Deployment Framework for BizTalk Server 2006 V4.0
User Guide for Core Tools & Sample Application

by Scott Colestock (www.traceofthought.net)
with contributions by Thomas F. Abraham

Document Version 1.0

Visit the project home page at http://www.codeplex.com/biztalkdeployment

This project is not affiliated with Microsoft Corporation.
The BizTalk Server name, logo and trademark are owned exclusively by Microsoft Corporation.

Table of Contents

1. Introduction

One of the more complicated aspects for a developer using BizTalk 2004/2006 is the large number of steps required during the edit/run/debug cycle. Since your BizTalk artifacts can't be run "in place" (that is, where they were compiled) but must instead be deployed to a local (or remote) BizTalk server, life is a bit more complicated.

If you have done much BizTalk 2004 development, you know the routine quite well at this point. If you have orchestrations, schemas, transforms, pipelines, and (say) C# components partitioned into separate assemblies - and you have a number of orchestrations with "Call/Start Orchestration" shape dependencies that introduce specific start/stop-ordering - you can spend a *lot* of time doing the whole stop/unenlist/undeploy/redeploy/enlist/start/bounce BizTalk routine. BizTalk 2006 did quite a bit in this area, but still leaves room for improvement.

It can be quite helpful to use a tool like NAnt to coordinate the "update BizTalk Server" process that must occur after each build. (NAnt is a large topic unto itself - suffice to say it is an XML-driven software build system.) As long as your NAnt build file (and BizTalk binding files) are kept up to date, the whole process for a developer can be reduced to:

1. Compile your solution (which might have multiple orchestrations, schemas, transforms, pipelines, and external components in separate assemblies)
2. Choose "Deploy to BizTalk" from your Tools menu.
3. Wait 90 seconds or so, enjoying the feeling that you have a consistent & reliable deployment mechanism.

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

The Deployment Framework includes the following features:

- Automation of the entire BizTalk application deployment and updating processes
- Full and Fast application update modes to reduce development cycle time
- Detailed logging for informational and troubleshooting purposes
- Integration with Visual Studio 2005 menu and output window
- 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
- Templated bindings file that automatically targets multiple runtime environments
- Enables use of un-encoded XML for adapter and port configurations in binding files
- Configuration settings infrastructure including user-friendly settings management spreadsheet and .NET object for settings access at runtime
- Single deployment script serves both development workstations and production servers
- Automated packaging of entire application into standard Windows Installer MSI file
- Support for side-by-side deployment of multiple versions of a single application
- Easily customizable installation wizard for server deployments
- Deployment verification via NUnit unit test tool
- Integrated deployment of Log4Net for runtime event logging
- Automated configuration of BizTalk runtime settings including debugging features and .NET assembly-to-AppDomain mappings
- Automated restart of one or more BizTalk host instances and IIS services
- Automatic addition of cross-application references
- Automatic deployment of debugging PDB files to the Global Assembly Cache (GAC)
- Support for Windows x64
- Infinite extensibility through open-source license

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

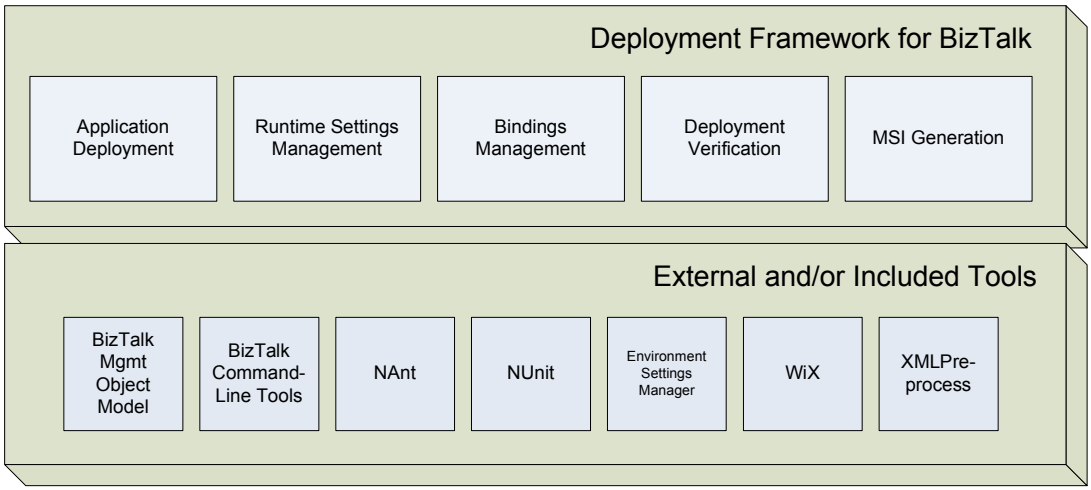


Figure 1 - Components of the Deployment Framework

A short "getting started" for NAnt & directions for using the Deployment Framework as a VS 2005 external tool (the “Tools” menu) can be found in Appendix A.

