Microsoft BizTalk Server 2006 Part-IV

Table of Contents

How to Configure Map Validation and Test Parameters

Before validating and testing a map, you need to set the map validation and test parameters in the **Property Pages** dialog box for the map.

To configure the map validation and test parameters

1. In Solution Explorer, right-click the map whose property pages you want to configure, and then click **Properties**.

2. In the ***<Map Name>*** **Property Pages** dialog box, do the following.

Use this	To do this
Validate TestMap Input	Configure whether you want to have the instance message validated against the source schema before you test the map.
Validate TestMap Output	Configure whether you want to have the instance message validated against the destination schema after you test the map.
TestMap Input Instance	Configure the location of the instance message data to use when you test the map. If you configure this property, you must also configure the **TestMap Input** property.
TestMap Input	Configure the input instance data format.
TestMap Output	Configure the output data type to use when you test the map.

3. In the ***<Map Name>*** **Property Pages** dialog box, click **OK**.

How to Validate Maps

After you have developed your map, one of the next steps is to validate it. This topic provides step-by-step instructions for validating maps.

To validate a BizTalk map

1. In Solution Explorer, open the map that you want to validate.

2. In Solution Explorer, right-click the map, and then click **Validate Map**.

3. In the **Output** window, verify the results.

How to Test Maps

After you have developed your map, one of the next steps is to test it. This topic provides step-by-step instructions for testing maps.

To test a BizTalk map

1. In Solution Explorer, right-click the map you want to test, and then click **Test Map**.

2. Verify the results in the **Output** window.

How to Create Multi-Part Mappings

If you have multiple maps that are used together, you will need to combine them in an orchestration to test them together. The **Test Map** command works on only one map at a time.

For more information about designing and using orchestrations, see Creating Orchestrations and Developing Orchestrations.

How to Specify XSLT Output Settings

You can specify whether an XML declaration is output or not. An XML declaration would appear (if you selected **No**) similar to the following.

Encoding provides the run-time engine with the information it needs to determine which character set to use when creating the output result of a map.

To include or exclude an XML declaration

1. In the Grid view, click on the mapper grid.

   The Properties window shows the **Grid** properties.

2. In the drop-down list for the **Omit XML Declaration** property, select **Yes** to omit an XML declaration, or select **No** not to omit an XML declaration.

To set encoding for output instance data

1. In the Grid view, click on the mapper grid.

   The Properties window shows the **Grid** properties.

2. In the drop-down list for the **Encoding** property, select the character set you want used for the output instance data.

Creating Pipelines Using Pipeline Designer

Microsoft® BizTalk® Server works mainly with the XML document format. For a message to take full advantage of BizTalk Server processing, it must often be transformed from its native format into its XML representation. BizTalk Server pipelines perform this transformation, as well as other data-specific actions (such as data encryption or decryption, property promotion, and so on) on incoming and outgoing messages. This section provides conceptual and task-specific information about pipelines and Pipeline Designer.

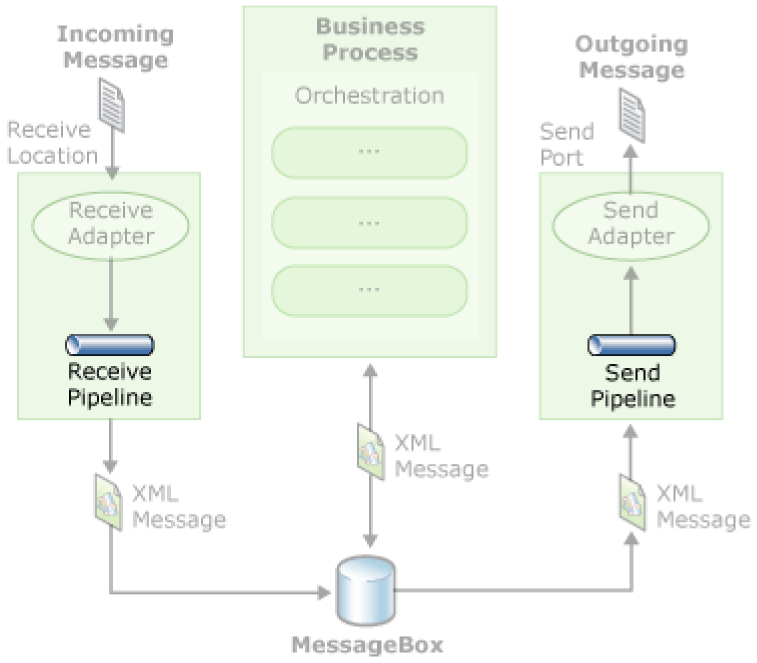
The purpose of a pipeline is to prepare a message for processing by the server after being received by an adapter or to prepare a message for sending after being processed by the server.

Pipelines commonly perform:

- Data normalization from various formats to XML.
- Data transformation from XML to various formats.
- Property promotion and demotion.
- Document disassembly and assembly.
- Document decoding and encoding.
- Document decryption and encryption.
- Document signing and digital signature verification.

The following figure shows the workflow involved in processing a message by using pipelines.

Depicts the message processing workflow.



As shown in the figure, the message is passed from the adapter to the receive pipeline where it is transformed to XML. The message can then be used by orchestrations, or passed to a send pipeline, and then to a send adapter.

In This Section

- About Pipelines, Stages, and Components
- Using Pipeline Designer
- Creating Pipelines with Pipeline Designer
- Developing Pipeline Components
- Configuring Pipeline Components
- Deploying Pipelines
- Securing Pipelines

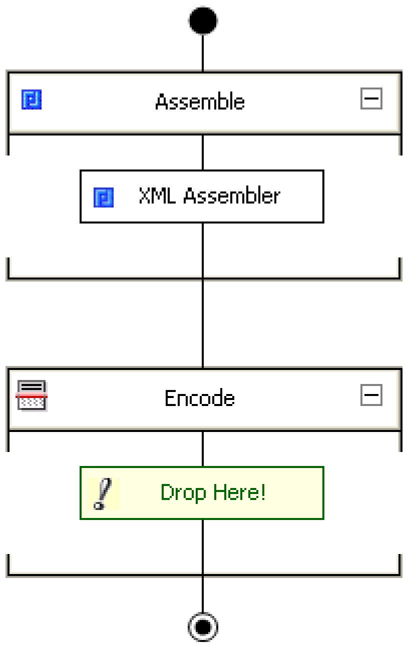
About Pipelines, Stages

A pipeline is a piece of software infrastructure that contains a set of .NET or COM components that process messages in a predefined sequence. A pipeline divides processing into categories of work called stages, and determines the sequence in which the stages are performed. Each stage defines logical work groups, determines which components can go in that stage, and specifies how the pipeline components in the stage are run.

Within each stage, pipeline components perform specific tasks. For example, components within stages of a receive pipeline may decode, disassemble, and then convert documents from other formats to XML. Send pipelines do essentially the opposite: convert documents from XML to other formats, assemble, and encrypt, with each pipeline component performing a portion of the entire process. Although a stage is a container of components, each stage is itself a component with metadata. Stages have no execution code, as opposed to pipeline components, which do have execution code.

The following figure shows how the pipeline design surface illustrates pipelines. This pipeline has two stages, the Assemble stage and the Encode stage. The XML Assembler pipeline component was added to the Assemble stage, but the Encode stage is still empty, because it still shows **Drop Here!** to indicate that a pipeline component can be added to the stage.

Illustrates stages and components in a BizTalk pipeline.



Microsoft® BizTalk® Server contains a set of pipeline templates, pipeline components, and default pipelines. You can create and configure pipelines by using the Pipeline Designer user interface; you implement pipelines by using the API in the **Microsoft.BizTalk.Component.Interop** namespace. You cannot modify the pipeline templates.

In This Section

- Types of Pipelines
- Types of Pipeline Components
- Pipeline Stages
- Default Pipelines
- Pipeline Templates
- Pipeline Components
- Schema Resolution in Pipeline Components
- Envelope Use in the XML Assembler and Disassembler Pipeline Components
- Property Demotion in Assembler Pipeline Components
- Property Promotion in Disassembler Pipeline Components
- Distinguished Fields in Disassembler Pipeline Components

Types of Pipelines

When creating a new pipeline in a BizTalk project, you have two options, a receive pipeline and a send pipeline. This section explains the differences between the pipeline types, and design considerations for each.

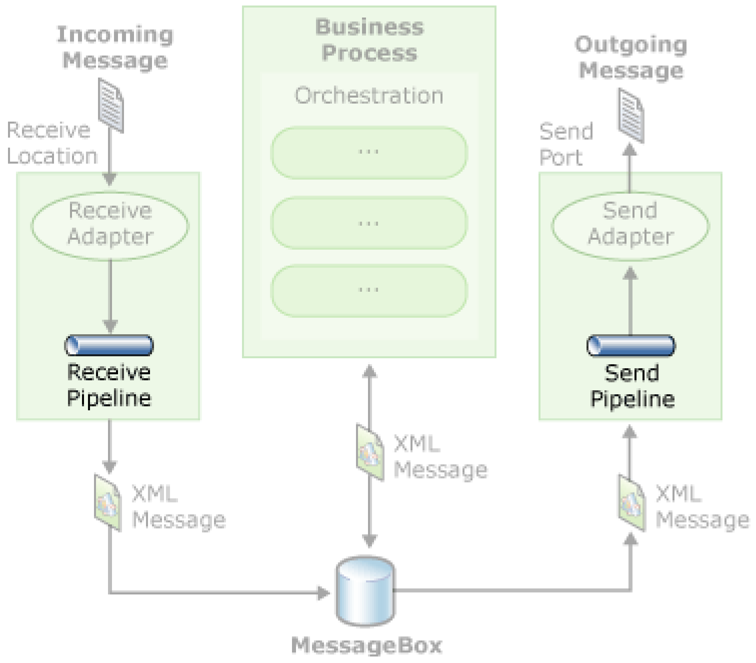
In This Section

- Receive Pipelines
- Send Pipelines

Receive Pipelines

The following figure shows the message processing workflow, highlighting the receive pipeline.

Depicts the message processing workflow.



A receive pipeline operates on a message after it is received by the receive adapter. The receive pipeline takes the initial message, performs some transformations, and disassembles the raw data into zero, one, or multiple messages. These individual messages can then be processed by BizTalk Server.

When creating a business process, you can create a new receive pipeline or you can use one of the two default receive pipelines included in BizTalk Server—the pass-through receive pipeline or the XML receive pipeline. For more information about these default pipelines, see Default Pipelines.

The receive pipeline consists of four stages: Decode, Disassemble, Validate, and ResolveParty. This topic contains design considerations for populating these stages.

Decode stage

- This stage is used for components that decode or decrypt the message.
  - The MIME/SMIME Decoder pipeline component or a custom decoding component should be placed in this stage if the incoming messages need to be decoded from one format to another.
- This stage takes one message and produces one message.
- This stage can contain between zero and 255 components.
- All components in this stage are run.

Disassemble stage

- This stage is used for components that parse or disassemble the message.
  - The components within this stage probe the message to see if the format of the message is recognized. Based on the recognition of the format, one of the components disassembles the message.
  - If this stage contains more than one component, only the first component that recognizes the message format is run. If none of the components within the stage recognize the message format, the message processing fails.
  - This stage should include any custom components that implement special behavior to disassemble the message contents.
  - This stage can contain between zero and 255 components. If there are no components in the stage, the message is passed through.

Validate stage

- This stage is used for components that validate the message format.
  - A pipeline component processes only messages that conform to the schemas specified in that component. If a pipeline receives a message whose schema is not associated with any component in the pipeline, that message is not processed. Depending on the adapter that submits the message, the message is either suspended or an error is issued to the sender.
  - Components in this stage are used to validate the XML messages produced by the Disassemble stage. Components in this stage specify schemas to perform the XML validation.
- This stage can contain between zero and 255 components.
- All components in this stage are run.
  - This stage may be run more than once. It runs once per message created by the Disassemble stage.

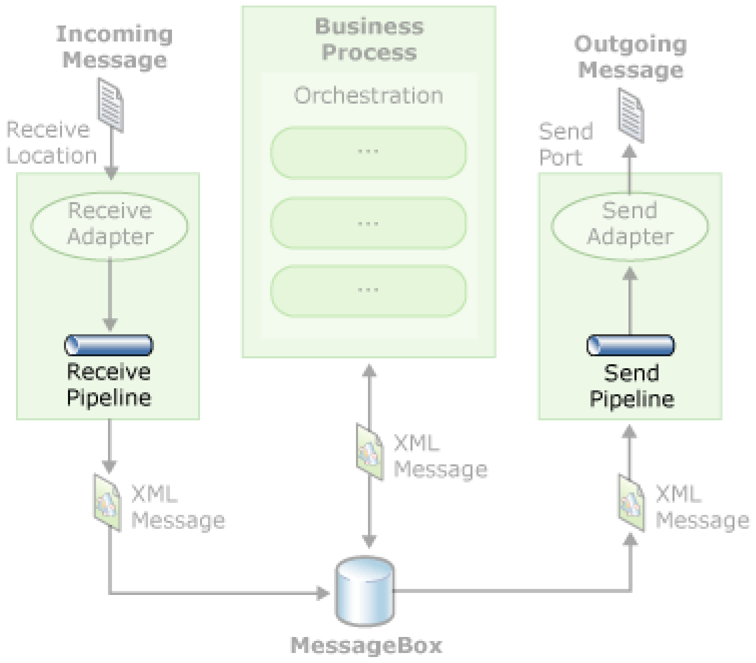
ResolveParty stage

- This stage is a placeholder for the Party Resolution Pipeline Component.
- This stage may be run more than once. It runs once per message created by the Disassemble stage.
- This stage can contain between zero and 255 components.
- All components in this stage are run.

Send Pipelines

The following figure shows the message processing workflow, highlighting the send pipeline.

Depicts the message processing workflow.



A send pipeline is responsible for processing documents before sending them to their final destinations. The send pipeline takes one message and produces one message to send.

You can create a new send pipeline or you can use one of the two default send pipelines included in BizTalk Server—the pass-through send pipeline and the XML send pipeline.

By default, the send pipeline consists of three empty stages: Pre-assemble, Assemble and Encode. This topic contains design considerations for populating these stages.

Pre-assemble stage

- This stage is a placeholder for custom components that should perform some action on the message before the message is serialized.
- This stage is run once per message.
- This stage can contain between zero and 255 components.

- All components in this stage are run.

Assemble stage

- Components in this stage are responsible for assembling or serializing the message and converting it to or from XML.
- This stage accepts zero components or one component.
- All components in this stage are run.

Encode stage

- This stage is used for components that encode or encrypt the message.
  - Place the MIME/SMIME Encoder component or a custom encoding component in this stage if message signing is required.
- This stage is run once per message.
- This stage can contain between zero and 255 components.
- All components in this stage are executed.

Types of Pipeline Components

Three types of pipeline components are included with BizTalk Server 2006: general, assembling, and disassembling. Any of these three types can also implement probing functionality. This topic describes each type of component and the stages in which each component is generally used.

General

General components take one message, process the message, and produce zero or one message.

The MIME/SMIME Decoder, MIME/SMIME Encoder, Party Resolution, and Validator components are the included general components. You may need to create custom general components to compress the size of a message before sending, or to consume a message while waiting for additional information to process it.

General components should be placed in the Decode, Encode, Pre-assemble, ResolveParty, or Validate stages.

For information about developing general pipeline components, see Developing a General Pipeline Component.

Assembling

Assembling components have numerous responsibilities to prepare the message to be sent. First, the component converts the XML message to the appropriate XML or non-XML native format of the message, based on the type of assembler and properties set in the schema. In addition, assembling components assemble and wrap the message in an envelope, or add a header or trailer (or both) to the message. During assembly, some properties are moved from the message context to the body of the document or to the envelope.

The BizTalk Framework Assembler, Flat File Assembler, and XML Assembler components are the default assembling components.

Assembling components should be placed in the Assemble stage of send pipelines.

For information about developing assembling pipeline components, see Developing an Assembling Pipeline Component.

Disassembling

Disassembling components complete many tasks to prepare the message to be split up into individual documents according to envelope and document schemas for use within BizTalk Server. First, the disassembling component may convert non-XML messages into their XML representation, which is required for processing by BizTalk Server. Next, the message is disassembled into singular messages that can be sent to separate orchestrations. The message is disassembled by removing the envelope, splitting the message up into individual documents according to envelope and message schemas, and then moving properties from the envelope to the individual message contexts. Additionally, some properties may be promoted from the body of the message to the header. The promoted properties are determined by the schema.

The disassembling component must also set the message type property, which is used for routing messages appropriately. The message type property is the Namespace#RootElement of the message body. Other properties, such as the content type and character set are set as part of the context property.

The BizTalk Framework Disassembler, Flat File Disassembler, and XML Disassembler components are the default disassembling components included with BizTalk Server.

Disassembling components should be used in the Disassemble stage of receive pipelines.

For information about developing disassembling pipeline components, see Developing a Disassembling Pipeline Component.

Probing

A probing component checks the first portion of the message to see if it is in a format that the component understands. If the format is known, the whole message is given to this component for processing.

For information about developing probing pipeline components, see Developing a Probing Pipeline Component.

Pipeline Stages

This topic discusses the **Execution Mode** property and stage affinity.

Execution Mode property

During the execution of a pipeline, the pipeline stages can run only the first component that recognizes the message format, or all components. The property that determines the execution pattern is **Execution Mode**. This property is read-only on the stages included in the pipeline templates, but understanding how it works is an important concept.

When the **Execution Mode** property is set to **All**, all the components within the stage are run in the configured sequence. Use this mode when several components must be run to complete a logical task. In this case, a run-time error results if any component encounters an error while processing a message during this pipeline stage.

When a pipeline is used to receive messages in several formats, then you can set its **Execution Mode** property to **FirstMatch**. In this mode, only the first component that recognizes the message is run. If no components in the stage recognize the message, a run-time error results.

Note that each stage can have its own **Execution Mode** setting, so different stages within a pipeline can have different execution modes.

To read pipeline stage properties

1. In Pipeline Designer, click a stage shape.
2. In the Properties window, in the **General** section, read the following properties:

Use this	To do this
Name	Indicates the name of the stage.
Execution Mode	Indicates the execution pattern of the stage.

	Valid values: **All** or **FirstMatch**
Minimum Number of Components	Indicates the minimum number of pipeline components that can be added to the stage.
Maximum Number of Components	Indicates the maximum number of pipeline components that can be added to the stage.
StageID	Indicates the unique identifier for the stage.

Stage affinity

Pipeline components have stage affinity, meaning that they are created for use within a particular stage or stages in a pipeline.

COM-based pipeline components express their stage affinity by registering themselves using the stage ID as the implementation category, while .NET-based pipeline components specify their stage affinity by using the **ComponentCategory** class attribute. Note that it is possible for a component to associate itself with more than one stage—components can have more than one implementation category or **ComponentCategory** attribute.

The following table shows the available component categories and their associated stages.

Component category	**Stage where component can be placed**	**Description**
CATID_Decoder {9d0e4103-4cce-4536-83fa-4a5040674ad6}	Decode	All decoding components should implement this category.
CATID_DisassemblingParser {9d0e4105-4cce-4536-83fa-4a5040674ad6}	Disassemble	All disassembling and parsing components should implement this category.
CATID_Validate {9d0e410d-4cce-4536-83fa-4a5040674ad6}	Validate	Validation components should implement this category.
CATID_PartyResolver {9d0e410e-4cce-4536-83fa-4a5040674ad6}	ResolveParty	Stage used for Party Resolution component.
CATID_Encoder {9d0e4108-4cce-4536-83fa-4a5040674ad6}	Encode	All encoding components should implement this category.

