocabulary Deployment Tool (DeployBTRules.exe)** is a tool that deploys, publishes or un-deploys BizTalk Rules Engine policy and vocabulary XML files. This is typically done automatically by setting the MSBuild property *IncludeVocabAndRules* to True in a Deployment Framework for BizTalk project file (.btdfproj). The BRE XML files must be exported using the BizTalk BRE Deployment Wizard.

```
DeployBTRules.exe [@argfile] [/RuleSetFile:<ruleSetFile>] [/
RuleSetName:<ruleSetName>] [/VocabularyName:<vocabularyName>] [/Undeploy[+|-]]
[/Unpublish[+|-]] [/RuleSetVersion:<ruleSetVersion>] [/Help|?|h] [/Version|v]
```

Arguments

Argument	Description
RuleSetFile	Path to an XML BRE rule policy or vocabulary file
RuleSetName	When RuleSetFile points to a policy XML file, the particular rule set name on which to operate.
VocabularyName	When RuleSetFile points to a vocabulary XML file, the particular vocabulary name on which to operate.
Undeploy	Undeploy the specified policy or vocabulary. When not specified, default is to deploy.
Unpublish	Unpublish the specified policy or vocabulary. When not specified, default is to publish (implies undeploy)
RuleSetVersion	Rule set version on which to operate
Help OR ? or h	Displays command-line help text.
Version OR v	Displays program version information

SSO Settings Editor GUI (SSOSettingsEditor.exe)

The **SSO Settings Editor GUI (SSOSettingsEditor.exe)** is a simple viewer and editor for custom application settings stored within the SSO system. The main prerequisite for this tool is that an XML settings file has been loaded into SSO by the SSO Settings Importer (SSOSettingsFileImport.exe). This is typically done automatically by setting the MSBuild property *IncludeSSO* to True in a Deployment Framework for BizTalk project file (.btdfproj).

```
SSOSettingsEditor [SSOAppName]
```

Arguments

Argument	Description
SSOAppName	The name of an SSO application created by the SSO Settings Importer. This is generally the name specified in a Deployment Framework project's *ProjectName* element.

Remarks

The associated SSO Settings File Reader API (SSOSettingsFileReader.dll) caches values in memory, so there may be a delay of up to 60 seconds after a save operation before the updated settings appear through the API.

SSO Settings Importer (SSOSettingsFileImport.exe)

The **SSO Settings Importer (SSOSettingsFileImport.exe)** is a tool that loads an XML settings file into an SSO affiliate application. This is typically done automatically by setting the MSBuild property *IncludeSSO* to True in a Deployment Framework for BizTalk project file (.btdfproj).

```
SSOSettingsFileImport.exe [@argfile] <AffiliateAppName> [/
SettingsFile:<value>] [/List[+|-]] [/DeleteApp[+|-]] [/UserGroupName:<value>]
[/AdminGroupName:<value>] [/PropToModify:<value>] [/PropValue:<value>] [/
Help|?|h] [/Version|v]
```

Arguments

Argument	Description
AffiliateAppNam	The name to be given to the SSO application. This is generally the name specified in a

e	Deployment Framework project's *ProjectName* element.
SettingsFile	Path to the XML file (usually created by EnvironmentSettingsExporter.exe) containing settings to load into SSO
List	Print out all name/value pairs currently stored in the SSO application
DeleteApp	Delete the SSO application
UserGroupNam e	SSO user group name (name or DOMAIN\name) used when SSO was configured
AdminGroupNa me	SSO admin group name (name or DOMAIN\name) used when SSO was configured
PropToModify	Name of an existing name/value pair to modify
PropValue	New value for an existing name/value pair to modify (with PropToModify)
Help OR ? or h	Displays command-line help text.
Version OR v	Displays program version information

Remarks

The associated SSO Settings File Reader API (SSOSettingsFileReader.dll) caches values in memory, so there may be a delay of up to 60 seconds after a modify operation before the updated settings appear through the API.

Credits

In this section

- Lead Authors
- Contributors and Acknowledgements

Lead Authors

Thomas F. Abraham
Thomas Abraham helps firms address their most challenging business software issues. Thomas has an extensive background in software development, architecture, configuration management and systems engineering, helping to build high-performance, mission-critical applications for companies including Nasdaq, Best Buy and Wells Fargo. Over 15+ years, Thomas has worked with technologies ranging from C/C++ to BizTalk Server to Exchange and .NET, was the lead author of the book "Visual Basic .NET Solutions Toolkit" from Wrox Press and a presenter at the 2006 SOA & Business Process Conference in Redmond, WA. Thomas holds a number of Microsoft certifications, including MCPD, MCSD, MCT and TS for both BizTalk 2004 and 2006. Thomas maintains a blog at http://www.tfabraham.com.

Scott Colestock
Scott Colestock, the owner of Trace Ventures, LLC in the Twin Cities of Minnesota, created the Deployment Framework for BizTalk in 2004 and developed it through mid-2008. He has delivered solutions in the SOA/BPM space for multiple clients in the area, as well as significant work in the Windows Mobile arena. Scott is recognized for his work with Team Foundation Server, helping several clients successfully deploy and adopt. Scott is a repeat BizTalk Server MVP, a certified ScrumMaster, and speaks frequently at user groups and industry conferences. Scott blogs at http://www.traceofthought.net.

Contributors and Acknowledgements

The Deployment Framework project builds upon the valuable contributions of countless contributors to open-source projects:

NUnit

XMLPreprocess courtesy of Loren Halvorsen

WiX

Log4Net

Environment Settings Manager courtesy of Thomas F. Abraham

ExportBamDefinitionXml courtesy of Thomas F. Abraham and Darren Jefford

SSO-based Configuration inspired by Jon Flanders

Thanks also to:

- Tim Rayburn for contributing a bug fix and unit tests for the ElementTunnel tool
- Giulio Van for contributing ideas and bug fixes.
- The hundreds of active users across the world who have promoted the Deployment Framework, reported

bugs, offered suggestions and taken the time to help other users!