2. Overview of the Deployment Sample

The download contains both:

1. The core, generic functionality needed to deploy a wide range of BizTalk solutions (the "deployment framework")
2. A reasonably complex sample application that demonstrates most of the provided functionality.

The NAnt scripts that are used are partitioned into files that a) you must change for your project, and b) files that are part of the core (which, as bugs are discovered and improvements are made, can be replaced independently.)

The sample BizTalk solution contains the following projects:

1. BizTalkSample.Components (a C# project, with a class called from orchestration)
2. BizTalkSample.Orchestrations (that contains three orchestrations, one which calls the other via a Call Orchestration shape. One orchestration uses http post, another uses the file transport.)
3. BizTalkSample.Pipelines (a send and receive pipeline, which are not needed but included to illustrate the deployment aspects)
4. BizTalkSample.Schemas (which includes two schemas)
5. BizTalkSample.Transforms (which contains a single transform used by one of the orchestrations)

The sample solution also contains BizTalkSample.sln.deploy.build, which represents a typical example of how to leverage the core functionality within BizTalkDeploymentInclude.NAnt (where the generic portions are housed.)

BizTalkDeploymentInclude.NAnt has a deployment target which captures the following dependency tree (only partially blown out here, for brevity - but this no doubt seems familiar to you if you've been using Biztalk 2004/2006 for awhile…)

Deploy
- Deploy Schemas
  - Undeploy Schemas
    - Undeploy Orchestrations
      - Unenlist Orchestrations
        - Stop Orchestration
  - Undeploy Transforms
    - Undeploy Orchestrations
- Deploy Components
- Deploy Pipelines
  - Undeploy Pipelines
    - Undeploy Orchestrations
- Deploy Transforms
  - Undeploy Transforms
    - Undeploy Orchestrations
- Deploy Orchestrations

Bounce (restart) BizTalk

Perhaps more illustrative is the output of the deployment itself – which you can see in the output window of Visual Studio when deploying the sample. Note that the NAnt script uses a few custom NAnt tasks (that delegate to the BizTalk Explorer Object Model) as well as other scripts within the "DeployTools" directory (as well as the BTSTask or BTSDeploy command line utilities.)

The sample solution also contains several additional components that are used to manage deployments to a BizTalk server, which will be discussed later on.

To try the developer experience with the sample, follow Appendix A to set up NAnt. Then, extract the download and select "Deploy BizTalk Solution" from your Visual Studio tools menu.

3. Getting Started with Developer Deployments

As just stated, the whole of the BizTalk deployment functionality presented here covers both developer workstation deployments as well as server deployments. But, to get started with just the developer workstation scenario (a big productivity boost all by itself) for your own BizTalk project, you will want to do the following. Note that MyProject always refers to your *solution name.*

Strongly consider partitioning your BizTalk projects as shown above in the sample, i.e. one type of BizTalk component per assembly:

a) MyProject.Orchestrations
b) MyProject.Schemas
c) MyProject.Transforms
d) MyProject.Pipelines
e) MyProject.PipelineComponents
f) MyProject.CustomFunctoids
g) MyProject.Components (for ordinary .NET components)

Related to (1), strongly consider having your assembly names, project names, and directory names all be identical (with the solution file sitting above them all.) The provided NAnt scripts don't require this, but they make it substantially easier to work this way.

BizTalk uses Visual Studio build configurations called "Development" and "Deployment". To make the location of binaries consistent across BizTalk and non-BizTalk projects, change the output location for the "Development" configurations to "bin\debug" and output location of "Deployment" configurations to "bin\release".

Alternative 1: After you have created all of your assemblies, 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 up the solution again and go to Build-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.

From the sample solution, copy the DeployTools into your project directory. Create an empty DeployResults directory as well.

Copy BizTalkDeploymentInclude.NAnt to your project directory, as well BizTalkSample.sln.deploy.build (or deploy.start, which is more bare-bones.) Rename the latter to MyProject.sln.deploy.build – this is what will allow the external tool in Visual Studio to work correctly, as described in Appendix A. (As an alternative to steps 4/5, you can unzip the “Deployment Framework (Core)” zip into your project directory. Rename the WiX directory to correspond to your project.)

Manually deploy your solution using the usual techniques, and use the BizTalk Deployment Wizard to export your binding file(s). Name the binding file MyProject.PortBindings.xml, or define a NAnt property named portBindings if you wish to have more than one binding file:

```
<property name="portBindings" value="file1.xml,file2.xml" />
```

Open up MyProject.sln.deploy.build, and change the “includeXYZ” properties that are near the top to reflect what BizTalk elements are actually present in your solution. The core script uses these properties to determine what functionality to include while running.

Add to the customDeployTarget and/or customUndeployTarget as needed (i.e. if you have deployment or undeployment work that you need to do that isn’t taken care of by the core scripts.)

If you have virtual directories, populate MyProject.VDirList.txt with "virtual directory, physical directory, app pool name" entries (see BizTalkSample.VDirList.txt.) Populate app pool names even on non-IIS6 machines. Note that the physical directories are interpreted *relative to the directory where the nant script is located.*

There are many other configuration options possible that will be discussed below, but this should get you started.