CATID_AssemblingSerializer {9d0e4107-4cce-4536-83fa-4a5040674ad6}	Serialize	All serializing and assembling components should implement this category.
CATID_Any {9d0e4101-4cce-4536-83fa-4a5040674ad6}	Any of these stages	If a pipeline component implements this category, it means that the component can be placed into any stage of a pipeline.

Default Pipelines

When you create a new application, the default pipelines are created and deployed by default and appear in the Microsoft.BizTalk.DefaultPipelines assembly in the \References folder for every BizTalk project. The default pipelines cannot be modified in Pipeline Designer. These pipelines can be selected when configuring a send port or receive location in BizTalk Explorer. This topic describes the default pipelines and when to use them.

PassThruReceive pipeline

The pass-through receive pipeline has no components. It is used for simple pass-through scenarios when no message payload processing is necessary. This pipeline is generally used when the source and the destination of the message are known, and the message requires no validation, encoding, or disassembling. This pipeline is commonly used in conjunction with the pass-through send pipeline.

Because it does not contain a disassembler, the pass-through receive pipeline cannot be used to route messages to orchestrations.

PassThruTransmit pipeline

The pass-through-send pipeline has no components. This pipeline is generally used when no document processing is necessary before sending the message to a destination.

XMLReceive pipeline

The XML receive pipeline consists of the following stages:

- **Decode.** Empty
- **Disassemble.** Contains the XML Disassembler component
- **Validate.** Empty

- **ResolveParty.** Runs the Party Resolution component, which resolves the certificate subject or the source security ID to the party ID.

XMLTransmit pipeline

The XML send pipeline consists of the following stages:

- **Pre-assemble.** Empty
- **Assemble.** Contains the XML Assembler component
- **Encode.** Empty

Pipelines included for backward compatibility

Do not use the StsFileReceive, StsReceive, or StsSend pipelines. They are included in BizTalk Server 2006 only for backward compatibility with the BizTalk SharePoint Messaging Adapter component in BizTalk Server 2006 Business Activity Services (BAS). In BizTalk Server 2006 this component has been replaced with the BizTalk 2006 native adapter for Windows SharePoint Services, which does not have a dependency on these pipelines or the containing assembly.

StsFileReceive pipeline

The StsFileReceive pipeline consists of the following stages:

- **Decode.** Empty
- **Disassemble.** Contains the XML Disassembler component
- **Validate.** Empty
- **ResolveParty.** Empty

StsReceive pipeline

The StsReceive pipeline consists of the following stages:

- **Decode.** Empty
- **Disassemble.** Contains the XML Disassembler component
- **Validate.** Empty
- **ResolveParty.** Runs the Party Resolution component, which resolves the source security ID to the party ID. The certificate subject is not used for party resolution.

StsSend pipeline

The StsSend pipeline consists of the following stages:

- **Pre-assemble.** Empty
- **Assemble.** Contains the XML Assembler component configured to add the Microsoft.BizTalk.KwTpm.StsDefaultPipelines.EnvSchema XML envelope around the message
- **Encode.** Empty

Pipeline Templates

In addition to the default pipelines, BizTalk Server 2006 includes two pipeline templates: a receive pipeline template and a send pipeline template. From a BizTalk project in Microsoft® Visual Studio® .NET, you can add a pipeline template to your project by using the **Add New Item** command on the **Project** menu. Each template has an associated policy file, which determines the pipeline's stages and indicates which pipeline components are allowed in the pipeline. While you cannot reorder the stages in a policy file, you can use Pipeline Designer to reorder the components within a stage.

A policy file specifies:

- The sequence of stages.
- The number of components allowed per stage.
- The execution mode of each stage.

The policy files for the pipeline templates are stored in <*BizTalk Server installation directory*>\Developer Tools\Pipeline Policy Files. Do not modify the policy files. To make changes to a pipeline, open the pipeline template and use Pipeline Designer to modify it. For more information about using Pipeline Designer, see Using Pipeline Designer.

The empty pipeline template files are stored in <*BizTalk Server installation directory*>\Developer Tools\BizTalkProjectItems, and are named BTSReceivePipeline.btp and BTSTransmitPipeline.btp. The file name extension .btp indicates that the file is a BizTalk Server pipeline and can be edited in Pipeline Designer.

Pipeline Components

The Visual Studio Toolbox is populated with several standard BizTalk Server components that you can use to create a pipeline.

In This Section

- BizTalk Framework Assembler Pipeline Component
- BizTalk Framework Disassembler Pipeline Component
- Flat File Assembler Pipeline Component
- Flat File Disassembler Pipeline Component
- XML Assembler Pipeline Component
- XML Disassembler Pipeline Component
- MIME/SMIME Decoder Pipeline Component
- MIME/SMIME Encoder Pipeline Component
- Party Resolution Pipeline Component
- XML Validator Pipeline Component

BizTalk Framework Assembler Pipeline Component

The BizTalk Framework is one approach for doing exactly-once guaranteed delivery using over-the-wire transport protocols such as HTTP or SMTP. This framework has existed since 1998, and can be thought of as a precursor to pending standards initiatives based on Web services, specifically WSReliable. Typically, the problem of guaranteed exactly-once delivery of data has been the domain of technologies like Message Queuing (also known as MSMQ). However, such technologies usually require common software at the two endpoints of a data flow, and also do nothing to address the use of open transport protocols based on public networks, for example, data that flows across enterprise boundaries by using the Internet.

Not surprisingly, the BizTalk Framework implements some of the same mechanisms present in earlier attempts to solve this problem of guaranteed exactly-once delivery of data. A good example of other solutions to the problem is electronic data interchange (EDI), where ANSI X12 control numbers and standard 997 functional acknowledgment documents form the basis of guaranteeing that data is received only one time, and that the sender is notified of any problems on the receiving end.

The BizTalk Framework assumes that, however disparate the systems trading data, they both understand the BizTalk Framework protocol requirements of:

- Using a predictable envelope format for wrapping transmissions.
- Tagging every outbound transmission with a globally unique identifier.
- Always returning to the sender an acknowledgment of receipt that includes the globally unique identifier, even for data already received, acknowledged, and processed.
- Some means by which the sender can repeat transmission until either a receipt arrives from the receiver, or some time period passes beyond which the transmission is no longer valid.

The BizTalk Framework Assembler pipeline component is responsible for serializing the BizTalk Framework envelope and contents onto the message before transmission and resending in the event that a receipt does not arrive in the allotted time period. It is also responsible for receiving and processing the receipts and deleting the message instance. (A copy of the message instance of the sent message is kept in the MessageBox database until BizTalk receives a confirmation receipt from the destination. After the confirmation receipt is received, the message instance is deleted by the Messaging Engine.)

BizTalk Framework Disassembler Pipeline Component

The BizTalk Framework Disassembler pipeline component parses XML data and determines whether it contains a BizTalk Framework-based messaging payload. The pipeline component saves the message context, and a new message context is created with the BizTalk Framework property that needs to be generated. This property is used to route the message to the BizTalk Framework inbound handler, so it can receive the message to process.

The BizTalk Framework Disassembler pipeline component generates an acknowledgment for the generated message if the sender requested one using the deliverReceiptRequest header within the BizTalk Framework envelope. This is controlled by the generate delivery receipt property in the BizTalk Framework Assembler pipeline component. For more information about the BizTalk Framework, see BizTalk Framework Assembler Pipeline Component.

For information about configuring the BizTalk Framework Disassembler pipeline component and creating the orchestration, see Configuring the BizTalk Framework Disassembler Pipeline Component.

Flat File Assembler Pipeline Component

A Flat File Assembler combines individual documents into a batch and optionally adds a header or trailer (or both) to it. For more information about flat files, see Flat File Messages with Delimited Records and Flat File Messages with Positional Records.

The Flat File Assembler pipeline component does not validate the incoming XML message. To ensure XML validation, include the XML Validator component in the Pre-Assemble stage of the send pipeline.

The Flat File Assembler pipeline component only supports conversions supported by the Microsoft .NET Framework. Any additional conversion can be done by writing to a custom text writer.

For information about configuring the Flat File Assembler pipeline component, see Configuring the Flat File Assembler Pipeline Component.

In This Section

- Character Encoding in the Flat File Assembler Pipeline Component
- Document Structure Enforcement in the Flat File Assembler Pipeline Component

Character Encoding in the Flat File Assembler Pipeline Component

The Flat File Assembler can produce messages in user-specified character encoding. You can specify the character encoding at several levels:

- **Schema.** Set the **codepage** property in the flat file schema for the document.
- **Component.** Set the **Target Charset** component property in Pipeline Designer.
- **Message.** Set the **XMLNorm.TargetCharset** property on the message context.

The value of the property set on a message context always overrides the one set in Pipeline Designer. Also, the value set in Pipeline Designer always overwrites the value set as a **codepage** property in a flat file document schema.

The Flat File Assembler uses the following algorithm to determine which encoding to use for an output message:

- If the **XMLNorm.TargetCharset** context property is set, its value is used for encoding.
- Otherwise, if the **Target charset** property in Pipeline Designer is specified, its value is used.
- Otherwise, if the **codepage** property in the flat file schema is specified, its value is used.

- Otherwise, if the **XMLNorm.SourceCharset** property is specified, its value is used.
- Otherwise, "UTF-8" is used. Note that the Flat File Assembler pipeline component does not put byte order mark on outgoing messages when UTF-8 encoding is used.

The Flat File Assembler saves encoding information on the body part of the BizTalk message object in the **IBaseMessagePart.Charset** property.

Document Structure Enforcement in the Flat File Assembler Pipeline Component

If document or envelope schemas are explicitly referenced in the Flat File Assembler, the Flat File Assembler ensures that only documents with the message type corresponding to referenced schemas are processed. All the other documents are not processed even though a schema may be deployed for them.

Flat File Disassembler Pipeline Component

The Flat File Disassembler component parses delimited and positional flat file format messages and converts them into an XML representation. The Flat File Disassembler also removes the header and trailer structures from the flat file message, and breaks the interchange within the message into individual documents. It also promotes properties from the documents and headers.

For more information about flat files, see Flat File Messages with Positional Records and Flat File Messages with Delimited Records.

For information about configuring the Flat File Disassembler pipeline component, see Configuring the Flat File Disassembler Pipeline Component.

In This Section

- Document Validation in the Flat File Disassembler Pipeline Component
- Character Encoding in the Flat File Disassembler Pipeline Component

Document Validation in the Flat File Disassembler Pipeline Component

By default, the Flat File Disassembler component does not validate documents it processes. However, you can turn validation on by setting the **Validate document structure** property on the component to **True**, or by setting the **FFDasm.ValidateDocumentStructure** message context property to **True**. When document validation is set to run, the Flat File Disassembler validates the document structure as well as the header and trailer structures to ensure that they conform to the document, header, and trailer schemas.

The Flat File Disassembler can remove empty fields and records when **suppress_empty_nodes="True"** is specified by the **schemaInfo** annotation in the flat file XSD schema. If you use the **schemaInfo** annotation in this way, the Flat File Disassembler removes empty fields and records regardless of whether they are optional. This may cause validation errors if you use XML validation (either by setting the Flat File Disassembler **Validate document structure** property to **True** or by using the XML Validator pipeline component). If a validation error occurs, the message is suspended. For more information about the suppress_empty_nodes property, see Additional Flat File Properties.

Character Encoding in the Flat File Disassembler Pipeline Component

The following algorithm is used by the Flat File Disassembler component to determine which encoding to use for processing an incoming message:

1. If a byte order mark exists in the data, encoding information is determined from it. This encoding information is not preserved by the disassembler (that is, it is not saved to the **XMLNorm.SourceCharset** property).

2. Otherwise, if the **IBaseMessagePart.Charset** property is set, the encoding specified there is used.

3. Otherwise, if the header or document schema contains codepage information, it is used.

4. Otherwise, UTF-8 encoding is used.

For the preceding cases 2, 3, and 4, the disassembler saves the encoding information on the message context in the **XMLNorm.SourceCharset** property.

XML Assembler Pipeline Component

The XML Assembler pipeline component combines XML serializing and assembling in one component. Its primary function is to transfer properties from the message context back into envelopes and documents.

The following actions occur in the XML Assembler component after receiving a batch of messages to form an interchange:

1. The assembler creates the envelope by using the specified envelope specification.

2. The component puts the content properties on the message instances by using the predefined XPaths coded as annotations in the XSD schemas associated with the message.

3. The component appends the message to the envelope.

4. The component puts the content properties on the envelope by using the predefined XPaths coded as annotations in the XSD schemas associated with envelopes.

For information about configuring the XML Assembler pipeline component, see Configuring the XML Assembler Pipeline Component.

In This Section

- Character Encoding in the XML Assembler Pipeline Component
- Processing Instructions in the XML Assembler Pipeline Component
- Unrecognized Messages in the XML Assembler Pipeline Component
- XML Information Set Elements in the XML Assembler Pipeline Component
- Document Structure Enforcement in the XML Assembler Pipeline Component

Character Encoding in the XML Assembler Pipeline Component

The XML Assembler pipeline component can produce messages in user-specified character encoding in two ways, as shown in the following table.

Encoding level	**Encoding method**
Component	Set the **Target charset** component property in Pipeline Designer.
Message	Set the **XMLNorm.TargetCharset** property on the message context. **Note** A message context property always overrides any context property set in Pipeline Designer.

The XML Assembler uses the following algorithm to determine output message encoding:

1. If the **XMLNorm.TargetCharset** context property is set, its value is used.

2. Otherwise, if the **Target charset** property is specified in Pipeline Designer, its value is used.

3. Otherwise, if the **XMLNorm.SourceCharset** property is specified, its value is used.

4. If none of the preceding properties is set, UTF-8 encoding is used.

The XML Assembler saves the encoding information of a BizTalk message object in the **IBaseMessagePart.Charset** property. When using Unicode or UTF-8 encoding, the XML Assembler always adds the byte order mark (BOM) to outgoing messages.

Note that when using the default XML send pipeline, which contains the XML Assembler component, the produced documents may be encoded by using the same charset as when they were submitted into the server, or they may be encoded by using UTF-8 if documents were created within the server and **XMLNorm.TargetCharset** was not specified.

Processing Instructions in the XML Assembler Pipeline Component

Processing instructions provide information to the application that processes an XML document. Such information may include instructions about how to process the document, how to display it, and so on.

Processing instructions are added to an XML document by the **Add processing instructions** property (or the equivalent **XMLNorm.ProcessingInstructionOption** property on the message context). Processing instruction text is specified with the **Add processing instructions text** property (or the equivalent **XMLNorm.ProcessingInstruction** property on the message context).

The **Add processing instructions** property (or **XMLNorm.ProcessingInstructionOption** property) has three possible values, which are described in the following table.

Value	Value	Description
Append	0	New processing instructions from the XML Assembler are appended to the processing instructions at the beginning of the document.
Create new	1	New processing instructions from the XML Assembler overwrite existing processing instructions at the beginning of the document.
Ignore	2	Processing instructions at the beginning of the document are removed.

The pair of processing instructions (or message context properties) specified on a message context take precedence over the property pair specified in Pipeline Designer. For example, if **XMLNorm.ProcessingInstructionOption** is specified as **Create new** (1) and **XMLNorm.ProcessingInstruction** is not specified, an empty processing instruction will replace an existing processing instruction.

As another example, if **XMLNorm.ProcessingInstruction** is specified but **XMLNorm.ProcessingInstructionOption** is not, none of the properties from the message context are used. In this case, the processing instructions from Pipeline Designer are used.

By default, **Add processing instructions** is set to **Append**, and **Add processing instructions text** is empty.

Processing Properties and Envelopes

Because processing instructions are not preserved for the envelopes, the following combination of flat file assembler settings results in only the outermost envelope having the processing instruction:

- **Processing instruction scope** property set to "Envelope."
- **Add processing instructions** property set to "Append."

The envelope would use the processing instruction specified in the assembler's **Add Processing instructions text** property.

Any existing processing instructions in either the outer or inner envelope(s), as specified in the incoming message, will not be present in the output message(s).

Unrecognized Messages in the XML Assembler Pipeline Component

The XML Assembler component treats a message as "unrecognized" if a message has:

- No body part.
- An empty body part.
- No data in the body part.
- No associated schema deployed.

The manner in which the XML Assembler handles an unrecognized message is controlled by the **XMLNorm.AllowUnrecognizedMessage** message context property.

When **XMLNorm.AllowUnrecognizedMessage** is set to **True**, the XML Assembler handles XML documents as follows:

- A message with no body part or with an empty body part or empty data in the body part passes unchanged through the assembler.

- A document that does not have a deployed schema associated with it passes unchanged through the assembler.

- A document with an associated deployed schema is processed by the assembler (regardless of whether the schema is explicitly referenced in a component property or found during the schema resolution process).

If **XMLNorm.AllowUnrecognizedMessage** is set to **False**, the XML Assembler handles XML documents as follows:

- A message with no body part or with an empty body part or empty data in the body part is not processed. An error is reported and the message is suspended.

- A message that does not have a deployed schema associated with it is not processed. An error is reported and the message is suspended.

- A document with an associated deployed schema is processed by the assembler (regardless of whether the schema is explicitly referenced in a component property or found during the schema resolution process).

- By default, the XML Assembler component does not allow unrecognized messages (that is, **XMLNorm.AllowUnrecognizedMessages** is considered **False** if it is not set on the message context).

XML Information Set Elements in the XML Assembler Pipeline Component

The XML Assembler component handles XML information set elements as follows.

Element	**Description**
comment	Comments are preserved between opening and closing XML tags in the document.
CDATA	CDATA is preserved between opening and closing XML tags in the document. CDATA values are not used for property promotion or distinguished fields.
processing instructions	Processing instructions located before the document XML tag are handled based on their value (for more information, see Processing Instructions in the XML Assembler Pipeline Component). Processing instructions between document open and close tags are preserved. Processing instructions after the closing XML tag are ignored, as are any instructions before, in, or after the envelope.
XML declaration	The XML declaration string, such as <?xml version='1.0'? encoding='UTF-8'>, may be preserved by the XML Assembler pipeline component. This is controlled by the **Add XML declaration** property, or its equivalent on the message context,

	XMLNorm.AddXMLDeclaration. If this option is set to **True** then the XML declaration will be added to the document. If the XML declaration already existed, it will be overwritten. If this option is set to **False**, no XML declaration will be added and any existing XML declaration will be removed. The value of this property in the message context takes precedence over the value specified in Pipeline Designer. Default value: **True** (the XML declaration is always added)

Document Structure Enforcement in the XML Assembler Pipeline Component

If document or envelope schemas are explicitly referenced in the XML Assembler, the XML Assembler ensures that only documents with the message type corresponding to referenced schemas are processed. All the other documents are not processed even though a schema may be deployed for them.

XML Disassembler Pipeline Component

The XML Disassembler pipeline component combines XML parsing and disassembling into one component. Its primary functions are:

- Removing envelopes.
- Disassembling the interchange.
- Promoting the content properties from interchange and individual document levels on to the message context.

The following actions occur in the XML Disassembler component after receiving an envelope:

1. The disassembler parses the envelope by using the envelope schemas statically associated with the component at design time or dynamically by determining envelope schemas from the message type at run time. The schema is used to verify the structure of the envelope during envelope parsing. If envelope structure is not defined, it is found recursively by using the root node's namespace and base name to look up the schemas.
2. The disassembler component parses each document within the envelope. For each document, the BizTalk message object is created with its own context where all the properties promoted from the envelope and from the document itself get copied. The component pulls the content properties from the envelope

and message instances by using the predefined XPaths coded as annotations in the XSD schemas associated with the envelope and message. The envelope schemas as well as the individual document schemas are associated with the disassembler component in Pipeline Designer.

The XML Disassembler only processes data in the body part of the message. Thus, only properties from body part can be promoted. Datetime values from the fields associated with the promotable properties get converted to UTC when property promotion occurs. Non-body parts are copied to the output message unchanged.

For information about configuring the XML Disassembler pipeline component, see How to Configure the XML Disassembler Pipeline Component .

In This Section

- XML Information Set Elements in the XML Disassembler Pipeline Component
- Unrecognized Messages in the XML Disassembler Pipeline Component
- Document Validation in the XML Disassembler Pipeline Component
- Document Structure Enforcement in the XML Disassembler Pipeline Component
- Character Encoding in XML Disassembler Pipeline Component

XML Information Set Elements in the XML Disassembler Pipeline Component

The XML Disassembler handles XML information set elements as follows.

Element	**Description**
comment	Comments are preserved in the document; however, any comments in XML envelopes are not preserved.
CDATA	CDATA is preserved in the document. CDATA elements appearing outside of a document (for example, in an envelope) are not preserved.
processing instructions	The XML Disassembler preserves processing instructions appearing in or before an XML document. Processing instructions appearing after an XML document or before or after an envelope are not preserved.
XML declaration	The XML declaration element of any incoming XML message is removed.

Unrecognized Messages in the XML Disassembler Pipeline Component

The XML Disassembler component may handle a message as "unrecognized" in the following cases:

- An XML message is received with no body, an empty body, or empty data in the body.
- An XML message is received but no schema is deployed for it.

An unrecognized message is handled based on the **Allow Unrecognized Messages** property (or on the **XMLNorm.AllowUnrecognizedMessage** property on the message context).

If **Allow Unrecognized Messages** is set to **True**, the following occurs:

- A message with no body, an empty/null body, or with empty/null data in the body passes unchanged through the XML Disassembler.
- An XML document with no associated deployed schema passes unchanged through the XML Disassembler.
- An XML document that has a deployed schema associated with it is processed by the XML Disassembler regardless of whether the schema is explicitly referenced in a component property or found during the schema resolution process.

If **Allow Unrecognized Messages** is set to **False**, the following occurs:

- A message with no body or an empty/null body or with empty/null data in the body is not passed through the XML Disassembler.
- An XML document that does not have a deployed schema associated with it is not passed through the disassembler. An error is reported and the message is suspended, if possible.
- An XML document that has a deployed schema associated with it is processed by the XML Disassembler regardless of whether the schema is explicitly referenced in a component property or found during the schema resolution process.

By default, the XML Disassembler does not allow unrecognized messages.

Document Validation in the XML Disassembler Pipeline Component

By default, the XML Disassembler does not validate XML documents against a schema. However, you can configure the XML Disassembler to validate an XML document by setting the **Validate Document Structure** property.

Document Structure Enforcement in the XML Disassembler Pipeline Component

If a document or envelope schema is explicitly referenced in the XML Disassembler, the XML Disassembler processes only those documents with message types conforming to the schema. No other documents are processed, regardless of whether a deployed schema is referenced for them.

The XML Disassembler enforces the specified order of envelope schemas; however, the order of document schemas is not enforced.

Character Encoding in XML Disassembler Pipeline Component

The XML Disassembler uses the following algorithm to determine which encoding to use for processing incoming messages:

1. If a byte order mark exists in the data, encoding information is determined from it.

2. Otherwise, if the **IBaseMessagePart.Charset** property is set, the encoding specified there is used.

3. Otherwise if the XML declaration is present in the XML document, the encoding specified there is used, provided the XML declaration is ANSI.

4. Otherwise, UTF-8 encoding is used.

For the preceding cases 2, 3, and 4, after the XML Disassembler determines the encoding, it saves it on the message context in **XMLNorm.SourceCharset** property. Messages produced by the XML Disassembler pipeline component always use UTF-8 encoding. For case 1, encoding determined from the byte order mark is not preserved.

Walkthrough: Using XML Envelopes (Basic)

This example demonstrates basic XML envelope disassembly by implementing part of a fictitious error tracking system. The example will meet the following requirements:

1. Error messages are logged at various physical sites within the company and sent to a central location for processing into various back-end systems.

2. Error messages are written in XML format.

3. An error message can be sent singly without an envelope or as a batch contained within an envelope.

Prerequisites

For this example you need to be comfortable with creating BizTalk Server projects, signing an assembly, and using the BizTalk Server Management Console to view applications and ports. You should also be comfortable with the ideas presented in Walkthrough: Deploying a Basic BizTalk Application.

What This Example Does

The example processes inbound messages containing either a single Error message or a batch of Error messages by defining an envelope schema and using the XmlDisassembler pipeline.

Example

To create the example, follow the steps outlined in the sections below.

Create a New BizTalk Project

Before building a solution we need to create a BizTalk Server project, ensure that it is strongly named, and assign it an application name. Assigning an application name will keep BizTalk from deploying the solution into the default BizTalk Server application.

To create and configure a new BizTalk Server project

1. Create a new BizTalk Server project using Visual Studio 2005. Call the project **BasicXMLEnvelope**.

2. Generate a key file and assign it to the project. See **How to Configure a Strong Name Assembly Key File** for more information on this task.

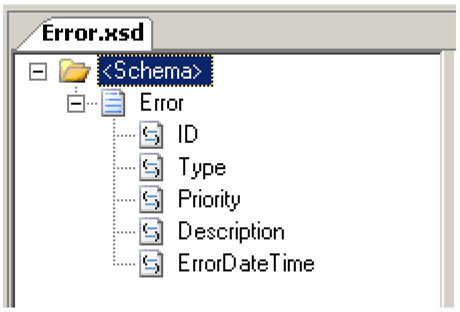
3. In the deployment configuration properties for the project, assign an **Application Name** and set **Restart Host Instances** to True. By setting this flag, the host will clear any cached instances of the assembly.

Create the Error Schema

In this step you will create the Error schema. It defines the key message for the system.

To create the Error schema

1. Add a new schema named "Error" to the project.
2. Change the target namespace for the schema to http://BasicXMLEnvelope.
3. Change the schema property **Form ElementDefault** under the **Advanced** category to **Qualified**.
4. Rename the root node "Error" and create 5 child elements with the types indicated:



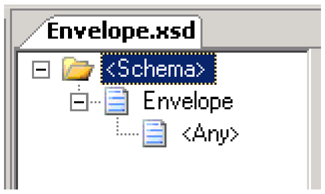
1. Create a sample message for this schema. This will be used to verify that single messages outside of an envelope will be processed properly. An example sample message is below.

Create the Envelope Schema

The envelope will contain one or more Error messages. In this basic example, the envelope will not have properties and elements of its own.

To create the envelope schema

1. Add a new schema named "Envelope" to the BasicXMLEnvelope project.
2. Change the target namespace to http://BasicXMLEnvelope.
3. Change the name of the root node from "Root" to "Envelope".
4. Now mark the schema as an envelope schema. Click on the **<Schema>** node. In the properties pane, set the schema reference property **Envelope** to **True**.
5. Set the **Body XPath** property. To do this, click on the "Envelope" node. In the properties pane, click on the ellipses for the Body XPath property, select "Envelope", then click **OK**.
6. Add an Any element child to the Envelope node. The Error message will be contained in this element. Your schema should look like the following:



1. Create a sample message containing an envelope and one or more sample messages. An example message is below.

Deploy and Configure the Send and Receive Ports

With the schemas created, we need to compile and deploy the project. Once it is deployed, the send a receive ports can be configured using the BizTalk Server Administration Console.

To Deploy BasicXMLEnvelope

1. Build BasicXMLEnvelope. Choose **Build | Rebuild BasicXMLEnvelope**.

2. Deploy the solution using Visual Studio 2005 by clicking **Build | Deploy BasicXMLEnvelope**.

3. Using the BizTalk Server Administration Console, expand the Applications group to verify that BasicXMLEnvelope is present as a custom application. This console will be used for configuring ports.

To configure the receive port

1. Use Windows Explorer to create a directory named "Receive" under the BasicXMLEnvelope project directory.

2. From the BizTalk Server Administration Console, expand the BasicXMLEnvelope application, right click on **Receive Ports** and choose **New | One-way Receive Port**.

3. In the Receive Port Properties dialog, set the name of the port to "Receive".

4. Click Receive Locations, then click **New** to add a receive port. Name the new port "ReceiveError". Set the **Receive Pipeline** to **XMLReceive**. For **Transport Type**, choose **FILE** then click **Configure**. Choose the receive directory you created

5. Click **OK**. Your receive port should now be configured. Click **OK** to close.

To configure the send port

1. Use Windows Explorer to create a directory named "Send" under the BasicXMLEnvelope project directory.

2. From the BizTalk Server Administration Console, expand the BasicXMLEnvelope application, right click on **Receive Ports** and choose **New | Static One-Way**.

3. In the Send Port Properties dialog, set the name of the port to "Send".

4. For transport type, select **FILE** and then click **Configure**. Set the destination folder to the send directory you created earlier. Click **OK** to close.

5. Since the example uses the **PassThruTransmit** send pipeline, you can click **OK** to complete the send port configuration. Your send port should be configured.

Running the Example

It is now time to run the example. After starting the BasicXMLEnvelope application using BizTalk Server Management Console, you will copy the test files to the receive location and observe what is produced in the send location.

To run the BasicXMLEnvelope example

1. In the BizTalk Server Administration Console, right-click the **BasicXMLEnvelope** application and choose **Start**. This will enlist and start the send and receive ports.

2. Drop each of the sample files into the receive directory. If the samples given above are used, you should find three indivudual Error messages in the send location when processing is completed.

Extending the Example

The example can be extended to accommodate other requirements. Two common scenarios will be addressed in this section:

- If a batch of errors is submitted within an envelope, individual message failures in disassembly should not prohibit other non-failing messages from being further processed.

- Errors should be delivered to different locations based on error priority. High priority messages will be expedited while other priorities will be handled through normal channels.

The sections below extend the example to handle each of these requirements.

Recoverable Interchange Processing

BizTalk Server 2006 supports recoverable interchange processing. By using this feature, you can ensure that batches of messages fail individually in disassembly and not as a batch.

To configure the example for Recoverable Interchange Processing

1. From the BizTalk Server Administration Console, expand the **BasicXMLEnvelope** application, click on **Receive Ports**, then double-click on the Receive port. This is the port you created previously.

2. In the Receive Port Properties dialog, click **Receive Locations**. Click **Properties** to bring up the ReceiveError Receive Location Properties dialog. Click the ellipses button for the **Receive Pipeline**.

3. In the Configure Pipeline - XMLReceive dialog, set the **Recoverable Interchange Processing** property to **True**, then click **OK**.

4. Click **OK** to close the Receive Location Properties dialog, then click **OK** to close the Receive Port Properties dialog.

To create a sample file and run the example

1. To create a sample file, make a copy of the envelope sample file created in the example and add a non-existent namespace to one of the Error instances, then save the file:

2. In the BizTalk Server Management Console, click on the Applications and verify that the BasicXMLEnvelope is running.

3. Drop the message in the receive location. After processing, you should find the first message in the send location and the second, low priority message in the suspend queue.

Content-Based Routing (CBR)

Content-based routing can be used to route messages based on the content of the message. In this scenario, routing will be based on Priority with High messages going to one send location and Low and Medium going to a different send location.

To extend the sample, the following tasks must be completed:

1. Promote the Priority field in the Error schema in the BasicXMLEnvelope project. Content-Based Routing relies on promoted properties to route messages. See Promoting Properties for more details.

2. Create and configure two additional send ports. The ports will use a filter to ensure they receive appropriate messages.

To promote the Priority field in the Error schema

1. In the BasicXMLEnvelope project, open the Error schema and expand the Error node.

2. Right-click on the Priority element then choose **Promote | Quick Promote**.

3. Click OK to confirm the addition of a new property schema for the promoted properties.

4. In the Visual Studio solution explorer, open the new property schema PropertySchema.xsd. Remove "Field1" from the schema.

5. Now recompile and redeploy the solution. To recompile, choose Build | Rebuild BasicXMLEnvelope. To redeploy, choose Build | Deploy BasicXMLEnvelope. The project was configured to reset the host instance on redeploy; if you have changed this you will need to stop and start the host.

To configure the low / medium priority send port

1. Use Windows Explorer to create a directory named "SendLowMediumPriority" under the BasicXMLEnvelope project directory.

2. From the BizTalk Server Administration Console, expand the BasicXMLEnvelope application, right click on **Receive Ports** and choose **New | Static One-Way**.

3. In the Send Port Properties dialog, set the name of the port to "SendLowMediumPriority".

4. For transport type, select **FILE** and then click **Configure**. Set the destination folder to the directory you created earlier. Click **OK** to close.

5. Click Filters and add three filter expressions:

- BTS.MessageType == http://BasicXMLEnvelope#Error And
- BasicXMLEnvelope.PropertySchema.Priority == Low Or
- BasicXMLEnvelope.PropertySchema.Priority == Medium

6. Click **OK** to complete the low and medium priority send port configuration.

To configure the high priority send port

1. Use Windows Explorer to create a directory named "SendLowMediumPriority" under the BasicXMLEnvelope project directory.

2. From the BizTalk Server Administration Console, expand the BasicXMLEnvelope application, right click on **Receive Ports** and choose **New | Static One-Way**.

3. In the Send Port Properties dialog, set the name of the port to "SendLowMediumPriority".

4. For transport type, select **FILE** and then click **Configure**. Set the destination folder to the directory you created earlier. Click **OK** to close.

5. Click Filters and add three filter expressions:

- BTS.MessageType == http://BasicXMLEnvelope#Error And
- BasicXMLEnvelope.PropertySchema.Priority == High

6. Click **OK** to complete the high priority send port configuration.

To test the routing solution

1. Using the BizTalk Server Administration Console, expand the Applications group, then right-click on the BasicXMLEnvelope application and choose **Start**. When prompted to confirm, click **Start**. This will enlist the new send ports.

2. Drop the different test message into the receive location. Notice how Error messages are routed to the different send locations:

- Error messages that have a Low, Medium or High priority and do not fail message processing are routed both to the original send location (configured in the core example) and the send location by priority. For a single message with a Low priority, a copy will end up in the original send location and the Low / Medium send location.

- If Recoverable Interchange Processing is enabled, the failed Error message is not routed and the non-failed message is properly routed as expected. The failed message is not routed because its message type does not match the type used in the configured filters.

MIME/SMIME Decoder Pipeline Component

The MIME/SMIME Decoder component provides MIME decoding functionality for messages. This pipeline component can be placed into the Decode stage of a receive pipeline, and it supports 7bit, 8bit, binary, quoted-printable, UUEncode, and base64 decoding. Localized data character set changes will not affect the decoding.

The MIME/SMIME Decoder component can decrypt and sign-validate an incoming message. Decryption certificates are used from the personal certificate store of the current user under which the service is running. Sign-validation certificates are used from the Address Book store of the local computer or from the message itself.

On successful decryption of a message, the MIME/SMIME Decoder pipeline component associates the thumbprint of the decryption certificate used to decrypt the message as a predicate of the message. This means that any service (orchestration or send port) that is subscribing to that message must be associated with a host that owns that key. Associations between hosts and keys can be completed in the BizTalk Administration console as a property of the host. For more information about configuring the host in the BizTalk Administration console, see Modifying Host Properties in the BizTalk Administration Console.

The MIME/SMIME Decoder pipeline component is the only out-of-the-box receive pipeline component that handles multi-part messages, including multi-part MIME/SMIME messages. The pipeline component parses the message and creates an equivalent multi-part BizTalk message. A multi-part BizTalk message has one unique part named the body part. All other pipeline components, such as the XML Disassembler pipeline component, only process the body part of the BizTalk Message. Also the MessageType corresponding to the BizTalk body part is used for routing the message to subscribers.

The following conditions are evaluated by the MIME/SMIME Decoder pipeline component to identify the BizTalk BodyPart corresponding to a multi-part MIME message. The order of the condition evaluation is as follows:

1. The first MIME/SMIME part that has the Content-Description header set to "body" (case-insensitive).

2. The first MIME/SMIME part that has the Content-Type header set to "text/xml"(case-insensitive).

3. The first MIME/SMIME part that has the Content-Type header set to "text/" (case-insensitive).

4. The first MIME/SMIME part.

For information about configuring the MIME/SMIME Decoder pipeline component, see Configuring the MIME/SMIME Decoder Pipeline Component. For more information about BizTalk Server support for decryption, sign-validation, and usage of certificates, see **Sending and Receiving Secure Messages**.

MIME/SMIME Encoder Pipeline Component

The MIME/SMIME Encoder component can be placed into the Encode stage of a send pipeline. It supports 7bit, 8bit, binary, quoted-printable, base64, and UUencode encoding. Localized data character set changes do not affect the encoding.

This component can be used to either MIME encode, sign or encrypt an outgoing message with encryption and signing certificates.

If there is an encoding component in the send pipeline that is configured to sign messages, and the BizTalk group that includes the host running the pipeline is not configured with a signing certificate, the outgoing message will be suspended (with the appropriate error) to the suspended queue of the host that is running the pipeline. If the signing certificate cannot be found in the personal certificate store of the current user under which the service is running, the message will be suspended to the suspended queue of the host that is running the pipeline.

If there is an encoding component in the outbound pipeline configured to encrypt outbound messages, and the certificate cannot be found in the certificate store, the message will be suspended to the suspended queue of the host that is running the pipeline.

To perform response encryption on a request-response port that is receiving signed and encrypted messages, a custom pipeline component must be added to the pipeline before the MIME/SMIME Encoder pipeline component. This custom pipeline component must identify the thumbprint corresponding to the encryption certificate and set it to the **BTS.EncryptionCert** property on the message context. For more information about message context properties, see Modifying BizTalk Group Properties.

For information about configuring the MIME/SMIME Encoder pipeline component, see Configuring the MIME/SMIME Encoder Pipeline Component. For more information about BizTalk Server support for encryption, signing, and usage of certificates, see **Sending and Receiving Secure Messages**.

Party Resolution Pipeline Component

The responsibility of the Party Resolution pipeline component is to map the sender certificate or the sender security identifier (SID) to the corresponding configured BizTalk Server party.

When the Party Resolution component reads the incoming message, it takes two message context properties as input: **WindowsUser** and **SignatureCertificate**. The **WindowsUser** property is populated by the adapter, or by a custom pipeline component, with the user name of the sender when it can reliably derive the sender information. The **SignatureCertificate** is populated by the adapter or the MIME/SMIME Decoder pipeline component with the thumbprint of the client authentication certificate.

If the message is signed, the thumbprint of the certificate that was used to validate the signature on the inbound message is then used to look in the Configuration Repository to determine which party it is associated with. If a party is found, the SourcePartyID for that party is placed in the context of the message as the originator of the message.

To enable the Party Resolution pipeline component to validate a Windows user, you must add the WindowsUser alias to a party. The Name and Qualifier should be set to WindowsUser and Value should be set to <domain\user name>. In a stand-alone scenario, the WindowsUser value used to configure the party should match the value that is set by the receive adapter. For more information about aliases, see Partner Management in BizTalk Explorer. For more information about adding an alias to a party, see Adding an Alias to a Party Using BizTalk Explorer.

If the message arrives at the Party Resolution component with both of the properties stamped, the Party Resolution component first tries to resolve the party by the certificate (assuming the **Resolve Party By Certificate** property is set to **True**). If the party is resolved, the SourcePartyID for that party is placed in the context of the message as the OriginatorPID of the message if the host process running the pipeline is marked as **Authentication Trusted** by the pipeline. If the party resolution cannot be completed by using the certificate, the OriginatorPID value on the message is stamped with "s-1-5-7", which is the SID of an anonymous user. For more information about the OriginatorPID property, see Securing Pipelines.