4. Server Deployments

As you study the build script, you will notice that it tackles such tasks as deploying virtual directories and applying permissions to them, managing IIS6 application pools, patching binding files to match “local conditions”, deploying BizTalk rules, and many other items.

The initial goal here is to ensure that in a team development scenario, the amount of “out of band” setup required for any given developer to establish their BizTalk project environment

is minimal – and that changing topology for the project is communicated efficiently (i.e. through the build file.)

After a development team has spent weeks or months with this build script, refining it to represent their exact situation and deployment needs, a question might arise: *Why not take this well-tested script and use it for production (or just non-development-machine) deployments as well?*

To do this, it is helpful to agree that a two-phase deployment of BizTalk-related projects is both acceptable and useful. The first phase is an MSI-based installation that simply installs all of the BizTalk-related assets in a specific directory, but doesn't deploy them to the BizTalk server. The second phase occurs when the user goes to the Start-Programs-MyProject menu, and chooses "Deploy MyProject". The user also has the option to un-deploy (leaving the assets still on the file system until the MSI is uninstalled) or redeploy (to support the case where a single file is "patched", etc.)

Figure 2 - Start menu after MSI portion of installation

The reason that two phases are useful is that if the deployment to BizTalk fails, we would like a complete log of the results (which in the deployment framework occurs in the DeployResults directory) for analysis/diagnostics, and we would rather not have the MSI simply roll back. The same holds true of un-deployment.

The BizTalkDeploymentInclude.NAnt file already accommodates dual-purposing the NAnt file you have been maintaining for developer workstations -- a serverDeploy target is provided which makes the build file work with "co-located" binaries, in addition to the standard development tree where binaries are located in separate directories and underneath bin\debug or bin\release directories.

When executing the serverDeploy target, additional functionality is included such as setting application pool identity for IIS6 (or COM+ application identity for Win2k/Winxp) for virtual directories.

The deployment framework leverages WiX and custom scripts to create the MSI. The process is quite useful because knowledge of what needs to be included in the MSI is automatically gathered from your deployment NAnt file. This approach will be discussed in a bit more detail later.

Deployments on a server can often require additional information beyond what is provided in the NAnt script. This information can be divided into two broad categories:

- Information that is specific to the environment you are installing in (i.e. development, QA, integration/staging, or production.) We call this "Environment-relative configuration."
- Information that has to be gathered from the person doing the installation (i.e. account names, passwords, etc.) We call this "Install-time configuration."

5. Install-time Configuration for Server Deployments

Install-time configuration could be gathered in any number of ways. However, to accommodate the two-phase approach to installation that is discussed above (and to keep things somewhat simple) a simple utility is provided here (and is used by the sample solution.) The utility (SetEnvUI.exe) does three things:

1. Consumes an Xml file that describes what information needs to be gathered, and dynamically creates a Wizard UI based on that.
2. Sets process-level environment variables corresponding to the data that is gathered
3. Starts the NAnt-based deployment (or any executable) once the wizard has been completed, with the environment variables in effect.

The sample solution uses InstallWizard.xml during deployments (and redeployments) and UnInstallWizard.xml during un-deployments. Note that choosing "Deploy BizTalk Deployment Sample" from the Start-Programs menu above launches SetEnvUI.exe, passing InstallWizard.xml and a batch file (which calls our NAnt script) as command-line parameters. Redeployment and undeployment are similar.

InstallWizard.xml gathers the following information for the sample solution:

1. The name of a domain-qualified user account to use for the IIS6 application pool.
2. The password for this account. **Note that the password will be held in a process-level environment variable for the duration of the script.** You may decide this is unacceptable (but note that no permanent environment variable is being created!) Someone with sufficient permissions can interrogate a process for the value of environment variables while the process (or child processes) are running.
3. The name of the account used by the BizTalk host instance which will be reading/writing files for file-transport ports.
4. A file that should be used for environment-relative configuration (discussed later.)
5. An indication of whether the deployment will deploy to the BizTalk management database (used for multi-server BizTalk groups.)

As an example, given this Xml fragment:

```
<SetEnvUIConfigItem>
      <PromptText>Please enter a domain-qualified user name for virtual directory (HTTP and
SOAP) identities.

For Windows Server 2003 (IIS6), ensure this user is in the IIS_WPG group.</PromptText>
      <PromptValue />
      <ValueType>Text</ValueType>
      <EnvironmentVarName>VDIR_UserName</EnvironmentVarName>
```

```
</SetEnvUIConfigItem>
```

SetEnvUI.exe will generate this wizard step:

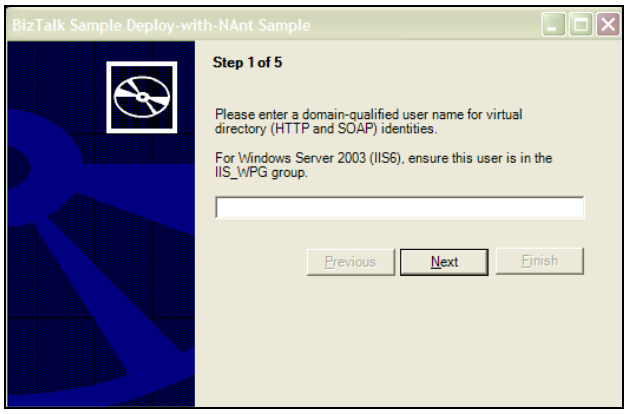


Figure 3 - SetEnvUI wizard interface

To use SetEnvUI.exe for your own projects, see SetEnvUIConfig.xsd (for the schema that the utility accepts) and InstallWizard.xml (for a good sample instance.)