The following table shows how the Party Resolution pipeline component attempts to resolve the party.

By SID	**By certificate**	**WindowsUser**	**SignatureCertificate**	**Result**
True	False	Available	Available or unavailable	Party is resolved.
True	False	Unavailable	Available or unavailable	Party is not resolved and is stamped as anonymous.
False	True	Available or unavailable	Available or unavailable	Party is not resolved and is stamped as anonymous.
True	True	Available	Available	Party is resolved.
True	True	Unavailable	Available or unavailable	Party is not resolved and is stamped as anonymous.
False	False	Available or unavailable	Available or unavailable	Party is not resolved and is stamped as anonymous.

For information about configuring the Party Resolution pipeline component, see Configuring the Party Resolution Pipeline Component.

XML Validator Pipeline Component

The XML Validator pipeline component can be used in both send and receive pipelines in any stage except for Disassemble or Assemble. The XML Validator component validates the message against the specified schema or schemas, and if the message does not conform to these schemas, the component raises an error and Messaging Engine places the message in the suspended queue.

For information about configuring the XML Validator pipeline component, see Configuring the XML Validator Pipeline Component.

Schema Resolution in Pipeline Components

Pipeline disassembler and assembler components use XSD schemas to process messages. The schemas contain information such as the list of promoted properties, distinguished fields, annotations for flat file messages, and annotations for XML envelopes.

Standard disassembler and assembler components support retrieval of deployed schemas by using the schema type name and message type. Some components retrieve by using both the schema type name and the message type, while others (for example, the Flat File Disassembler) retrieve only by the schema type.

Pipeline components that receive XML messages determine the message type by examining the message root element and namespace. For example, the message type for the following XML is "http://MyDocument.org#MyDocument".

If a schema does not have a namespace defined for it, the message type is "<**rootNode**>". For example, if the preceding example XML had no namespace, the message type would be "MyDocument".

Standard pipeline components use the message type to retrieve the appropriate schema from the database. Default XML receive and send pipelines always determine which schema to load by using the message type dynamically discovered at runtime from the message XML content (unless the pipeline component is set to allow unrecognized messages). The XML Disassembler can remove the message envelope by using this mechanism; however, the XML Assembler cannot create an envelope for an outgoing message without knowing what envelope schema to use.

You specify the envelope schema in the configuration properties for the XML Assembler from within the pipeline designer.

How Schemas Are Resolved

Assuming you are not specifying the schema directly in the XmlDisassembler, schemas will be resolved in the following manner:

1. First, an unqualified search on the deployed schemas is made using the root node name and namespace (for example, http://MyNamespace#MyRoothttp://MyNamespace#MyRoot). If a unique match is found the schema is resolved. If multiple matches are found and differ only by version number and only one is active, that version is used and the schema is resolved. If the same schema is active in multiple applications, multiple active schemas will be found and the search will continue with step 2 below.

2. If there are multiple matches that step 1 can not resolve, the search is qualified by the assembly the pipeline is executing in. If a unique schema is found within the same assembly the pipeline is executing in, then the schema is resolved.

3. If there are still no matches, then the publisher is used to resolve the schema. The search is narrowed by using the root node, namespace, and public key token. If a unique schema is found using this search, the schema is resolved.

4. Schema cannot be resolved. An error is returned noting that no unique schema can be found.

See Namespace Management for other considerations including the effect of SQL Server collation and case-sensitivity on schema resolution.

To create an envelope in the XML Assembler or a header and trailer in the Flat File Assembler, you may do one of the following:

- Create a custom send pipeline and specify the schemas for the envelope in the configuration properties for the XML Assembler. This is done from within the pipeline designer.

- In the BizTalk Server Administration console specify XMLTransmit for the Send pipeline property on a send port. Click the ellipsis to configure the pipeline properties and specify the schema for the envelope in the EnvelopeDocSpecNames property.

Schema resolution by message type may not work if several versions of the same schema are intentionally or accidentally deployed in the database (for example, in a side-by-side deployment or if multiple scenarios do not have unique message types). If schema resolution by message type fails, a "schema ambiguity" error is added to the event log. To ensure that schema resolution by message type succeeds, do one of the following:

- Define the schemas in the same BizTalk project as your custom pipeline.

- Sign the assembly with the schemas with the same key as the assembly containing the pipelines.

- Explicitly specify schemas in pipeline components (message type names should be unique across the BizTalk Management database).

Envelope Use in the XML Assembler and Disassembler Pipeline Components

An XML message can include zero or more envelopes. The following example shows an envelope (in bold) wrapping an XML document.

Envelopes serve two purposes:

- They can include field values to use for property promotion and demotion.

The XML Disassembler component promotes properties, and the XML Assembler component demotes properties. Property promotion and demotion can also occur in XML documents.

- They can combine several XML documents into a single interchange.

  Because a well-formed XML document can have only one root element, an envelope enables you to combine multiple XML documents to share one root element.

You can enforce the canonical form by specifying the envelope order by using the **Envelope Specification Names** design-time property in the XML Assembler, or with the **XMLNORM.EnvelopeSpecNames** message context property before the XML Assembler is run. The XML Assembler produces an enveloped document in canonical form.

Nesting envelopes

You can nest envelopes to form complex document structures where several enveloped XML documents can be combined into a larger interchange. The following example shows an interchange wrapped by two envelopes.

The preceding example illustrates a flexible form, which means that a document can be on the same hierarchy level as an envelope. After disassembling the enveloped document, four separate documents are created (document1, document2, and so on).

Property Demotion in Assembler Pipeline Components

You can use property demotion to copy a property value from the message context into the message content or to its header or trailer. You accomplish property demotion by using an XPath expression specified in the document or in the header and trailer schema.

When writing datetime data from the context property into the resulting document, BizTalk Server assumes that all datetime data is in UTC format.

The format used to write properties into the data is determined by the XSD data type as shown in the following table.

Data type	Format
xs:datetime	yyyy-MM-ddTHH:mm:ss.fffffffZ
xs:date	yyyy-MM-ddZ
xs:gDay	---ddZ

xs:gMonth	--MM—Z
xs:gMonthDay	--MM-ddZ
xs:gYear	yyyyZ
xs:gYearMonth	yyyy-MMZ
xs:time	HH:mm:ss.fffffffZ

Property Demotion and Envelopes

It is often useful to demote values from one or more of the system namespaces -- or one of your own namespaces -- when assembling files within an envelope. Some common scenarios include:

- You want to include the original file name submitted to the system in outbound messages so back-end systems can track the origin of data.
- You want to write data from the body message to the header. For example, for a purchase order it might be useful to write the ship to name to the envelope for down-stream systems.
- You want to combine many different fields into the header without writing custom code. Property demotion in the Xml assembler or flat file assembler can do the job.

It is important to remember that the XML and flat file assembler components both allow you to specify which schema to use for the envelope and document body. You can choose the same schemas used in disassembly or create a new envelope schema with different fields.

See EnvelopeProcessing (BizTalk Server Sample) for an example of these concepts.

Property Promotion in Disassembler Pipeline Components

Property promotion is a process by which a property value is extracted from an XML document by using an XPath expression and placed on the message context so that it can be used for message routing.

If a promoted property does not have a default or fixed value, the XML field for that property is missing, and the Validate Document Structure property is **False**, the property is not promoted.

A custom pipeline component can promote multivalued (that is, arrayed) properties. Messages that contain multivalued properties are only supported in content-based routing (CBR) scenarios; they cannot be routed to orchestrations or be used for tracking purposes.

The XML Disassembler does not promote default or fixed values for an empty element if it has a closing tag. For example, <field1> is not promoted in the following XML.

However, an empty element without a closing tag (as shown in the following example) is promoted.

When reading datetime data from a document and placing it into the context property, if the data is in UTC format, that format is preserved. If the datetime data is in local+offset format, BizTalk Server converts the datetime format to the UTC format that results from adding the offset to the local time. If the datetime format does not specify time zone or UTC format, the time is assumed to be local and is converted to UTC based on the current time zone.

Distinguished Fields in Disassembler Pipeline Components

Distinguished fields defined in a schema are written to the message context by the XML Disassembler, BizTalk Framework Disassembler, or Flat File Disassembler pipeline components in the following format:

name used is the distinguished field in XPath

namespace URI is "http://schemas.microsoft.com/BizTalk/2003/btsDistinguishedFields"

The value of the property is the **System.String** value extracted from the XML document using specified XPath.

The following example schema has a distinguished field Price.

For the document instance

the XML Disassembler writes a distinguished field on a message context as follows:

Name of the property on the context: "/*[local-name()='PO' and namespace-uri()='http://SendHtmlMessage.PO']/*[local-name()='Price' and namespace-uri()='']"

Namespace of the property: http://schemas.microsoft.com/BizTalk/2003/btsDistinguishedFields

Value of the property: 10

Using Pipeline Designer

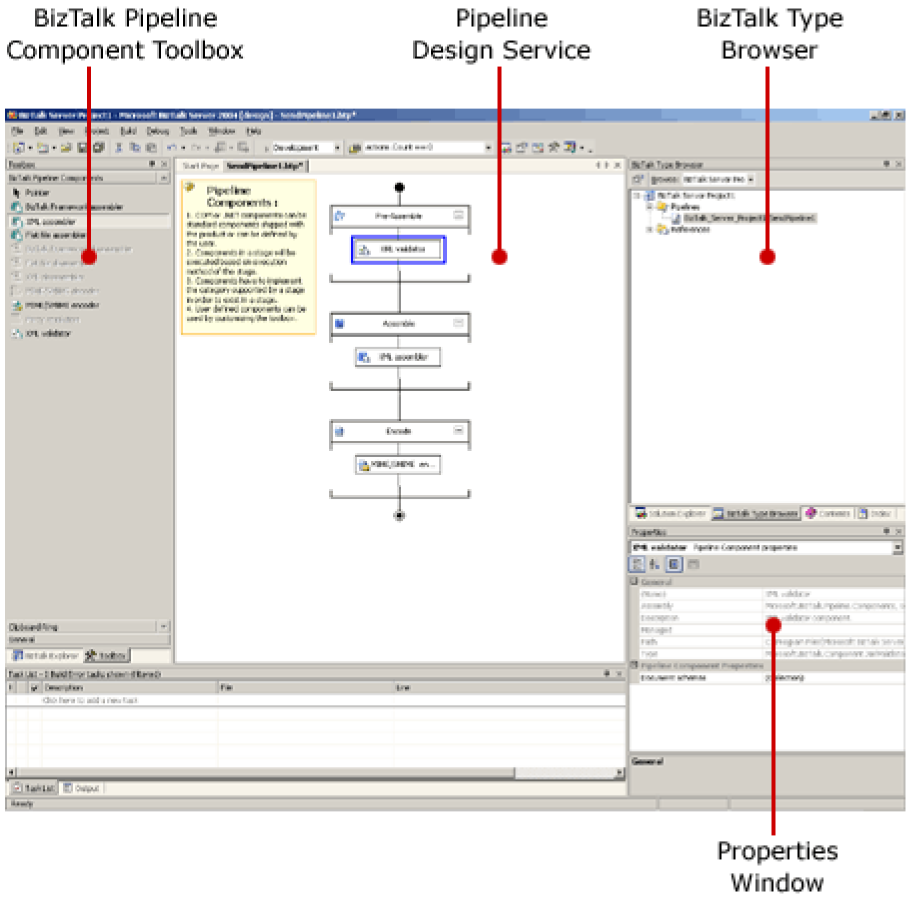
Pipeline Designer is a graphical editor, hosted in Microsoft® Visual Studio® .NET, which enables you to create new pipelines; view the pipeline templates included with Microsoft® BizTalk® Server; move pipeline components within a pipeline; and configure pipelines, stages, and pipeline components.

Pipeline Designer uses three key tools of the Visual Studio shell as part of the design experience:

- The Properties window, where most of the characteristics of pipeline objects are viewed and modified.
- The Toolbox, which is used as a source for the design surface.
- The design surface, where components from the Toolbox are dragged and dropped.

The following figure shows the Pipeline Designer environment.

Depicts the Pipeline Designer environment.



Pipeline Designer is integrated with the BizTalk project template to enhance your development experience. After using the project system to create a new BizTalk project, you can use the **Add New Item** command on the **File** menu to add a pipeline to your solution. For more information about the BizTalk project template, see Using the BizTalk Project System.

The Pipeline Designer design surface enables you to draw a graphical representation of a pipeline. The design surface occupies the main section of the Visual Studio .NET window and enables you to edit the pipelines belonging to a BizTalk project. You can navigate between pipelines by clicking the tabs above the design surface.

Each pipeline is composed of stages, with each stage containing one or more components. If there are no components in a stage, a watermark with text indicates that shapes can be inserted from the Toolbox. When the first shape is inserted into a stage, the initial text disappears. The design surface shows the pipeline vertically, running from top (start) to bottom (end).

As with other common Microsoft® Windows® programs, you can perform several tasks, such as **Open** and **Save** from the **File** menu.

Creating Pipelines with Pipeline Designer

This section describes how to:

- Create, open, and save pipelines
- Add components to a pipeline
- Use the Schema Collection Property Editor
- Use the Toolbox
- Navigate with the keyboard
- Work with read-only pipeline component properties.

For information about using Pipeline Designer, see Using Pipeline Designer .

In This Section

How to Create a New Pipeline

You can add a pipeline template to your project to create a new pipeline.

To create a new pipeline

1. In Solution Explorer, select the project in which you want to create the pipeline.
2. On the **File** menu, click **Add New Item**.
3. In the **Add New Item** dialog box, select a **Receive Pipeline** or **Send Pipeline** template by clicking it once.
4. In the **Name** field, type a name for the pipeline.
5. Click **Open**.

   The new pipeline appears in Solution Explorer. The design surface displays pipeline stages and a default set of components.

.How to Open a Pipeline

Opening a pipeline file causes Pipeline Designer to appear with the defined pipeline and its stages in the design area. In addition, the Pipeline Designer Toolbox and the Properties window will appear if they are not already open.

To open a pipeline

1. In Solution Explorer, double-click the pipeline you want to open.

   —Or—

2. In Solution Explorer, select the pipeline, and then on the **View** menu, click **Open**.

How to Add a Component to a Pipeline

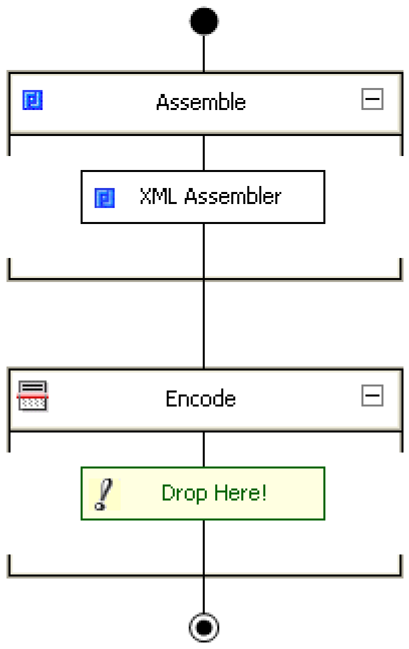
You add components to pipelines by dragging from the Toolbox to the design surface.

To add a component to a pipeline

1. In the Toolbox, drag the pipeline component onto a **Drop Here!** box on the design surface.

The following illustration shows how the pipeline design surface illustrates pipelines. This pipeline has two stages, the Assemble stage and the Encode stage. The XML Assembler pipeline component was added to the Assemble stage, but the Encode stage is still empty, because it still shows **Drop Here!** to indicate that a pipeline component can be added to the stage.

Illustrates stages and components in a BizTalk pipeline.



How to Set Pipeline Component Properties

Each component has properties that should be configured prior to use.

To set pipeline component properties

1. In the Properties window, in the **Pipeline Component Properties** section, click the **Value** cell in the **Property** row, and enter the new value. If the **Value** cell contains an arrow button, click the button for a list of choices. If the **Value** cell contains an ellipsis () button, click the button to use the Schema Collection Property Editor to configure the property. For more information, see Using the Schema Collection Property Editor.

How to Use the Schema Collection Property Editor

You use the Schema Collection Property Editor to select schemas for your pipeline component.

To use the Schema Collection Property Editor

1. In the Schema Collection Property Editor, in the **Available Schemas** box, select the schemas that should be available for the pipeline component. Use the CTRL key to select more than one schema.

2. Click **Add**.

   The selected schemas now appear in the **Added Schemas** box.

3. Click **OK**.

How to Save a Pipeline

Saving a pipeline stores information about the pipeline configuration to a .btp file, which specifies:

- The name of the pipeline.
- The template and policy that it is based on (send or receive).
- The sequence of components that are part of the pipeline, including the order and set of components used in each stage.
- The configuration information for each of the pipeline components.

To save a pipeline

1. On the **File** menu, click **Save**.

   —Or—

2. Display the pipeline on the design surface by clicking its tab near the top of the screen.
3. On the **File** menu, click **Save As**.
4. In the **Save File As** dialog box, in the **File** name field, type a name for the pipeline. A default name is supplied for you.
5. Click **OK**.

How to Use the Toolbox

You create a pipeline by dragging components (shapes) from the Toolbox to the design surface. Pipeline Designer helps you assemble valid pipelines by placing certain restrictions on the creation process. You can only select Toolbox components that apply to the pipeline type you are creating. For example, a receive pipeline will show decoders, disassemblers, and validators as valid Toolbox components, while encoders and assemblers will be disabled (dimmed).

In addition, stages can accept only valid components. For example, you cannot drag an encoder component to an Assemble stage. When you drag a component near a valid drop location, an arrow appears, indicating the point where the component can be inserted.

If you drag a valid component onto a stage that is collapsed, pause the mouse over the stage for a few seconds to expand it, and then drop the component into the stage.

Adding a custom component to the Toolbox in Pipeline Designer is similar to the standard Microsoft® Visual Studio® .NET procedure.

Start and End shapes

Start and **End** shapes appear as bullets on all pipelines. They are provided on the design surface for visual organization only. There is only one of each, and they can neither be added nor deleted. The **Start** and **End** shapes do not appear in the Toolbox.

To add a pipeline component to the Toolbox

1. On the **Tools** menu, click **Choose Toolbox Items**.

   —Or—

   Right-click the Toolbox and then click **Choose Items**.

2. In the **Customize Toolbox** dialog box, click the **BizTalk Pipeline Components** tab.

   A dialog box displays the compatible pipeline components. You can navigate through the categories by clicking the tabs at the top of the dialog box.

3. Select the component you want to add to the Toolbox.

4. Click **OK**. The component will appear on the **BizTalk Pipeline Components** tab of the Toolbox.

To remove a pipeline component from the Toolbox

1. Open the **Customize Toolbox** dialog box (as in the preceding procedure) and clear the component to be removed.

   A component remains registered even after it has been removed from the Toolbox.

How to Navigate with the Keyboard

You can navigate through the design surface by using your keyboard instead of your mouse. The following table shows the keys that you can use.

Key	Effect
DOWN ARROW	Moves the cursor to the next connecting line or shape below, including **Start** and **End** shapes.
UP ARROW	Moves the cursor to the previous connecting line or shape above, including **Start** and **End** shapes.
PAGE UP	Changes the cursor focus to the previous starting shape of a stage. This has the effect of scrolling up the page so that the earlier parts of the pipeline are shown.
PAGE DOWN	Changes the cursor focus to the next starting shape of a stage. This has the effect of scrolling down the page so that the next parts of the pipeline are shown.
HOME	Changes focus to the **Start** shape of the pipeline.
END	Changes focus to the **End** shape of the pipeline.

How to Read Pipeline Component Properties

The Properties window contains two sections for components: general properties and component properties. General properties are common to all components, though their values change between components.

To read the general properties for a pipeline component

1. Drag the pipeline component into a stage of the pipeline, or select a component if it already exists in the pipeline.

2. In the Properties window, in the **General** section, do the following.

Use this	To do this
Name	Indicates the component name.
Assembly	Indicates the assembly and associated .NET information for the component.
Description	Indicates the friendly name of the pipeline component.
Managed	Indicates whether the pipeline component is managed. **Yes** if the

	component is a .NET managed component; **No** if the pipeline component is a COM component.
Path	Indicates the fully qualified path to the .NET component.
Type	Indicates the component type, the assembly, and the associated .NET information for the component.

Configuring Native Pipeline Components

Pipeline components can expose their own custom properties at design time. Any public property defined in the component will be rendered in Pipeline Designer providing that read and write accessors for that property are implemented. Pipeline Designer will display the component properties in accordance with their declaration; for example, if the property is declared as read-only, it will be displayed as such in Pipeline Designer.

Custom pipeline components must implement the **IPersistPropertyBag** interface to enable the creation of these custom properties. Properties created with the **IPersistPropertyBag** interface can be set in the Properties window of Microsoft® Visual Studio® 2005, just like all the properties of the native pipeline components. This section contains procedures for configuring each of the included pipeline components.

In This Section

- How to Configure the BizTalk Framework Assembler Pipeline Component
- How to Configure the BizTalk Framework Disassembler Pipeline Component
- BizTalk Framework Schema and Properties
- How to Configure the Flat File Assembler Pipeline Component
- How to Configure the Flat File Disassembler Pipeline Component
- How to Configure the MIME/SMIME Decoder Pipeline Component
- How to Configure the MIME/SMIME Encoder Pipeline Component
- MIME/SMIME Property Schema and Properties
- How to Configure the Party Resolution Pipeline Component
- How to Configure the XML Assembler Pipeline Component
- How to Configure the XML Disassembler Pipeline Component

- XML and Flat File Property Schema and Properties
- How to Configure the XML Validator Pipeline Component

How to Configure Native Pipeline Components

The following code shows how to configure pipeline components.

How to Configure the BizTalk Framework Assembler Pipeline Component

To configure the properties for the BizTalk Framework Assembler pipeline component

1. Drag the BizTalk Framework Assembler pipeline component into the Assemble stage of the send pipeline.
2. In the Properties window, in the **Pipeline Component Properties** section, set the following property values.

Use this	To do this
Add processing instructions text	Allows assembled XML documents to contain processing instructions as a value of this property. This also allows documents to contain instructions for applications. **Note** Processing instruction text should conform to the W3C XML processing instruction standards. Default value: None
Add XML declaration	Adds an XML declaration to an outgoing message. When true, the following XML declaration is added to the outgoing message: `<?xml version='1.0' encoding='UTF8'>`. The encoding specified depends on the encoding used by the BizTalk Framework Assembler, which honors the specific run-time properties carrying the encoding information. Default value: **True**
Delivery receipt address	Specifies the address to which the delivery receipt for the BizTalk Framework document should be sent. **Warning** When using the BizTalk Framework Assembler with acknowledgements enabled, there is a chance that acknowledgement processing can become backlogged resulting in a deadlock. The deadlock occurs when multiple acknowledgements are processed in separate batches for a single message. To avoid

	this problem you should configure the batch size to 1 for the delivery receipt address port location entered.
Delivery receipt address type	Specifies the type of address to which the delivery receipt for the BizTalk Framework document should be sent. Default value: biz: **Note** The biz: prefix was used to signify organization identifiers for the source and destination endpoints in Microsoft® BizTalk® Server 2000 and BizTalk Server 2002, and for interoperability with those systems. The prefix is provided as a default. For example, type = "biz:OrganizationName", (src|dest) = "Party1".
Delivery receipt send by time	Specifies the time (in minutes) by which the delivery receipt for the BizTalk Framework document must be received. Default value: 30
Destination address	Specifies the destination address. Default value: None
Destination address type	Specifies the destination address type. Default value: biz: **Note** The biz: prefix was used to signify organization identifiers for the source and destination endpoints in BizTalk Server 2000 and BizTalk Server 2002, and for interoperability with those systems. The prefix is provided as a default. For example, type = "biz:OrganizationName", (src|dest) = "Party1".
Document schemas	Indicates the namespace and typename of the schema or schemas to be applied to the document. For more information, see Using the Schema Collection Property Editor. **Note** You may receive a "Two or more of the selected schema share the same target namespace" error if you specify two or more schemas for the **Document schemas** property. Default value: Empty collection
Document topic	Identifies a URI reference that uniquely identifies the overall purpose of the BizTalk Framework document. Default value: None

Envelope schemas	Indicates the namespace and typename of the schema or schemas that will be applied to the envelope. For more information, see Using the Schema Collection Property Editor. **Note** You may receive a "Two or more of the selected schema share the same target namespace" error if you specify two or more schemas for the **Envelope schemas** property. Default value: BTF2Schemas.btf2_envelope
Generate delivery request receipt	Indicates whether the delivery receipt request for the BizTalk Framework document should be generated. This is used to enable reliable messaging for the BizTalk Framework. Default value: **True**
Message time to live (in minutes)	Specifies the amount of time (in minutes) that the message is valid. Default value: 30
Processing instructions	Specifies how XML processing instructions are handled in the XML instance document. **Append**: The value of **Add processing instructions text** should append to pre-existing processing instructions in the message. **Create New**: The value of **Add processing instructions text** entered in the field should overwrite or replace any pre-existing processing instructions in the message. If **Create New** is selected, **Add processing instructions text** must contain valid processing instructions. **Ignore**: If processing instructions text exists in the message, it is removed. Default value: **Append**
Source address	Specifies the source address. Default value: None
Source address type	Specifies the source address type. Default value: biz: **Note** The biz: prefix was used to signify organization identifiers for the source and destination endpoints in BizTalk Server 2000 and

	BizTalk Server 2002, and for interoperability with those systems. The prefix is provided as a default. For example, type = "biz:OrganizationName", (src|dest) = "Party1".
Target charset	Specifies the target character set used for encoding of outgoing messages. Default value: None

How to Configure the BizTalk Framework Disassembler Pipeline Component

The BizTalk Framework Disassembler pipeline component should be used in the Disassemble stage of a receive pipeline.

To configure the properties for the BizTalk Framework Disassembler pipeline component

1. Drag the BizTalk Framework Disassembler pipeline component into the Disassemble stage of a receive pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	**To do this**
Allow unrecognized message	Indicates whether to allow messages that do not have a recognized schema to be passed through the disassembler. Default value: **False**
Document schemas	Indicates the namespace and typename of the schema or schemas to be applied to the document. For more information, see Using the Schema Collection Property Editor. Schemas specified in this property should have unique target namespaces. If any of the schemas have the same namespace, the validation of the document instances may not work as expected. If schemas must have the same namespace, you should either create a separate pipeline for each schema and specify one schema per BizTalk Framework Disassembler pipeline component or use one pipeline but do not specify any schemas as parameters for the BizTalk Framework Disassembler pipeline component. Default value: Empty collection
Envelope	Indicates the namespace and typename of the schema or schemas

schemas	to be applied to the envelope. For more information, see Using the Schema Collection Property Editor. Schemas specified in this property should have unique target namespaces. If any of the schemas have the same namespace, the validation of the document instances may not work as expected. If schemas must have the same namespace, you should either create a separate pipeline for each schema and specify one schema per BizTalk Framework Disassembler pipeline component or use one pipeline but do not specify any schemas as parameters for the BizTalk Framework Disassembler pipeline component. Default value: Empty collection
Validate document structure	When **True**, performs a validation of the incoming message to the disassembler, including the envelopes. **Note** When **True**, you may receive a "Two or more of the selected schema share the same target namespace" error if you specify two or more schemas for the **Document schemas** or **Envelope schemas** properties. Default value: **False**

BizTalk Framework Schema and Properties

The **http://schemas.microsoft.com/BizTalk/2003/btf2-properties** namespace contains properties you can use to set message and part context properties for the BizTalk Framework Disassembler pipeline component. The BizTalk Framework Disassembler pipeline component uses these properties to generate the appropriate headers in the message that is created. The following table describes the BizTalk Framework properties.

Name	**Type**	**Description**
IsReliable	xs:boolean	Indicates whether the BizTalk Framework message should be resent until an acknowledgment is received from a destination. This property is set internally by BizTalk Framework components and used by the engine. Do not change the value in this property from your code.
PassAckThrough	xs:boolean	Indicates whether an acknowledgement message should be passed through a BizTalk Framework Dissembler pipeline component instead of being consumed.

eps_to_address	xs:string	Specifies the destination address.
eps_to_address_type	xs:string	Specifies the destination address type.
eps_from_address	xs:string	Specifies the source address.
eps_from_address_type	xs:string	Specifies the source address type.
prop_identity	xs:string	A URI reference that uniquely identifies the BizTalk Framework document for purposes of logging, tracking, error handling, or other document processing and correlation requirements.
prop_sentAt	xs:string	The send timestamp of the BizTalk Framework document.
prop_topic	xs:string	A URI reference that uniquely identifies the overall purpose of the BizTalk Framework document.
svc_deliveryRctRqt_sendTo_address	xs:string	Specifies the address to which the delivery receipt for the BizTalk Framework document should be sent.
svc_deliveryRctRqt_sendTo_address_type	xs:string	Specifies the type of address to which the delivery receipt for the BizTalk Framework document should be sent.
svc_deliveryRctRqt_sendBy	xs:dateTime	Specifies the time (in minutes) by which the delivery receipt for the BizTalk Framework document must be received.
svc_commitmentRctRqt_sendTo_address	xs:string	Specifies the address where the notification of the recipient's decision about processing of the sender's request should be sent to.
svc_commitmentRctRqt_sendTo_address_type	xs:string	Specifies the type of address where the notification of the recipient's decision about processing of the sender's request should be sent to.
svc_commitmentRctRqt_sendBy	xs:dateTime	Specifies the time (in minutes) by which the commitment receipt for the BizTalk Framework document must be received by the sender.
prc_type	xs:string	Provides a URI reference that specifies the type of business process involved in the

		processing of BizTalk Framework messages.
prc_instance	xs:string	Provides URI reference that uniquely identifies a specific instance of the business process that the BizTalk Framework document is associated with.
deliveryRct_receivedAt	xs:dateTime	Specifies the receiving timestamp for the document acknowledged by this receipt. The receiving timestamp may reflect either the time when the first copy was received or the time at which the copy being acknowledged was received.
deliveryRct_identity	xs:string	Specifies an identity of the BizTalk Framework document acknowledged by the delivery receipt.
commitmentRct_identity	xs:string	Specifies the identity of a BizTalk Framework document acknowledged by the commitment receipt.
commitmentRct_decidedAt	xs:string	Specifies the processing decision timestamp for the document acknowledged by this receipt.
commitmentRct_decision	xs:string	Specifies the actual decision, with possible values of positive or negative.
commitmentRct_commitmentCode	xs:QName	Specifies the qualified name (in XSD) that specifies a more specific status regarding the processing decision.

How to Configure the Flat File Assembler Pipeline Component

The Flat File Assembler pipeline component is used to serialize an XML document into delimited or positional flat file format before sending it out of the server.

To configure the properties for the Flat File Assembler pipeline component

1. Drag the Flat File Assembler pipeline component into the Assemble stage of the send pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	To do this
Document schema	Select a flat file document schema to use for serializing the message from XML to the flat file format. If no schema is specified, run-time schema discovery is done. You can create the flat file document schema in BizTalk Editor. Default value: None
Header schema	Select a schema for the header part of the flat file message. A header schema can also be specified in the message context property named **HeaderSpecName** under the xmlnorm namespace. In this case, it will override the header schema specified in Pipeline Designer. You can create the schema for the header part of the flat file message with BizTalk Editor. Default value: None
Target charset	Specifies the target character set used for encoding of outgoing messages. Default value: None
Trailer schema	Select a schema for the trailer part of the flat file message. You can create the schema for the trailer part of the flat file message in BizTalk Editor. Default value: None

How to Configure the Flat File Disassembler Pipeline Component

The Flat File Disassembler pipeline component is used for disassembling documents in flat file format and converting them into XML format.

To configure the properties for the Flat File Disassembler pipeline component

1. Drag the Flat File Disassembler pipeline component into the Disassemble stage of a receive pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	To do this
Document schema	Select a flat file document schema to use for parsing the message from flat file to XML format. The flat file document schema for parsing can be created in BizTalk Editor. Default value: None **Note** You must specify a schema for this property, or a compile-time error will occur.
Header schema	Select a schema for the header part of the flat file message. The schema for the header part of the flat file message can be created in BizTalk Editor. Default value: None
Preserve header	Set this property to **True** if you need to store the flat file message header on the message context. Preserving the header of the flat file message enables the header structure and content to flow with the message through BizTalk Server. The header can then be used when serializing the message back to flat file format in the Flat File Assembler pipeline component. When the preserved header is being serialized by the Flat File Assembler, the header document design-time property can lack the name of the header schema, because this information can be obtained dynamically at run time. This is accomplished by using the message type of the preserved header. Default value: **False**
Trailer schema	Select a schema for the trailer part of the flat file message. The schema for the trailer part of the flat file message can be created in BizTalk Editor. Default value: None
Recoverable interchange processing	When "false" - indicates that entire interchange is disassembled as a unit (if any contained message fails, entire interchange is suspended). When "true" - indicates that messages within interchange are extracted individually by disassembler with possibility of some propagating through messaging pathway and others being suspended. For details, see section on "Recoverable Interchange

	Disassembly."
Validate document structure	Set this property to **True** if you need to validate all the parts of the flat file message (header, body, and trailer) to make sure they conform to their schemas. This option reduces the performance of the Flat File Disassembler, so it is set to **False** by default. Default value: **False**

How to Configure the MIME/SMIME Decoder Pipeline Component

The MIME/SMIME Decoder pipeline component is used for decoding and decrypting MIME/SMIME encoded messages and for verifying digital signatures of signed messages. This component is useful when secured document interchange is needed between external partners and BizTalk Server. This component can also be used for receiving messages with attachments.

To configure the properties for the MIME/SMIME Decoder pipeline component

1. Drag the MIME/SMIME Decoder pipeline component into the Decode stage of a receive pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	To do this
Allow Non MIME Message	Set this property to **True** if you expect that the receive pipeline may get messages that are not MIME encoded. This option is useful when the MIME/SMIME Decoder pipeline component is used in a pipeline that receives messages in different forms, for example, MIME encoded or plain XML. By default, this property is set to **False**, meaning that the component generates an error if it receives a non-MIME encoded message on input. Default value: **False**
Check Revocation List	Set this property to **True** if you want to check the certificate revocation list for the certificates that senders use for signing messages that are being sent to BizTalk Server. Disabling this option increases the performance of the component. The certificate revocation list associated with a certificate is downloaded from a remote Web site (for example, Verisign.com). If the BizTalk Server cannot connect to the remote Web site, the

	message fails in the pipeline. Default value: **True**

How to Configure the MIME/SMIME Encoder Pipeline Component

Use the MIME/SMIME Encoder pipeline component to encode and encrypt outgoing messages and to sign outgoing messages. This component is useful when you require secured document interchange between BizTalk Server and external partners. You can also use this component to send BizTalk Server multi-part messages.

⊟To configure the properties for the MIME/SMIME Encoder pipeline component

1. Drag the MIME/SMIME Encoder pipeline component into the Encode stage of a send pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	**To do this**
Add Signing Cert To Message	If the **Signature Type** property is not **NoSign**, you can select whether to add the signing certificate to the signed message by setting the **Add Signing Cert to Message** property. Default value: **True**
Check revocation list	Specifies whether to check the certificate revocation list while processing a SMIME message. Default value: **True**
Content transfer encoding	Indicates the encoding format. Options: Base64, QuotedPrintable, SevenBit, EightBit, Binary, and UUEncode. Default value: **Base64**
Enable encryption	Set to **True** if you want to encrypt the outgoing message. If this option is enabled, the user can select the encryption algorithm to use by setting the **Encryption algorithm** property. For message encryption, the MIME/SMIME Encoder pipeline component uses the public key certificate that is associated with a send port in BizTalk

	Explorer. Default value: **False**
Encryption algorithm	Define the encryption algorithm. This property can only be set if **Enable encryption** is set to **True**. Options: DES3, DES, RC2. Default value: **DES3**
Send body part as attachment	Set to **True** if you want to send the body party of a BizTalk message as a MIME attachment. Default value: **False** **Important** You should not set this property to **True** when sending messages between BizTalk Servers. Otherwise, the message decoding will require custom coding because the message is interpreted on the receive side as a two-part message.
Signature Type	If you want to sign the outgoing message, select a signing format with this property. This property has three values: • **NoSign**. The message will not be signed. • **ClearSign**. The signature will be appended to the message. **ClearSign** cannot be used if **Enable Encryption** is set to **True**. • **BlobSign**. The signature will be appended to the message and the message will be encoded. For message signing, the MIME/SMIME Encoder component uses the private key client certificate that is associated with BizTalk Group in the BizTalk Administration console. Default value: **NoSign**

To set the file name for MIME attachments, use the **FileName** property in the **System** namespace for the message part.

MIME/SMIME Property Schema and Properties

The **http://schemas.microsoft.com/BizTalk/2003/mime-properties** namespace contains properties you can use to set message and part context properties for the MIME/SMIME Encoder pipeline component. The MIME/SMIME Encoder uses these properties to generate the appropriate MIME/SMIME headers in the message that is created. The following table describes the MIME/SMIME properties.

Property	Scope	Type	Description
FileName	Per message part	xs:string	Sets the file name header of the MIME/SMIME part.
ContentID	Per message part	xs:string	Sets the Content-ID header of the MIME/SMIME part.
ContentDescription	Per message part	xs:string	Sets the Content-Description header of the MIME/SMIME part.
ContentTransferEncoding	Per message part	xs:string	Sets the Content-Transfer-Encoding header of the generated MIME/SMIME part. This value overrides the **Content transfer encoding** value configured in Pipeline Designer. For a multi-part message, you can use different encodings for different MIME/SMIME parts.
ContentLocation	Per message part	xs:string	Sets the Content-Location header of the generated MIME/SMIME part.
IsMIMEEncoded	Per message part	xs:boolean	Specifies whether the message has a MIME/SMIME payload. The MIME component writes to this value, so you do not have to set it.

The MIME/SMIME Encoder also uses the following part properties from the **System** namespace.

Property	Type	Description
ContentType	xs:string	Corresponds to the Content-Type header of the generated MIME/SMIME part.
FileName	xs:string	Corresponds to the file name.

How to Configure the Party Resolution Pipeline Component

The Party Resolution pipeline component is used to map the user security ID and the certificate subject for the client to a BizTalk Server party. The mapping is used to enforce authentication of the parties who send messages to BizTalk Server. For more information about partner management see Partner Management in BizTalk Explorer.