6. Multiple Servers in a BizTalk Group

As noted above, SetEnvUI.exe is used to ask the person responsible for the server installation whether or not the deployment will deploy to the BizTalk management database. The question is actually phrased as:
"Is this the LAST server in the BizTalk Group you are deploying to? If so, BizTalk assemblies will be deployed to the BizTalk Management Database."

A checkbox is offered to accept the response. The idea is that you don't actually want to deploy to the BizTalk management database (and enable all ports and orchestrations) until every other server in the BizTalk Group (that may have host instances which contain your orchestrations) have the appropriate assemblies in the GAC. If you select "no" the question above, the NAnt script will simply place assemblies in the GAC (as well as a few other items – consult the script for details.)

Likewise, when undeploying, the question is asked "Is this the FIRST server in the BizTalk group you are undeploying from?" If so, BizTalk assemblies will be undeployed from the BizTalk management database (and orchestrations/ports are stopped). Subsequent servers (where the answer will be "no") will simply have assemblies removed from the GAC, etc. Within the core script, this is coordinated through use of the deployBizTalkMgmtDB property.

7. Environment-Relative Configuration for Server Deployments

Environment-relative configuration (that is, configuration that is specific to development vs. QA vs. production environments, etc.) could be gathered in any number of ways. The tools provided here assume that it will be useful for an operations staff member to be able to easily modify this type of configuration, but that it is relatively stable. Examples of this kind of configuration include database connection strings, queue names, FTP locations, HTTP urls, etc.

The approach starts by providing the operations staff with a spreadsheet that looks like this:

	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/btshttpreceive.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

You will find an example of this Excel spreadsheet in the download package under the "EnvironmentSettings" directory (SettingsFileGenerator.xml).

The process of using the spreadsheet looks like this:

1. The BizTalk developers on a project must arrive at what the environment-relative settings are. For each such setting, a row is added to the spreadsheet. Column 'A' is used for a symbolic name (i.e. CatalogFTPDrop_Server – and it is helpful to add an Excel comment on the cell to document further.) The developers will likely populate the "default value" and/or the "Development" environment column with the correct values.

2. The other columns are used for your other physical environments, and will likely be maintained by your operations staff. One could imagine that as the project progresses, "ownership" of this entire spreadsheet will transition to the operations group.

3. BizTalk developers will lace their port binding files with a syntax that looks like the following for each environment-relative setting. The syntax allows for "pre-processing" (prior to importing the bindings into a particular environment) while leaving the file completely useable. Note that within $() symbols, the symbolic name used in Column 'A' of the spreadsheet appears, as in $(FileSendLocation).

```
<PrimaryTransport>
    <!-- ifdef ${ _xml_preprocess} -->
    <!-- <Address>${FileSendLocation}\%MessageID%.xml</Address> -->
    <!-- else -->
    <Address>C:\temp\BizTalkSample_OutDir\%MessageID%.xml</Address>
    <!-- endif -->
```

4. When a particular environment setting file (such as QA_Settings.xml) is selected during the install-time configuration phase (discussed above) a tool called XmlPreProcess.exe is used to apply the settings to the port binding file, prior to actual deployment to the BizTalk management database.

5. The settings files can be maintained in source control and deployed with the MSI, or (if they contain sensitive information, which will often be the case) placed on a secured file share that is browsed to during install-time configuration.

Note that you can encapsulate environment-relative settings in the spreadsheet that affect more than just the port bindings file! You can pre-process as many XML files as you like. In a customDeployTarget within MyProject.sln.deploy.build, consider something like the following:

```
<if propertyexists="serverDeploy" propertytrue="serverDeploy">
<exec program="deploytools\xmlpreprocess.exe" failonerror="true"
commandline="/o:SomeOtherFile.xml /i:SomeOtherFile.xml
/s:&quot;${sys.env.ENV_SETTINGS}&quot;"/>
</if>
```

In addition, if you need to make use of environment-relative settings in your project-specific NAnt script (MyProject.sln.deploy.build), define a property in that file as shown below. It should contain a comma separated list of NAnt properties whose values should be determined from the settings file that is in effect. This slightly different technique is required since pre-processing the NAnt file itself (with XmlPreProcess) would be problematic given the sequence of operations in a deployment.

In this example, we are showing the case where you want the actual names of the SSO Application Users group and SSO Administrators group to flow from the settings spreadsheet (because the group names may vary by environment you deploy to.) (This could also be considered install-time configuration and gathered as a Wizard step as discussed above.)

```
<property name="propsFromEnvSettings" value="ssoAppUserGroup,ssoAppAdminGroup"/>
```

Complete credit goes to Loren Halvorsen for the spreadsheet and XmlPreProcess tools – see http://weblogs.asp.net/lorenh and the active project at http://sourceforge.net/projects/xmlpreprocess. Loren developed this process to help manage web.config and other xml files for ASP.NET projects. Thomas Abraham improved on the original concept by eliminating the need for Excel macros and instead exporting the settings at runtime, and allowing the spreadsheet to be saved in either binary or SpreadsheetML (XML) format (http://www.codeplex.com/EnvSettingsManager).

8. Runtime Configuration with SSO-Backed Storage

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.

In addition to storing environment-relative settings within SettingsFileGenerator.xml that are required for "install time substitution", as discussed above, you can also store name-value pairs *that are required at run time.* (If these are *not* environment-relative, just populate the "default" column in the spreadsheet.)

In this case, the "symbolic name" that appears in the first column of the spreadsheet will not be used as a "macro" within a port binding file (or other XML file.) Instead, it will be retrieved programmatically within an orchestration or within an assembly that the orchestration calls.

To get this behavior, set the "includeSSO" property within MyProject.sln.deploy.build to "true". This will cause the settings file (generated by the spreadsheet) that is appropriate for the environment to be deployed to the BizTalk SSO database. The "affiliate application" that is created will be named after your nant project name.

Then, have your orchestration and component assemblies reference the SSOSettingsFileReader assembly (in the DeployTools directory). The SSOSettingsFileReader class has static methods for retrieving an individual value (best for orchestrations) as well as for retrieving a hashtable for the entire set of name-value pairs (best for assemblies that orchestrations call.) See TopLevelOrch and MyClass.cs within the sample. Note that this class caches the settings so that the SSO database is only contacted once.

```
public class SSOSettingsFileReader
{
    public static string ReadString(string affiliateApplication, string valueName)...
    public static int ReadInt32(string affiliateApplication, string valueName)...
    public static void ClearCache(string affiliateApplication)...
    public static Hashtable Read(string affiliateApplication)...
}
```

Once again, it is worth noting that the sensitive information in the spreadsheet-generated settings files should (eventually) be managed by the operations team that has access to production environments, and stored on a secured file share that is browsed to when installing the application MSI.

If you need runtime configuration with SSO-backed storage *without* putting the values in the settings file (at least values for *all* environments) then consider using the "updatessoconfigitem" NAnt task to create or update a particular name-value pair. This can be used in your project-specific nant file (in a customSSO target), and can pull values from environment variables established by SetEnvUI (see 'Install-time Configuration for Server Deployments' above.) Here is an example:

```
<target name="customSSO" if="${property::exists('serverDeploy') and serverDeploy}">
   <updatessoconfigitem ssoitemname="databaseUserName"
ssoitemvalue="${sys.env.databaseUserName}"/>
   <updatessoconfigitem ssoitemname="databasePassword"
ssoitemvalue="${sys.env.databasePassword}"/>
</target>
```

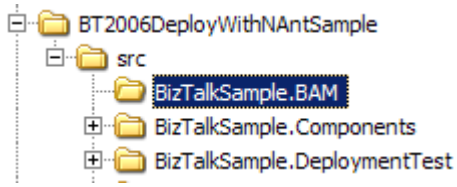
9. Deploying BAM Activities and Views

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.

BizTalk includes a separate set of command-line tools that are used only for BAM deployment and management. (A GUI wrapper is available from http://www.codeplex.com/bamgui.) The tools manipulate the BAM runtime based on a BAM model file defined by a developer in Excel or an XML file. The Deployment Framework uses these tools to manipulate the BAM model.

9.1. 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 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*. The folder structure for the BAM model is identical to the other artifact types:



Inside the BAM subfolder, the BAM XLS file must be named <projectname>.BAM.xls.

To enable this feature, simply specify in your project's build file:

```
<property name="includeBAM" value="true" />
```

When the includeBAM property is true, the Deployment Framework will export the BAM XML definition from the XLS file and then use bm.exe to deploy it.

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's build file:

```
<property name="skipBamUndeploy" value="false" overwrite="true" />
```

You can also remove the model using this command in the project root: `nant undeployBam -D:skipBamUndeploy=false`, or by calling the undeployBam target from a _customUndeployTarget target.

9.2. 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 (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.