To configure the properties for the Party Resolution pipeline component

1. Drag the Party Resolution pipeline component into the ResolveParty stage of a receive pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	To do this
Resolve Party By Certificate	Set to **True** if you want to enable party resolution based on the thumbprint of the signature certificate from the party from where the messages are received. Default value: **True**
Resolve Party By SID	Set to **True** if you want to enable party resolution based on the security identifier (SID) of the sender account. If both properties are set to **True**, the **Resolve Party By Certificate** property takes precedence over the **Resolve Party By SID** property, meaning that if both the certificate and the SID are available, the certificate subject is used. If the party cannot be resolved by using the certificate subject, the component does not fall back to the **Resolve Party By SID** property, and the default value (s-1-5-7) is stamped for **BTS.SourcePartyID**. Default value: **True**

How to Configure the XML Assembler Pipeline Component

The XML Assembler pipeline component is used to wrap the XML documents into the XML envelopes before sending a message from BizTalk Server 2006.

To configure the properties for the XML Assembler pipeline component

1. Drag the XML Assembler pipeline component into the Assemble stage of a send pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	To do this
Add processing instructions text	Specifies the XML processing instructions to add to the beginning of the target XML document. **Note** Processing instruction text should conform to the W3C XML processing instruction standards. Default value: None
Add XML declarations	When **True**, adds an XML declaration similar to the following to the outgoing document: `<?xml version='1.0' encoding='UTF-8'>`. Encoding is determined by the target document encoding set at run time. Default value: **True**
Document schemas	Indicates the namespace and typename of the schema or schemas to be applied to the document. For more information, see Using the Schema Collection Property Editor. **Note** You may receive a "Two or more of the selected schema share the same target namespace" error if you specify two or more schemas for the **Document schemas** property. Default value: Empty collection
Envelope schemas	Indicates the namespace and typename of the schema or schemas that will be applied to the envelope. For more information, see Using the Schema Collection Property Editor. **Note** You may receive a "Two or more of the selected schema share the same target namespace" error if you specify two or

	more schemas for the **Envelope schemas** property. Default value: Empty collection
Add processing instructions	**Append**: Value of **Add processing instructions text** should append to pre-existing processing instructions in the message. **Create New:** Value of **Add processing instructions text** entered in the field should overwrite or replace any pre-existing processing instructions in the message. If **Create New** is selected, the **Add processing instructions text** must contain valid processing instructions. **Ignore**: If processing instructions text exists in the message, it is removed. Default value: **Append**
Target charset	Specifies the character set used to encode outgoing messages. Default value: None

How to Configure the XML Disassembler Pipeline Component

The XML Disassembler pipeline component should be used in the Disassemble stage of a receive pipeline.

To configure the properties for the XML Disassembler pipeline component

1. Drag the XML Disassembler pipeline component into the Disassemble stage of a receive pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, do the following.

Use this	**To do this**
Allow unrecognized message	Indicates whether to allow messages that do not have a recognized message type to be passed through the disassembler. Default value: **False**
Document schemas	Indicates the namespace and typename of the schema or schemas to be applied to the document. For more information,

	see Using the Schema Collection Property Editor. Schemas specified in this property should have unique target namespaces. If any of the schemas have the same namespace, the validation of the document instances may not work as expected. If schemas must have the same namespace, you should either create a separate pipeline for each schema and specify one schema per XML Disassembler pipeline component or use one pipeline but do not specify any schemas as parameters for the XML Disassembler pipeline component. Default value: Empty collection
Envelope schemas	Indicates the namespace and typename of the schema or schemas to be applied to the envelope. For more information, see Using the Schema Collection Property Editor. Schemas specified in this property should have unique target namespaces. If any of the schemas have the same namespace, the validation of the document instances may not work as expected. If schemas must have the same namespace, you should either create a separate pipeline for each schema and specify one schema per XML Disassembler pipeline component or use one pipeline but do not specify any schemas as parameters for the XML Disassembler pipeline component. Default value: Empty collection
Recoverable Interchange Processing	When "false" - indicates that entire interchange is disassembled as a unit (if any contained message fails, entire interchange is suspended). When "true" - indicates that messages within interchange are extracted individually by disassembler with possibility of some propagating through messaging pathway and others being suspended. For details, see section on "Recoverable Interchange Disassembly."
Validate document structure	When **True**, performs a validation of the incoming message against document and optionally envelope schemas. If a promoted property does not have a default or fixed value and this property is set to **False**, the property is not promoted. **Note** When **True**, you may receive a "Two or more of the selected schema share the same target namespace" error if you

	specify two or more schemas for the **Document schemas** or **Envelope schemas** properties. Default value: **False**

XML and Flat File Property Schema and Properties

The **http://schemas.microsoft.com/BizTalk/2003/xmlnorm-properties** namespace contains properties you can use to configure Flat File Assembler and Flat File Disassembler pipeline components. The properties are described in the following table.

Property	**Type**	**Description**
FlatFileHeaderDocument	xs:string	The header of an incoming flat file document can be stored with this property.
AllowUnrecognizedMessage	xs:Boolean	Specifies whether unrecognized messages should be processed by XML components.
ProcessingInstruction	xs:string	Processing instruction text for outgoing documents.
DocumentSpecName	xs:string	The schema for XML components to use for parsing or serializing documents. Schemas specified in this property should have unique target namespaces. If any of the schemas have the same namespace, the validation of the document instances may not work as expected. If schemas must have the same namespace, you should either create a separate pipeline for each schema and specify one schema per XML Disassembler pipeline component or use one pipeline but do not specify any schemas as parameters for the XML Disassembler pipeline component.
EnvelopeSpecName	xs:string	The envelope specification for XML components to use for parsing or serializing documents. Schemas specified in this property should have unique target namespaces. If any of the schemas have the same namespace, the validation of the document instances may not work as expected. If schemas must have the same namespace, you should either create a

		separate pipeline for each schema and specify one schema per each XML Disassembler pipeline component or use one pipeline but do not specify any schemas as parameters for the XML Disassembler pipeline component.
TargetCharset	xs:string	The character set for XML components to use for encoding output messages.
SourceCharset	xs:string	The character set used to encode a document before being processed by the XML Disassembler.
ProcessingInstructionOption	xs:int	Specifies how processing instructions are added to outgoing documents. For more information about the ProcessingInstructionOption, see Processing Instructions in the XML Assembler Pipeline Component.
AddXMLDeclaration	xs:boolean	Specifies whether an XML declaration should be added to an outgoing document.
HeaderSpecName	xs:string	Specifies a flat file document header.
TrailerSpecName	xs:string	Specifies a flat file document trailer.
PromotePropertiesOnly	xs:boolean	When set to **True**, the XML Disassembler component does not remove a message envelope or disassemble it. Only property promotion is performed.

How to Configure the XML Validator Pipeline Component

The XML Validator pipeline component can be used in any stage (except Disassemble or Assemble) in a send or receive pipeline.

To configure the properties for the XML Validator pipeline component

1. Drag the XML Validator pipeline component into the Validate stage of a receive pipeline.

2. In the Properties window, in the **Pipeline Component Properties** section, set the following.

Use this	To do this
Document schemas	Indicates the namespace and typename of the schema or schemas to be applied to the document. For more information, see Using the Schema Collection Property Editor. If no schemas are specified, the run-time schema discovery will be done by using the message's target namespace and root element name information. **Note** You may receive a "Two or more of the selected schema share the same target namespace" error if you specify two or more schemas for the **Document schemas** property. Default value: Empty collection

How to Deploy Pipelines

Pipelines are compiled and deployed as part of the solution build and deploy process. The compiler calls the **Validate** method on each component, allowing the components to return compile errors on the configured information. After building, the pipeline is deployed in the same assembly with the rest of the solution when the solution is deployed.

Per-instance pipeline configuration

Per-instance pipeline configuration is used to modify properties of pipeline components within a deployed pipeline at the send port or receive location level. Per-instance pipeline configuration is useful when only a few pipeline component properties need to be modified per instance. When the XML file describing the per-instance configuration of the pipeline components is read, it overrides the properties set in the pipeline file.

Per-instance pipeline configuration is set by using the BizTalk® Explorer object model. The BizTalk Explorer object model provides the **ReceivePipelineData** property on the **IReceiveLocation** and **ISendPort** interfaces for setting the configuration of receive pipeline components. The BizTalk Explorer object model also provides the **SendPipelineData** method on the **IReceivePort** and **ISendPort** interfaces for setting configuration of send pipeline components.

Per-instance pipeline configuration does not support the following:

- Rearranging stages within the pipeline
- Adding or removing stages
- Rearranging components within stages
- Adding or removing components

The only supported changes are in the configuration of pipeline components. Per-instance configuration of a pipeline component overrides the common pipeline component configuration. If a parameter of a component is not specified in per-instance pipeline configuration, the common configuration for that parameter (as configured in Pipeline Designer) is used.

How to Secure Pipelines

Hosts can be marked in the administration console as **Authentication Trusted**. Denoting a host as Authentication Trusted means that the Microsoft® BizTalk® Server will trust the security-related properties sent on the message context of a message from that host. The security-related properties on the message context are the **OriginatorPID**, which corresponds to the message context property BTS.SourcePartyID, and the **OriginatorSID**, which corresponds to the message context property **BTS.WindowsUser**. For more information about the message context properties, see **Message Context Properties**.

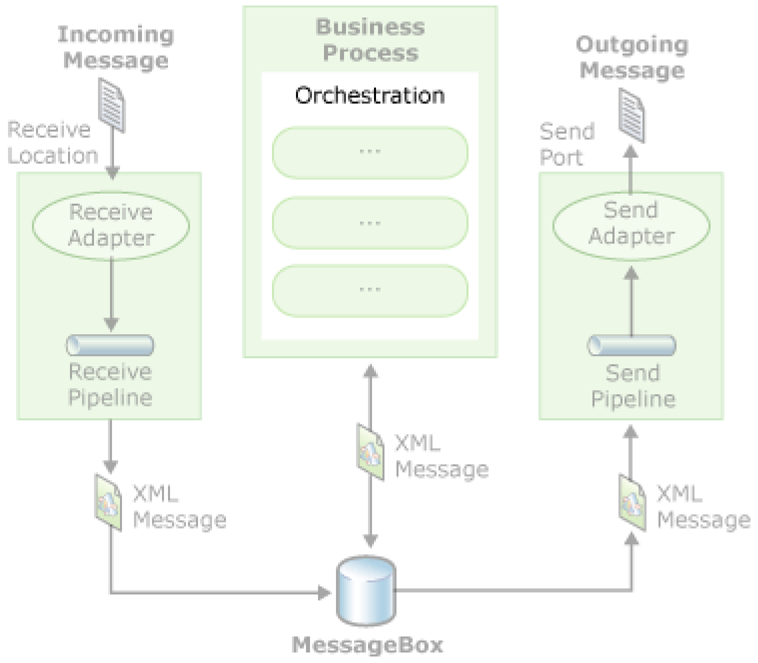
A host that is marked as **Authentication Trusted**is allowed to indicate that the trusted host is adding a message to the queue from someone other than itself as the sender of the message. In other words, hosts that are not marked as **Authentication Trusted** are not allowed to add a message to the queue from a message sender other than themselves.

For information about encoding and decoding messages sent over SMTP or HTTP, see MIME/SMIME Encoder Pipeline Component and MIME/SMIME Decoder Pipeline Component.

For information about signature verification when dealing with third parties, see Party Resolution Pipeline Component and Partner Management in BizTalk Explorer.

Creating Orchestrations Using Orchestration Designer

Orchestration in BizTalk Server 2006 workflow



BizTalk® Orchestration Designer is a tool for conveniently creating visual representations of your business processes that are automatically reflected in underlying code and which you build into an executable module. It provides a wide variety of shapes that correspond to different actions that you might want to perform.

This section provides conceptual information, reference information, and task-related information to help you to understand and use BizTalk® Orchestration and BizTalk Orchestration Designer.

In This Section

- About Orchestration
- **Using Orchestration Designer**
- Creating Orchestrations
- Building and Running Orchestrations
- Working with the Orchestration Engine
- **Orchestration Developer Reference**

About Orchestration

Orchestration is a flexible, powerful tool for representing your executable business processes. You can design flow, interpret and generate data, call custom code, and organize it all in an intuitive visual drawing.

Messages, the send and receive actions that operate on them, and the ports through which they are transported are all fundamental elements of an orchestration. The message is the medium by which orchestrations communicate with the outside world and by which e-business is conducted.

Receive and **Send** shapes encapsulate the functionality you need to receive messages in your orchestration and send messages from it. You should become familiar with the various shapes that Orchestration Designer provides to represent the logical flow of your orchestration.

You should understand advanced orchestration concepts such as Web services, correlation, and long-running transactions. You might not need to use all of these facilities, but it is helpful to know what they can do for you.

Web services are programs with interfaces that adhere to the standards set forth in the Web Services Description Language (WSDL). By defining message types, ports, port types, and operations in a standard way, disparate systems can communicate effectively with each other.

Correlation is the mechanism by which messages are associated with particular running instances of an orchestration, so that your business processes gets the appropriate information when many instances are running and many messages are being sent back and forth.

Transactions enable you to maintain the state of an orchestration appropriately if any unexpected issues arise. Orchestration Designer makes available various exception-handling facilities, which enable you to deal with errors in a controlled and predictable manner.

In This Section

- Steps in Orchestration Development
- Security Considerations for Developing Orchestrations

Steps in Orchestration Development

To develop an orchestration, you typically perform the following basic actions:

- Add shapes to represent your business processes.

  Orchestration Designer provides you with a toolbox of shapes that can be used to represent different actions or other abstractions. For more information, see Orchestration Shapes.

- Define schemas to describe the format of your messages.

  You can define schemas with BizTalk® Editor. For more information, see Creating Schemas for XML Messages.

- Define ports through which messages are sent and received.

  You can define ports to specify how and where messages are sent and received. For more information, see Using Ports in Orchestrations.

- Bind **Send** and **Receive** shapes to ports.

  You can connect your **Send** and **Receive** shapes to ports and specify the port operations that they use. For more information, see Using the Send Shape and Using the Receive Shape.

- Assign or transform data between messages.

  You can use the **Construct Message** shape to assign message values or do message transformations. For more information, see Constructing Messages.

- Identify any custom components that you might want to write or add as reference to work within your orchestration.

- Define and assign orchestration variables to manage data in your running orchestration.

  You can use the Variables folder in the Orchestration View window to declare your orchestration variables, and BizTalk Expression Editor to edit expressions that assign and use the variables in various shapes. For more information, see Using Expressions and Variables.

- Build your orchestration to test it for completeness.

  You build your orchestration when you compile the project or solution that contains it. For more information, see Building Orchestrations.

Security Considerations for Developing Orchestrations

When designing your orchestrations, you should consider potential security issues.

Avoid subscriptions based on content from untrusted messages

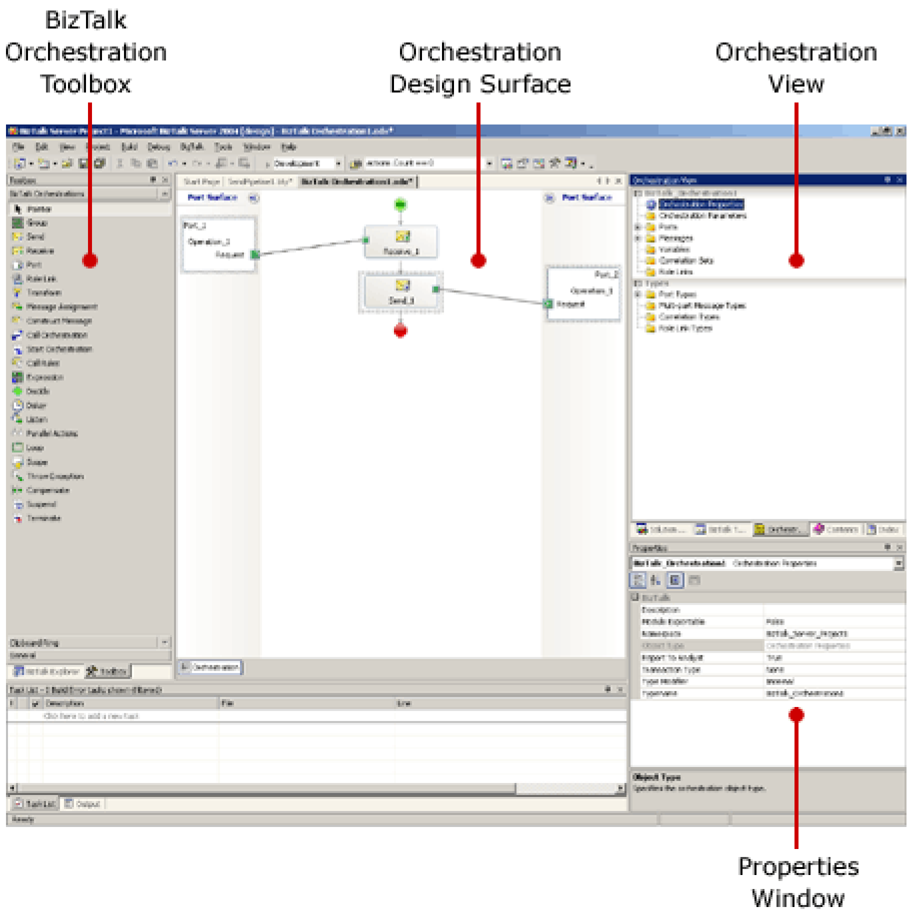
To ensure that a low-privilege message does not initiate an orchestration instance that could potentially create subscriptions based on the message content or context, it is highly recommended that your orchestrations do not create their message subscriptions based on the content or context of a message that is not trusted.

For information about security in BizTalk Server 2006, see Securing BizTalk Server.

Creating Orchestrations

This section describes the various tasks that you need to perform as you create your orchestrations.

Orchestration Design Surface



The Orchestration Design Surface is a visual designer that you can use to create a BizTalk Orchestration, and is the central component of Orchestration Designer. It is a canvas that you can drag shapes onto from the Toolbox, and then configure the shapes. As a Visual Studio editor window, it occupies the main window area used by other Visual Studio editor windows.

The name of the orchestration is displayed on the top tab of the Orchestration Design Surface window and in the Visual Studio window title bar.

The design surface itself is divided into three areas: the Process Area and two Port Surfaces. The central Process Area contains shapes that describe the actual process flow of the orchestration. It is flanked on both sides by Port Surfaces, which contain only Port and Role Link shapes that interact with the **Send** and **Receive** shapes in the Process Area.

Process Area

The Process Area is the main part of the Orchestration Design Surface of Orchestration Designer, and is always horizontally centered in the Orchestration Design Surface.

On either side of the Process Area you see Port Surfaces. The **Begin** shape is placed at the top of the design surface and the orchestration grows downward as you add shapes.

Port Surfaces

Two Port Surfaces are displayed in the Orchestration Design Surface, one on either side of the Process Area. Port Surfaces can contain two kinds of shapes: **Ports** and **Role Links**. These shapes interact with the **Send** and **Receive** shapes in the Process Area.

It makes no difference which Port Surface you use for a shape; that is, the shape functions identically on either the right or the left Port Surface. Having two Port Surfaces on which to place new ports lets you create orchestrations with fewer crisscrossing connectors that therefore are easier to read.

Both Port Surfaces can be collapsed or expanded by double-clicking on them or by clicking on the double arrow icon.

Zoom Support

BizTalk Server 2006 provides the ability to zoom in or zoom out of the Orchestration designer surface. You can use zoom support by completing one of the following steps:

- Right-click the Orchestration designer surface and click the **Zoom** menu option. You can then select the percentage of magnification that you would like to apply.
- Use the CTRL + MWHEEL combination to zoom in or zoom out. Hold down the CTRL button and simultaneously rotate the mouse wheel up or down. Use the

CTRL+MWHEELUP combination to zoom out and the CTRL+MWHEELDOWN combination to zoom in.