The format of the values is:
ViewName1**:**DOMAIN\GroupName1,DOMAIN\UserName1**;**ViewName2:BUILTIN\Administrators,COMPUTERNAME\UserName2**;**<etc. etc.>

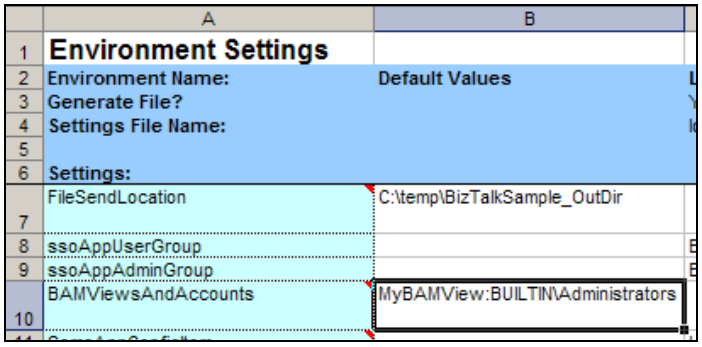


Figure 4 - Sample BAM Security Value

On an application re-deploy when the BAM model already exists, it is normal to see an error during this phase of deployment. The error is simply reporting that the specified users already have rights to the views.

10. Controlling .NET Application Domains (AppDomains)

.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 enable this feature, simply specify in your project's build file:

```
<property name="useIsolatedAppDomain" value="true" />
```

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

11. Side-by-Side (SxS) Version Deployment

Sometimes one must deploy multiple versions of the same BizTalk application to the same BizTalk server. To make this easier, the script can auto-append a version number to the BizTalk application name, SSO affiliate app name, MSI product name and install path to avoid conflicts. *In the future* it may also auto-append a version number to the port names.

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 rest (**Note: 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.

We declare a script variable called "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.**

Finally, we declare a "ProductVersion" variable in the WiX setup script. 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 is that ProductVersion will change often and perhaps automatically (maybe on every rebuild), but projectVersion will change infrequently.

To enable side-by-side, add these lines to your project's build file:

```
<property name="enableSideBySide" value="true" />
<property name="projectVersion" value="1.0" />
```

Set the projectVersion value to the same Major.Minor version of your BizTalk assemblies.

12. MSI Installer Generation

To use the automated WiX MSI generation, you should look at the BizTalkSample.WiXSetup directory contents in the sample. WiX is an open-source tool set released by Microsoft – see http://sourceforge.net/projects/wix/. It describes an Xml grammar for MSI files, and contains a set of tools for compiling these xml documents into an MSI.

The same pattern that appears for the deployment NAnt scripts is applied here – the sample solution contains BizTalkSample.WiXSetup.build, which represents a typical example of how to leverage the core functionality within BizTalk.WiXSetup.nant (where some of the generic portions are housed.)

You supply your own implementation of BizTalkSample.WiXSetup.build – say MyProject.WiXSetup.build – that includes BizTalk.WiXSetup.nant. These NAnt scripts coordinate the following:

1. Assembling a directory structure (derived from your existing directory structure!) that describes what should appear in the MSI. This directory structure winds up appearing underneath BizTalkSample.WiXSetup\Setup\redist. Knowledge of the assemblies that are needed is gleaned by including the deployment nant scripts! Any custom work needed should appear in a target called customRedist.
2. Calling out to a script called generate-install-script.js that creates portions of the needed WiX xml document.
3. Calling out to the WiX "candle" utility to compile the MSI.

Copy the entire contents of the BizTalkSample.WiXSetup directory to your own project (renamed appropriately.) Rename BizTalkSample.WiXSetup.Build to MyProject.WixSetup.build and customize if necessary (including whether debug or release binaries will be included in the MSI.) At that point, you can cause the MSI to be built simply by running nant.exe in the MyProject.WiXSetup\Setup directory.

Note: Do not disable 8.3 short name functionality through the registry (for the file system.)

13. Deployment Verification

If you build the Setup project included with this sample, you will find you can run the MSI, and then choose "Deploy…" from your Start-Programs menu and complete the deployment process. At that point, you can drop the file located in the TestFiles directory into BizTalkSample_InDir, and expect to see output in BizTalkSample_OutDir – this exercises the TopLevelOrch orchestration.

Likewise, you can issue an HTTP post against the http://localhost/BizTalkSampleVDir/btshttpreceive.dll URI to exercise the Echo orchestration. Notice that in both cases, the environment has been prepared by the deployment script to match the installation conditions (i.e. virtual directory and file send/receive locations.)

After a deployment of your BizTalk 2006 assets through the MSI-and-NAnt approach described above, it is helpful to have a mechanism to verify that the deployment is correct and functioning.

To do this, it is recommended that you take a subset of your NUnit-based tests, and deploy a subset of NUnit as shown in the BizTalkSample.WiXSetup project (set includeDeploymentTest=true.) Note that the BizTalkSample.DeploymentTests project contains some extremely simple (non-real-world) unit tests, and that this assembly is deployed to our target directory by the MSI. (To build BizTalkSample.DeloymentTests, you may need to patch the references to NUnit assemblies in this project to match your local installation.) A TestFiles directory is deployed as well, which contains files we need to conduct tests.

Moreover, the "Verify Deployment" menu choice that is deployed by the sample MSI (shown above) will fire off the NUnit GUI and run the tests contained in this assembly. This general technique can be an *extremely effective* method of ensuring that operations personnel have a reliable way to test the integrity of the system. You may even build a monitoring strategy off of nunit-console.exe with the /xmlConsole parameter – a monitoring tool can schedule a job that parses the returned xml for pass/fail/time-taken.

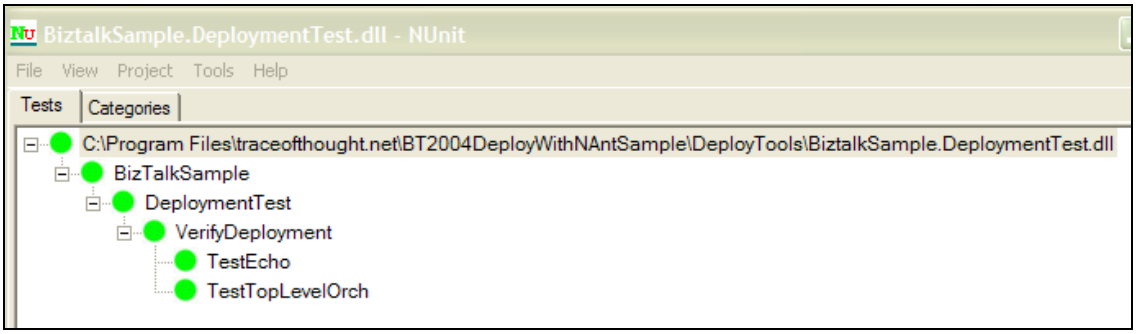


Figure 5 - Sample NUnit test output

14. Additional Options in the Deployment Script / FAQ

Q: How can I use different names for my various projects than what is proposed in the naming conventions?
A: In your MyProject.sln.deploy.build file, provide your own definitions for the nant properties defined in BizTalkDeploymentInclude.nant (i.e. components, schemas, pipelines, transforms, pipelineComponents, customFunctoids, orchestrations.) These will override the defaults.

Q: How can I *not* use the proposed naming conventions for directories, project names, etc?

A: In your MyProject.sln.deploy.build, set the useCustomDirs property to true, and override the properties discussed above to include *full path names*. You will have to manage the variations between debug, release, and server deployments yourself (whereas with useCustomDirs set to false, this is handled for you.)

Q: How can I have multiple schema assemblies, orchestration assemblies, component assemblies, etc.?
A: Supply your own definition for the properties defined above (in Q1), providing a comma-separated list rather than a single assembly. Order schema assemblies in least-dependent-first order if they reference each other, likewise with orchestrations assemblies.

Q: How can I make sure my virtual directory that uses btshttpreceive.dll is registered as a web service extension with IIS6?
A: Supply a value for the wseExtensionPath property (to be equal to the name of the subdirectory of your solution where btshttpreceive.dll resides) and the wseExtensionName property (just a symbolic name.) See the deployVDirs target for details, but the sample shows how this works.

Q: What is the difference between "deploy" and "redeploy" for server deployments?
A: Deploy assumes that your BizTalk assets have not been deployed to this server before, so it doesn't spend time trying to undeploy. Redeploy functions like the developer workstation experience, where the script attempts to undeploy your assemblies, etc. first. Redeploy should be used when install-time or environment-relative configuration has changed, or an assembly has been "patched" (i.e. dropped in without a formal MSI install.)

Q: How do I selectively restart BizTalk hosts, rather than restarting them all?
A: In your MyProject.sln.deploy.build file, override the property definition for bizTalkHosts

Q: All I did was change my orchestration (or my components, or SSO data) a little. Do I have to redeploy the world?
A: No, you should create an external tool in Visual Studio for the updateOrchestration target. This also redeploys your components and SSO configuration – it is pretty quick.

Q: My http receive virtual directory or SOAP virtual directory needs to run with a particular account identity.
A: The script can apply an account to either a) the associated COM+ package for a high-isolation vdir under IIS 5.0/5.1 for Win2k/Winxp or b) an IIS6 application pool. Populate the MyProject.VDirList.txt file with virtual directory, physical directory, app-pool tuples (comma separated.) Of course, multiple virtual directories can use the same physical directory when using btshttpreceive.dll

Q: I have to deploy BizTalk rule engine rules…
A: Set the ruleVocabulary, rulePolicy, rulePolicyName properties, as shown in BizTalkSample.sln.deploy.build. The task within BizTalkDeploymentInclude.nant delegates to DeployBTRules.exe (which you could call directly from your customDeployTarget if you like.)

Q: How is starting/stopping orchestrations and ports handled?
A: By using a set of custom NAnt tasks (within BizTalk.NAnt.tasks.dll) that call the BizTalk Explorer OM. The names of the orchestrations are derived from the assembly itself, and the names of the various kinds of ports are derived from the binding file. For 2006, we start/stop using the Application object in the Explorer OM.

Q: I have some ports (or orchestrations) that I *don't* want automatically started when I deploy.
A: In your MyProject.sln.deploy.build file, provide a definition for the property 'portsToExcludeFromStart' – a comma separated list of send port groups, send ports, and receive locations that should not be started. For orchestrations, define the property 'orchestrationsToIgnore' with a comma separated list. (This doesn't work for 2006 – you can get rid of the startApplication target and manually start using the MMC if you like.)