In This Section

Working with Patterns in Orchestrations Creating and Modifying Orchestrations Designing Orchestration Flow How to Use Ports in Orchestrations Using Role Links How to Work with Messages in Orchestrations Using Expressions and Variables Using Transactions and Handling Exceptions Using Business Rules in Orchestrations Executing a Pipeline from within an Orchestration

Working with Patterns in Orchestrations

This section discusses some of the integration patterns such as,

- Aggregator
- Splitter
- Scatter and Gather
- Content Based Router
- Composed Message Processor
- Message Broker
- Dynamic Router
- Correlation Identifier
- Message Translator
- Invalid Message Channel

Creating and Modifying Orchestrations

After you have started a BizTalk project, you can create new orchestrations and add existing orchestrations to the project. See the following procedures to create and save an orchestration, to add an existing orchestration to a project or remove one from it, to change the name of an orchestration, and to set orchestration properties.⊟To create an orchestration

1. In Solution Explorer, right-click the project name, select **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, in the **Categories** pane, click **BizTalk Project Items**, and then in the **Templates** pane, click **BizTalk Orchestration**.

3. In the **Name** box at the bottom of the dialog box, supply a name for the orchestration, and then click **Add**.

   The new orchestration is created and displayed in Orchestration Designer, and a corresponding .odx file is created and displayed in Solution Explorer.

To add an existing orchestration to a project

1. In Solution Explorer, right-click the project name, click **Add**, and then click **Existing Item**.
2. In the **Add Existing Item** dialog box, navigate to the directory containing the orchestration, select the orchestration, and then click **Add**.

   The orchestration is added to the project.

To change the name of an orchestration

1. In Solution Explorer, right-click the .odx file you want to change, and then click **Rename**.
2. Type the new file name you want, and then press ENTER.

To save an orchestration

1. On the **File** menu, click **Save <orchestration name>**.

To remove an orchestration from a project

1. In Solution Explorer, right-click the file you want to remove, and then click **Exclude From Project**.

To include an excluded orchestration in a project

1. In Solution Explorer, click the **Show All** toolbar button, right-click on the .odx file you want, and select **Include in Project**.

To set orchestration properties

1. Open the orchestration by double-clicking on the .odx file in the project, or by selecting the tab containing the orchestration in the Process Area.
2. In the Orchestration View window, select **Orchestration Properties**.

   —Or—

   Click the **Process Area** background of the Orchestration Design Surface.

3. In the Properties window, specify the following properties. Note that some properties appear only under certain circumstances.

Property	**Description**
Batch	Determines whether an orchestration that is an atomic transaction can be batched with other instances.
Compensation	Specifies what type of compensation to perform on the orchestration.
Isolation Level	For transactional orchestrations, determines the degree to which data is accessible among concurrent transactions.
Module Exportable	Determines whether or not the module can be exported to BPEL4WS.
Module XML Target Namespace	The XML target namespace used when exporting types to BPEL4WS.
Namespace	Determines the name of the containing module that includes the orchestration and the orchestration types.
Orchestration Exportable	Indicates whether this orchestration is intended to be exportable to BPEL4WS.
Orchestration XML Target Namespace	The XML target namespace used when exporting this orchestration to BPEL4WS.
Retry	Specify whether to retry a transactional orchestration if it fails.
Timeout	The time in seconds until a transactional orchestration fails due to inactivity.
Transaction Identifier	Unique identifier for a transactional orchestration.
Transaction Type	Determines whether the orchestration is an atomic transaction, a long-running transaction, or is not transacted.
Type Modifier	Determines the scope of orchestration-level variables: Private—Access to this orchestration is limited to the containing module. Public—Access to this orchestration is not limited. Internal—Access to this orchestration is limited to modules within the same project.

Typename	Determines the name of this orchestration within the containing module.

In This Section

- Adding Parameters to Orchestrations
- Adding Shapes to Orchestrations
- Making Orchestrations Transactional

How to Add Parameters to Orchestrations

You can specify what parameters your orchestration should take in the Orchestration View window. An orchestration can take the following items as parameters:

- Messages
- Variables (including objects)
- Correlation sets
- Role links
- Ports

Parameters can be passed between orchestrations as in parameters or out parameters. In parameters can be passed by value or by reference. Out parameters can only be passed by reference. Parameters can include variables, messages, correlation sets, role links, and ports.

To set orchestration parameters

1. In the Orchestration View window, use the **Orchestration Parameters** folder to add variables, messages, and ports.
2. For each item added to the **Orchestration Parameters** folder, use the Properties window to specify the **Direction** property:
   - In—A parameter passed in by value.
   - Ref—A parameter passed in by reference.
   - Out—A parameter passed out by reference.

To add a parameter to an orchestration

1. In the Orchestration View window, right-click the **Orchestration Parameters** folder and then click the kind of parameter you want.
2. For configured ports and role links, use the wizard to configure the parameter.

   —Or—

   For other parameter types, use the properties page to configure the parameter.

Parameter types

Parameters can be passed by value, as reference parameters, and as out parameters. When a parameter is passed by value to an orchestration, a copy of the data is made and used by the orchestration.

When you use a reference parameter, no copy is made. The memory location that contains the data is shared between the calling program and the orchestration, and the contents of this memory location can be modified by the orchestration. Such a modification means that the value of the parameter is changed not only in the orchestration, but also in the calling program.

An out parameter is similar to a reference parameter, but the orchestration cannot assume that it contains valid data when passed in; rather, the calling program expects the orchestration to assign a value to this parameter.

Rules for orchestration parameters

- You can pass only messages and variables (including objects) as out or reference parameters.
- You cannot pass out or reference parameters to an orchestration in a **Start Orchestration** shape.
- In parameters, including any role links and dynamic ports, must be definitely assigned before being passed to an orchestration.

Orchestration Shapes

Orchestration Designer is a visual tool for creating orchestrations. It provides several shapes that you can place on the design surface as visual representations of underlying actions, and they can help you to efficiently design and implement an orchestration.

Insufficient Configuration Smart Tag

The following table lists the available shapes, along with a brief description of the function of each shape.

Shape	Shape Name	Purpose
	Call Orchestration	Enables your orchestration to call another orchestration synchronously. For more information, see How to Use the Call Orchestration Shape .
	Call Rules	Enables you to configure a Business Rules policy to be executed in your orchestration. For more information see How to Use the Call Rules Shape .
	Compensate	Enables you to call code to undo or compensate for operations already performed by the orchestration when an error occurs. For more information, see How to Use the Compensate Shape .
	Construct Message	Enables you to construct a message. For more information, see How to Use the Construct Message Shape .
	Decide	Enables you to conditionally branch in your orchestration. For more information, see How to Use the Decide Shape .
	Delay	Enables you to build delays in your orchestration based on a time-out interval. For more information, see How to Use the Delay Shape .
	Expression	Enables you to assign values to variables or make .NET calls. For more information, see How to Use the Expression Shape .
	Group	Enables you to group operations into a single collapsible and expandable unit for visual convenience. For more information, see How to Use the Group Shape .
	Listen	Enables your orchestration to conditionally branch depending on messages received or the expiration of a timeout period. For more information, see How to Use the Listen Shape .
	Loop	Enables your orchestration to loop until a condition is met. For more information, see How to Use the Loop Shape .
	Message	Enables you to assign message values. For more information, see How to Use the Message Assignment Shape

	Assignment	.
	Parallel Actions	Enables your orchestration to perform two or more operations independently of each other. For more information, see How to Use the Parallel Actions Shape .
	Port	Defines where and how messages are transmitted. For more information, see How to Use Ports in Orchestrations .
	Receive	Enables you to receive a message in your orchestration. For more information, see How to Use the Receive Shape .
	Role Link	Enables you to create a collection of ports that communicate with the same logical partner, perhaps through different transports or endpoints. For more information, see How to Use the Role Link Shape .
	Scope	Provides a framework for transactions and exception handling. For more information, see How to Use the Scope Shape .
	Send	Enables you to send a message from your orchestration. For more information, see How to Use the Send Shape .
	Start Orchestration	Enables your orchestration to call another orchestration asynchronously. For more information, see How to Use the Start Orchestration Shape .
	Suspend	Suspends the operation of your orchestration to enable intervention in the event of some error condition. For more information, see How to Use the Suspend Shape .
	Terminate	Enables you to immediately end the operation of your orchestration in the event of some error condition. For more information see How to Use the Terminate Shape .
	Throw Exception	Enables you to explicitly throw an exception in the event of an error. For more information, see Using the Throw Exception Shape .
	Transform	Enables you to map the fields from existing messages into new messages. For more information, see How to Use the Transform Shape .

Adding Shapes to Orchestrations

This section describes the procedures for adding and removing shapes in your orchestration. For more information about the available shapes, see Orchestration Shapes.

To add a shape to an orchestration

1. In the Toolbox, on the **BizTalk Orchestrations** tab, drag the shape onto a connecting line on the Orchestration Design Surface.

   —Or—

2. Right-click the connecting line or the shape placeholder where you want to add the shape, point to **Insert Shape**, and then click the shape name you want to add.

   A Smart Tag appears on shapes that are not yet fully configured.

To remove a shape from an orchestration

1. Right-click the shape on the design surface, and then click **Delete**.

   —Or—

2. Select the shape and press the **DELETE** key.

Making Orchestrations Transactional

You can configure your orchestration as a transaction, in much the same way that you can configure a **Scope** shape as a transaction. For more information, see Transactions.

In This Section

- Configuring Transactional Properties on an Orchestration
- Adding Custom Compensation to an Orchestration

How to Configure Transactional Properties on an Orchestration

An orchestration can be treated as an atomic transaction, a long-running transaction, or neither.

To make your orchestration an atomic transaction

1. In the Orchestration View window, select **Orchestration Properties**.
2. In the Properties window, select **Atomic** in the drop-down for the **Transaction Type** property.

   Batch, **Compensation**, **Isolation Level**, **Retry**, **Timeout**, and **Transaction Identifier** appear as properties in the Properties window, just as they do for any scope. For more information on these properties, see Using the Scope Shape.

To make your orchestration a long-running transaction

1. In the Orchestration View window, select **Orchestration Properties**.
2. In the Properties window, select **Long Running** in the drop-down for the **Transaction Type** property.

   Compensation, **Timeout**, and **Transaction Identifier** appear as properties in the Properties window. For more information on these properties, just as they do for any scope. For more information on these properties, see Using the Scope Shape

How to Add Custom Compensation to an Orchestration

An orchestration transaction that is configured as long running can have custom compensation code to reverse or undo the effects of the transaction. If the orchestration has completed successfully and has been called by another orchestration, the calling orchestration can invoke its compensation block using a **Compensate** shape.

To specify that an orchestration will use custom compensation

1. In the Orchestration View window, select **Orchestration Properties**.
2. In the Properties window, select **Custom** in the drop-down for the **Compensation** property.

   The **Compensation** tab appears next to the **Orchestration** tab at the bottom of the Design Surface.

To design custom compensation for an orchestration

1. Click on the **Compensation** tab at the bottom of the Design Surface.

   The **Compensation Design Surface** appears.

2. Add shapes to the **Compensation Design Surface** just as you would in the **Orchestration Design Surface**.

   For more information, see Adding Shapes to Orchestrations.

How to Use the Select Artifact Type Dialog Box

An *item* is used to configure elements of an orchestration in Orchestration Designer. Examples of items are schemas, maps, pipelines, port types, and multi-part message types. When you develop an orchestration and its constituent parts such as port shapes, transform shapes, and messages, you may need to refer to items that do not reside in the current orchestration, but are in the current project or another project that has been compiled into a BizTalk Server assembly. You use the **Select Artifact Type** dialog box to locate and then specify items when configuring an element within an orchestration.

The **Select Artifact Type** dialog box is available from many locations in Orchestration Designer. You can select <Select from referenced assembly> in a drop-down list that displays configuration options; clicking this text opens the **Select Artifact Type** dialog box.

Before you can select an item, you must first select the element you want to configure.

You can use the **Select Artifact Type** dialog box for the following specific purposes:

Action	**Purpose**
Select a pipeline	Select a pipeline for the pipeline property when configuring a port for direct (early) binding.
Select a map	Select a map to use with a **Transform** shape.
Select a schema	Select schemas in the project when creating multi-part message types.
Select a port type	Refer to existing port types when creating a port.
Select a multi-part message type	Refer to existing multipart types when creating messages.
Select a .NET type	Refer to existing .NET types when creating variables or

	messages.

Reference pane

The reference pane of the **Select Artifact Type** dialog box displays references in the current project and in other available assemblies. To select a reference in this pane, click it. To expand a container in this pane (such as the **Assemblies** container), click the plus sign (+) beside it.

Item pane

The item pane of the **Select Artifact Type** dialog box displays the items contained in the reference currently selected in the reference pane. The item pane has two columns, **Item** and **Qualified Name**, which display information about the items in the current reference.

To use the Select Artifact Type dialog box

1. Expand the **My References** node in the left pane. The pane displays available projects and assemblies.

2. Expand the **Assemblies** node. One or more assemblies are listed, such as SYSTEM.DLL and MICROSOFT.BIZTALK.PIPELINES.DLL.

3. Click an assembly. If the assembly contains items, they are displayed in the right pane. The qualified name of an item is displayed in the right column of the right pane.

4. To select an item in the right pane, click it and then click **OK** to exit the **Select Artifact Type** dialog box. This assigns that item as the type for the selected element. To close the **Select Artifact Type** dialog box without selecting and assigning an item, click **Cancel**.

BizTalk Orchestrations Tab, Toolbox

The Visual Studio Toolbox contains tabs, which contain tools. The Toolbox always displays two tabs, the **General** tab and the **Clipboard Ring** tab.

When you open a project that uses Orchestration Designer (a BizTalk Server orchestration), the Toolbox also displays the **BizTalk Orchestrations** tab. The tools on this tab, called *shapes*, are visible and available for use when the **BizTalk Orchestrations** tab is expanded.

To use an orchestration on the BizTalk **Orchestrations** tab, drag it onto the design surface of the active BizTalk orchestration. You can drag **Port** shapes and **Role Link** shapes onto either the left Port Surface or the right Port Surface. Onto the central

Process Area you can drag all other shapes—shapes that describe the process flow of the orchestration.

For more information about using the Toolbox and **Toolbox** tabs, see "Managing Tabs and Items in the Toolbox" in the Visual Studio .NET documentation.

Designing Orchestration Flow

This section discusses the various actions you can use to direct the flow of control in your orchestrations.

In This Section

- Sending and Receiving Messages in Orchestrations
- Using Flow Control Shapes
- Nesting Orchestrations

Sending and Receiving Messages in Orchestrations

When sending or receiving messages, you need to know what types of messages you are working with, and how to ensure that they get to the right place. For conceptual information on working with messages, see Working with Messages in Orchestrations.

In This Section

- Using the Send Shape
- Using the Receive Shape

How to Use the Send Shape

Send shape

If you expect to receive an indirect or asynchronous response (not using a request-response port) to the message that you have sent, you need to correlate the message with the currently running instance of the orchestration, so that the respondent can get the response to the correct instance. You can apply a following correlation set to the **Send** shape for a previously initialized correlation, or you can apply an initializing correlation set. For more information, see Correlating Messages with Orchestration Instances.

To configure a Send shape

1. Set a message and a port operation.

a. In the Orchestration View window, verify that your orchestration has both a message and a port operation defined for the multi-part message type being sent.

b. In the Properties window, select the message to send from the **Message** property drop-down list.

c. In the Properties window, select the port operation that sends the message from the **Port Operation** drop-down list.

—Or—

Drag the send connector from the **Send** shape to the port socket that sends the message.

2. Specify correlation sets to restrict the messages the **Send** shape will send or to initialize the values in a correlation set.

a. For each correlation set you want to use, check a correlation set from the drop-down on the **Following Correlation Sets** property.

b. For each correlation set that you want to initialize, check a correlation set from the drop-down on the **Initializing Correlation Sets** property.

Delivery Notification

To test whether you have successfully sent a message over a send port, complete the following steps:

1. Put your Send shape in a non-transactional, long-running or atomic scope.

2. On your send port, set the DeliveryNotification property to **Transmitted**.

3. Add a catch handler to your scope to handle a DeliveryFailureException.

The orchestration waits for acknowledgment at the end of the enclosing non-atomic scope, or the end of the orchestration, to receive the acknowledgment.

How to Use the Receive Shape

Receive shape

A **Receive** shape can be used to start an orchestration. If you set the **Activate** property to **True**, the runtime engine will test an incoming message to see whether it is of the right type and, if a filter has been applied, whether the filter expression is satisfied. If the criteria for receipt of the message are met, the runtime engine creates and runs a new orchestration instance, and the **Receive** shape receives the message.

If you expect to receive an indirect or asynchronous response (not on a request-response port) to a message that you have previously sent, you need to correlate the message with the currently running instance of the orchestration, so that the respondent can get the response to the correct instance. You can apply an initializing correlation set to the Receive shape if you plan to do subsequent correlation on values in the incoming message, or a following correlation set for correlating using a previously initialized correlation set. For more information, see Correlating Messages with Orchestration Instances.

To configure a Receive shape

1. Set a message and a port operation.

   a. In the Orchestration View window, verify that your orchestration has both a message and a port operation defined for the message type being received.

      In the Properties window, select the message to receive from the **Message** property drop-down list.

   b. In the Properties window, select the port operation to receive the message from the **Port Operation** drop-down list.

      —Or—

      Drag the receive connector from the **Receive** shape to the port socket that will receive the message.

2. Specify that the **Receive** shape will activate the orchestration.

3. In the Properties window, set the Activate property to True.

   a. In the Properties window, click the **Ellipsis** (**...**) button for the Filter Expression property to create a filter to restrict the messages that this **Receive** shape accepts.

      —Or—

      Right-click the **Receive** shape and then click **Edit Filter Expression**.

   b. The **Filter Expression** dialog box appears. Use this dialog box to create one or more filter expressions. For more information, see Filter Expression Dialog Box and Using Filters to Receive Messages.

4. Specify correlation sets to restrict the messages the **Receive** shape accepts.

- For each correlation set you want to follow, check a correlation set from the drop-down on the **Following Correlation Sets** property.
- For each correlation set that you want to initialize, check a correlation set from the drop-down on the **Initializing Correlation Sets** property.

Using the Filter Expression Dialog Box

You use the **Filter Expression** dialog box to specify filter expressions for **Receive** shapes. You use filter expressions to express conditions under which a **Receive** shape receives messages.

Filter Expression grid control

You build a filter expression by using this grid control to define the predicates that make up the expression. You can add, edit, and delete predicates from the cells of the grid. This grid control has four columns: Property, Operator, Value, and Grouping.

- **Property.** You can type a property reference, or select one from the cell's drop-down list. The list contains properties on the incoming message.
- **Operator.** You can type in this cell, or select an operator from the drop-down list. Possible selections are:

Operand	Meaning
==	Is equal to
!=	Is not equal to
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
Exists	Exists

- **Value.** Cells in the **Value** column can hold any constant that you type in: a string-literal, an integer-literal, or null.
- **Grouping.** Use this column to control predicate grouping. Filter expressions are always expressed in Disjunctive Normal Form (DNF) so grouping can be determined automatically. AND means the predicate is to be grouped with the predicate following it, while OR means the predicate is separate from the predicate in the next row. Gray brackets to the left of the grid control appear

when predicates are grouped together. Predicate groups cannot be nested. If you do not specify a value in this cell, the value of the cell defaults to AND.

Hint label

This field provides user guidance. The label text changes depending on which column contains the active cell. The text displays the column name followed by guidance text as follows:

- **Property.** Please select a property on the incoming message from the list.
- **Operator.** Select an operator to compare the Property with the Value.
- **Value.** Select a message property from the list, or type in a literal value.
- **Grouping.** Specify how this row is to be grouped with the next row. 'AND' will join the rows, and 'OR' will separate them.