Q: It sure seems like there are a lot of pieces here…is there a clear picture of what I own, vs. what the deployment framework owns?
A: See the BizTalk2006NAntDeploymentElements.vsd diagram.

Q: I have more custom work that I need to do. Can I do stuff before/after deployment takes place?
A: Yes – you can define any of customDeployTarget, customPostDeployTarget, customUndeployTarget, customPostUndeployTarget can be defined if need be.

Q: I have some installer classes that I need to have called. Can I make that happen with the deployment framework?
A: Set a property called includeInstallUtilForComponents to 'true' in your project-specific nant file. This will cause all your component assemblies to be called with "installutil.exe" (with the /u flag for undeployment.) This is useful if you have .NET Installer-derived classes that need to get called for creating event sources, perf counters, etc. (Yes, you could certainly argue these should be called by the WiX piece of the script infrastructure instead, but this method works and is more in keeping with the spirit of the entire process.) Note there is no harm if a given component assembly listed in your components property does not have an installer class.

Q: I need a log of success/failure kept for all BizTalk deployments in my BizTalk group.
A: The deployment framework will preserve past DeployResult.txt files with this convention: 'DeployResults_<machinename>_05032005_0922.txt' Note that the machine name, date, and time are preserved – so you can gather these files across all the servers in a BizTalk group, if need be.

Q: I need to change how much time is allocated for host service restarts and IIS restarts.
A: Alter the bizTalkHostResetTime or iisResetTime properties

15. Appendixes

15.1. Appendix A: Getting Started with this Sample

In order to play with this sample, modify it, build it, and deploy it, you will need to perform some pre-requisite tasks.

15.1.1. Bootstrapping NUnit

To get started with NUnit, you can follow these steps:

1. Download and install the latest NUnit (currently NUnit-2.4.8-net-2.0.msi) from http://www.nunit.org/index.php?p=download
2. Test the NUnit installation
    a. Start -> Programs -> NUnit 2.4.8 -> NUnit-Gui
    b. File -> Open -> nunit.tests.dll (located in the NUnit installation bin directory)
    c. Click the 'Run button' (all 605 tests should pass)

15.1.2. Bootstapping NAnt

To get started with NAnt, you can follow these steps:

1. Download the latest NAnt release (currently 0.85) from http://nant.sourceforge.net and unzip it under C:\Program Files\NAnt.
2. Download the latest NAntContrib release (currently 0.85) from http://nantcontrib.sourceforge.net and copy all files in NAntContrib's bin directory into NAnt's bin directory.

15.1.3. Install external tools into Visual Studio

Download and unzip "DFBT2006WithNAntSample_V40.zip" from http://www.codeplex.com/biztalkdeployment/Release/ProjectReleases.aspx and run the MakeBizTalkExternalTools_VS2005.vbs script. This will add the appropriate external tools to Visual Studio.

15.1.4. BizTalk Server 2006 Sample build and deploy

1. First perform the build process without the deployment of the built assemblies: Build -> Rebuild Solution
2. Once the build succeeds, try the NAnt deployment solution: Tools -> Deploy BizTalk Solution
3. Create a c:\temp\BizTalkSample_OutDir directory
4. For development purposes, add your IWAM user account to the BizTalk Isolated Host users group (for Windows XP – otherwise, do so for the account associated with the application pool that was created.)
5. Use NUnit to run tests in BizTalkSample.DeploymentTest.dll

15.1.5. Unattended or Continuous Builds

You may wish to use NAnt to kick off full unattended builds for nightly builds or continuous integration systems (like Cruise Control). Since Visual Studio 2005 is the only supported way to build BizTalk projects - despite that tempting XSharpP.exe file sitting in your BizTalk installation directory - this part of your NAnt build file must defer to calling Devenv.exe in command-line fashion. The included sample assumes that any infrastructure for unattended or continuous builds will be done in a distinct NAnt file, and is not discussed here.

15.2. Appendix B: Common Errors

When I try to deploy from the Visual Studio Tools menu, I get file not found errors.

Ensure that your solution is set to Debug configuration, not Release. The VS Tools menu deployment options are configured to deploy only debug versions of assemblies.

15.3. Appendix C: Credits

The Deployment Framework project builds upon the valuable contributions of countless contributors to open-source projects:

NAnt
(http://nant.sourceforge.net/)

NUnit
(http://www.nunit.org/)

XMLPreprocess courtesy of Loren Halvorsen
(http://sourceforge.net/projects/xmlpreprocess/)

WiX
(http://sourceforge.net/projects/wix/)

Log4Net
(http://logging.apache.org/log4net/)

Environment Settings Manager courtesy of Thomas F. Abraham
(http://www.codeplex.com/EnvSettingsManager)

Colored output from NAnt 0.85 via a custom logger is courtesy of Eric Liu
(http://twericliu.blogspot.com/2008/03/colorized-nant-console-output.html)

SSO-based Configuration inspired by Jon Flanders
(http://www.masteringbiztalk.com/blogs/jon/PermaLink,guid,6e4b84db-d15f-45e9-b245-08b1eb6c4def.aspx)