Move Up button

Click this to move a selected row up. (First select a row by clicking the *right arrow* (**>**) button at the left side of the grid control.)

Move Down button

Click this to move a selected row down. (First select a row by clicking the *right arrow* (**>**) button at the left side of the grid control.)

Subscription Expression field

This read-only text box shows the expression as you are building it.

Using Flow Control Shapes

Orchestration Designer provides a number of shapes that you can use to control the flow of your orchestration, either explicitly by using the **Parallel Actions** shape or conditionally by using the **Decision**, **Listen**, or **Loop** shapes. It also provides the **Group** shape to help organize your orchestration.

For more information, see Orchestration Shapes and Designing Orchestration Flow

In This Section

- Using the Decide Shape
- Using the Parallel Actions Shape
- Using the Listen Shape

- Using the Loop Shape
- Using the Delay Shape
- Using the Suspend Shape
- Using the Terminate Shape
- Using the Group Shape

How to Use the Decide Shape

Decide shape

Each branch of a **Decide** shape, except the **else** branch, has a rule associated with it. You can use BizTalk Expression Editor to create a Boolean expression in the rule that is evaluated for the execution of that branch. Because the **else** branch implies the negation of the Boolean expression in the previous branch, it does not have an expression associated with it.

Below the rule or **else** clause, a branch of a **Decide** shape can contain additional shapes, just like any other part of the orchestration.

To configure a Decide shape

1. If BizTalk Expression Editor is not visible, right-click the rule and click **Edit Boolean Expression**, or in the Properties window, click the **Ellipsis** (**...**) button for the **Expression** property.
2. In BizTalk Expression Editor, create a Boolean expression for each branch except the **Else** branch. For more information, see BizTalk Expression Editor.

To add a branch to a Decide shape

1. Right-click the **Decide** shape, and then click **New Rule Branch**.

   —Or—

2. Drag a new shape between two existing branches.

How to Use the Parallel Actions Shape

Parallel Actions shape

Synchronization of data access

It is possible that more than one branch of a **Parallel Actions** shape will attempt to access the same data. To avoid errors, place any shapes that access the data inside synchronized scopes. You can specify in the properties of a **Scope** shape that it is synchronized or not synchronized. For more information, see Scopes.

To add a branch to a Parallel Actions shape

1. Right-click the **Parallel Actions** shape, and then click **New Parallel Branch**.

   —Or—

2. Drag a new shape between two existing branches.

How to Use the Listen Shape

Listen actions enable applications to wait for one of several messages on one or more ports, or to stop waiting after a specified time-out interval, and branch based upon the results.

Listen shape

You can add as many branches as you like. You can place any other shape below the initial **Receive** or **Delay** shape.

It is possible to use an activation receive in a **Listen** shape. If one branch of the **Listen** shape contains an activation receive, then all branches must contain activation receives, and no timeout can be used. The activation receive must be the first action in each branch.

You can use the **Listen** shape to branch orchestration flow based on the occurrence of one or more events. The first shape in each branch must be either a **Delay** or a **Receive** shape. The first branch that meets its condition—the occurrence of a time-out for a **Delay** shape or the receipt of a message for a **Receive** shape—will execute. You can add additional branches if needed.

To configure a Listen shape

1. Select a branch.
2. In the Properties window, specify the **Branch Type** property.

   —Or—

   Drag a **Delay** or **Receive** shape onto the branch.

To add a branch to a Listen shape

1. Right-click the **Listen** shape and then click **New Listen Branch**.

How to Use the Loop Shape

Loop shape

To configure a Loop shape

1. If BizTalk Expression Editor is not visible, right-click the **Loop** shape and click **Edit Boolean Expression**, or in the Properties window, click the **Ellipsis** (**...**) button for the **Expression** property

2. In BizTalk Expression Editor, create a Boolean expression for the iteration condition. For more information, see BizTalk Expression Editor.

How to Use the Delay Shape

Delay shape

There are two ways to specify the timeout for a **Delay**:

- You can use **System.DateTime**, which causes your orchestration to pause until the specified date and time is reached.

  System.DateTime.UtcNow.AddSeconds(60)

- You can use **System.TimeSpan**, which causes your orchestration to pause for the specified length of time.

  System.TimeSpan(0, 1, 0)

Note If your **Delay** shape is inside a **Listen** shape, you do not need to add a semicolon at the end of the expression.

For more information on **System.DateTime** and **System.TimeSpan**, see "DateTime Structure" and "TimeSpan Structure" in the Visual Studio 2005 Combined Collection.

To configure a Delay shape

1. If BizTalk Expression Editor is not visible, right-click the **Delay** shape and click **Edit Delay**, or in the Properties window, click the **Ellipsis** (**...**) button for the **Expression** property

2. In BizTalk Expression Editor, create an expression that returns a **System.DateTime** object or a **System.TimeSpan** object. For more information, see BizTalk Expression Editor**.**

How to Use the Suspend Shape

Suspend shape

When an orchestration instance is suspended, an error is logged. You can specify a message string to accompany the error to help the administrator diagnose the situation.

All of the state information for the orchestration instance is saved, and is reinstated if and when the administrator resumes the orchestration instance.

To configure a Suspend shape

1. You can use the **Error Message** property to specify text that you want to be logged when a **Suspend** shape is encountered. This text may be a literal string, or an expression that evaluates to a **System.String**.

How to Use the Terminate Shape

Terminate shape

The Terminate shape is used to end an orchestration instance. You can specify a message string to accompany the shape when viewed in HAT.
To configure a Terminate shape

1. You can use the **Error Message** property to specify text that you want to associate with the shape when viewed in HAT. This text may be a literal string, or an expression that evaluates to a **System.String**.

How to Use the Group Shape

Group shape

Annotations and placeholders

You can use the **Group** shape as a placeholder for functionality yet to be added, or you can use it to make annotations about what actions take place within it.

The name property on the **Group** shape can be up to 512 characters long. You can type annotations into the **Name** property and, when the shape is collapsed, the entire name will be displayed. When the shape is expanded, the name is truncated.

Nesting Orchestrations

You can use the **Call Orchestration** shape and the **Start Orchestration** shape to invoke one orchestration from another. You can nest orchestrations to arbitrary depths as well: for example, a called orchestration can call a third orchestration, which can call a fourth, and so on.

For more information, see Adding Parameters to Orchestrations and Designing Orchestration Flow.

In This Section

- Using the Call Orchestration Shape
- Using the Start Orchestration Shape

How to Use the Call Orchestration Shape

The **Call Orchestration** shape can be used to synchronously call an orchestration that is referenced in another project. This allows for reuse of common orchestration workflow patterns across BizTalk projects. When you invoke another nested orchestration synchronously with the **Call Orchestration** shape the enclosing orchestration waits for the nested orchestration to finish before continuing.

You can specify parameters that will be passed to the nested orchestration. Parameters can be messages, variables, port references, role links, or correlation sets. Passed-in port references, role links, and correlation sets all perform like self-addressed envelopes: they supply the nested orchestration information it can use to send information back to the enclosing orchestration.

To configure a Call Orchestration shape

1. Right-click the **Call Orchestration** shape, and then click **Configure**.
2. Use the **Call Orchestration Configuration** dialog box to configure the **Call Orchestration** shape. For more information, see Call Orchestration Configuration Dialog Box.

For the referenced orchestration to be callable, ensure that the following properties have been configured for the called orchestration:

- Set the **Type Modifier** property to **Public** for the called orchestration. To set the **Type Modifier** property for an orchestration to **Public**, open the orchestration in Microsoft Visual Studio 2005, click the green start shape at the top of the orchestration to display the **Orchestration Properties** dialog and set the **Type Modifier** property to **Public**.
- Set the **Activate** property of the initial receive shape in the orchestration to **False**.

How to Use the Call Orchestration Configuration Dialog Box

You use the **Call Orchestration Configuration** dialog box to configure a **Call Orchestration** shape when you:

- Specify an orchestration that you want the shape to call.
- Specify arguments to pass to the orchestration (if it is a parameterized orchestration) that is called.

Orchestration Selection drop-down list box

Click the Down arrow in the drop-down list box to view available services and select one. This list contains all the services that can be called from the current orchestration, including referenced assemblies.

Orchestration Parameters grid control

You specify the arguments to pass to a parameterized orchestration by using the **Orchestration Parameters** grid control. The grid has four columns: Variables in Scope, Parameter Name, Parameter Type, and Parameter Direction. You can make changes only in the first column; the other columns are read-only.

When you select a valid orchestration, its parameters populate the parameter name, type and direction columns of the grid control. You then select the variables in each row to pass as arguments. You select these variables from a drop-down list present in each cell in the Variables in Scope column. This list displays all the available variables of the type specified in the adjacent Parameter Type cell. If only one object of that type is available, the Variables in Scope cell is automatically populated with that object. You can also type in a Variables in Scope cell to select a variable that is available in the drop-down list.

If an orchestration you are calling has no defined parameters, the grid control in this dialog box is unavailable.

To use the Call Orchestration Configuration dialog box

1. Using the **Orchestration Selection** drop-down list box, select an orchestration from the list.
2. Using the **Orchestration Parameters** grid control, specify arguments to pass to the orchestration—as specified in the **Orchestration Selection** drop-down list box—that is called. You specify these arguments in the cells of the Variable column, one variable per cell, by typing the name of a variable or clicking a variable from a drop-down list in a cell.
3. To configure the **Call Orchestration** shape according to the service and arguments that you specified in the dialog box, click **OK**. To close the **Call Orchestration Configuration** dialog box without making any changes to the **Call Orchestration** shape, click **Cancel**.

How to Use the Start Orchestration Shape

The **Start Orchestration** shape is similar to the **Call Orchestration** shape, but you invoke another orchestration asynchronously with the **Start Orchestration** shape—that is, the flow of control in the invoking orchestration proceeds beyond the invocation, without waiting for the invoked orchestration to finish its work.

You can specify parameters that will be passed to the called orchestration. Parameters can be messages, variables, port references, role links, or correlation sets. Passed-in port references, role links, and correlation sets all perform like self-addressed envelopes: they supply the invoked orchestration information it can use to send information back to the enclosing orchestration. The **Start Orchestration** shape can only take in parameters; it cannot take out or reference parameters.

The **Start Orchestration** shape is the only shape in which you can reverse the polarity on a port being passed as a parameter—for example a *uses* port can be passed in to a started orchestration, but the started orchestration can be treated as an *implements* port. Note that this can only be done with ports that use direct binding.

To configure a Start Orchestration shape

1. Right-click the **Start Orchestration** shape and then click **Configure**.
2. Use the **Start Orchestration Configuration** dialog box to configure the **Start Orchestration** shape. For more information, see Start Orchestration Configuration Dialog Box.

How to Use the Start Orchestration Configuration Dialog Box

You use the **Start Orchestration Configuration** dialog box to configure a **Start Orchestration** shape when you:

- Specify an orchestration that you want the shape to start.
- Specify arguments to pass to the orchestration (if it is a parameterized orchestration) that is started.

Orchestration Selection drop-down list box

Click the Down arrow in the drop-down list box to view available orchestrations and select one. This list contains all the orchestrations that can be started from the current orchestration, including referenced assemblies.

Orchestration Parameters grid control

You specify the arguments to pass to a parameterized orchestration by using the **Orchestration Parameters** grid control. The grid has four columns: Variables in Scope, Parameter Name, Parameter Type, and Parameter Direction. You can make changes only in the first column; the other columns are read-only.

When you select a valid orchestration, its parameters populate the parameter name, type, and direction columns of the grid control. You then select the variables in each row to pass as arguments. You select these variables from a drop-down list present in each cell in the Variables in Scope column. This list displays all the available variables of the type specified in the adjacent Parameter Type cell. If only one object of that type is available, the Variables in Scope cell is automatically populated with that object. You can also type in a Variables in Scope cell to select a variable that is available in the drop-down list.

If an orchestration you are executing has no defined parameters, the grid control in this dialog box is unavailable.

To use the Start Orchestration Configuration dialog box

1. Using the **Orchestration Selection** drop-down list box, select an orchestration from the list.

2. Using the **Orchestration Parameters** grid control, specify arguments to pass to the orchestration—as specified in the **Orchestration Selection** drop-down list box—that is started. You specify these arguments in the cells of the Variable column, one variable per cell, by typing the name of a variable or clicking a variable from a drop-down list in a cell.

3. To configure the **Start Orchestration** shape according to the service and arguments that you specified in the dialog box, click **OK**. To close the **Start Orchestration Configuration** dialog box without making any changes to the **Start Orchestration** shape, click **Cancel**.

How to Use Ports in Orchestrations

Ports specify how your orchestration will send messages to and receive messages from other business processes. Each port has a type, a direction, and a binding, which together determine the direction of communication, the pattern of communication, the location to or from which the message is sent or received, and how the communication takes place.

Depending on these factors, a port may have associated with it a URI (a physical location), a transport (either FILE, HTTP, SOAP, SMTP or BizTalk Message Queuing), a send pipeline to prepare a message for sending (for example, by assembling, encrypting, compressing, or performing some other action on it), and a receive pipeline to prepare a received message for processing (for example, by disassembling, decrypting, or decompressing it).

You add ports to an orchestration in much the same way that you add controls to a Web Form or Windows Form. You can also add ports by using the Orchestration View window.

To add a new port

1. Right-click a **Port Surface** or a **Role Link**, and then click **New Port**.

   —Or—

   In the Orchestration View window, right-click **Ports** and then click **New Port**.

   A DesignTip appears on ports that are not yet fully configured.

To add and configure a new port

1. From the **BizTalk Orchestrations** tab of the Toolbox, drag the **Port** shape onto a **Port Surface** or a **Role Link**.

   —Or—

   Right-click a **Port Surface** or a **Role Link**, and then click **New Configured Port**.

   —Or—

   In the Orchestration View window, right-click **Ports** and then click **New Configured Port**.

   The Port Configuration Wizard appears.

2. Follow the steps in the wizard to configure the port. For more information, see Port Configuration Wizard.

To remove a port

1. Right-click the port to remove, and then click **Delete**.

To configure a port by using the Port Configuration Wizard

1. Right-click the port to configure, and then click **Configure Port**.

2. Follow the steps in the Port Configuration Wizard to configure the port. For more information, see Port Configuration Wizard.

To configure a port manually by using the Properties window

1. Select the port to configure.

2. In the Properties window, specify the following properties:

Property	Description
Port Type	Determines the communication pattern, operations, and multi-part message types associated with a port.
Communication Direction	Determines whether this orchestration is the sender or the receiver for this communication.
Communication Pattern	Determines if this port is used for request-response or one-way communication. (This property is determined by the **Port Type** property and is read-only.)
Binding	Determines how a message gets to its destination: Direct—Communication is with another orchestration. Dynamic—Communication is with an endpoint that is determined at run time. Specify later—Communication is with an endpoint that is determined by an administrator at configuration time. Specify now—Communication is with an endpoint that is known at design time.
Identifier	The name used for this port in the orchestration.
Delivery Notification	For send ports, determines whether want to receive an acknowledgment when a message is sent successfully. For more information, see "Delivery Notification" under Using the Send Shape.

3. The remaining properties to specify are determined by the **Binding** property:

- Direct binding—Used when communicating directly with another orchestration.

Property	Description
Partner Orchestration Port	Determines to which port the partner orchestrations will be directly bound.

- Dynamic binding—Used when communicating with an endpoint that is determined at run time.

Property	**Description**
Receive Pipeline	Determines which pipeline to use for incoming messages.
Send Pipeline	Determines which pipeline to use for outgoing messages.

- Specify later—Used when communicating with an endpoint that is configured by an administrator.
- Specify now—Used when communicating with an endpoint that is known at design time.

Property	**Description**
Receive Pipeline	Determines which pipeline to use for incoming messages.
Send Pipeline	Determines which pipeline to use for outgoing messages.
Transport	Determines which transport to use for sending messages.
URI	Determines where to send messages.

To add a port operation

1. Right-click the port to which you want to add an operation, and then click **New Operation**.

To remove a port operation

1. Right-click the port operation to remove, and then click **Delete**.

In This Section

- Working with Port Types
- Communication Pattern
- Communication Direction
- Port Bindings

How to Work with Port Types

A port type consists of a communication pattern, a set of operations (requests or responses), and the message types that those operations can work on. The pattern can be either one-way or request-response (two-way), and all operations defined on that port type must use the same pattern. Note that port types are direction-agnostic: direction is specified on individual ports.

The scope of a port type is defined by the **Type Modifier** property. A port type can be public, private, or internal. If it is public, it is visible to anyone interacting with the orchestration. If it is private, it is visible to other orchestrations within the same project and namespace. If it is internal, the port type is visible only within the project. Since a port type definition includes message types, the scope of the message type must encompass that of any port type that uses it.

To add a request-response port type

1. In the Orchestration View window, right-click **Port Types** and then click **New Request-response Port Type**.

   The **Port Types** node expands, if collapsed, and a new request-response port type is added with one default operation.

2. Specify a name for the port type.
3. Define one or more port operations.

   You can name your port operations, but when you are selecting them from another project, you will see them only as "Request" and "Response." If you select a port operation from another project, verify that it has the correct message type.

To add a one-way port type

1. In the Orchestration View window, right-click **Port Types** and then click **New One-way Port Type**.

   The **Port Types** node expands, if collapsed, and a new one-way port type is added with one default operation.

2. Specify a name for the port type.
3. Define one or more port operations.

To add a Web port type

1. Add a project reference to an assembly containing a proxy class for a Web service. For more information, see Creating Web Ports.

To remove a port type

1. In the Orchestration View window, right-click the port type to delete, and then click **Delete**.

To set the type modifier for a port type

1. In the Properties window, set the following property:

Property	Description
Type Modifier	Determines the scope of the port type: Private—Access to this port type is limited to the containing module. Public—Access to this port type is not limited. Internal—Access to this port type is limited to modules within the same project.

Communication Pattern

Each *port type* has a communication pattern. The communication pattern determines whether one-way or two-way (request-response) transmissions can take place on ports of the given type. For more information, see Port Configuration Wizard**.**

Communication Direction

Each *port* has its own communication direction. The communication direction is used in combination with the communication pattern of the port's type to complete the definition of how a port can be used. The communication direction, or polarity, determines in which direction messages will be transmitted over that port.

If the port's type has a one-way communication pattern, its communication direction can be either Send or Receive. If the port's type has a two-way (request-response) communication pattern, its communication direction can be Send-Receive, in which a request is sent and a response is received, or Receive-Send, in which a request is received and a response is sent.

Port Bindings

A port binding is the configuration information that determines where and how a message will be sent or received. Depending on type, bindings might refer to physical locations, pipelines, or other orchestrations.

Binding at Deployment Time

You can bind your port to a receive location or to a send port. If you do not have all of the information you need to specify a physical location, you can select the "Specify Later" port binding option in Orchestration Designer, and you only need to specify the port type that describes the port. The information about the actual location will be specified separately after the application has been deployed, either by an administrator in BizTalk Explorer or programmatically, perhaps using script.⊟Binding at Design Time

You can select the "Specify Now" port binding option in Orchestration Designer to specify at design time what pipeline you want to use and the location you want to communicate with. This is useful if you know in advance the source or destination of transmitted messages.

Direct Binding

Your orchestration can communicate directly with another orchestration by using direct binding. With direct binding, the message appears to go directly from one orchestration to another. The engine accomplishes this by adding the originating orchestration's port to messages sent from the orchestration. The engine also creates a receive port on the partner orchestration. The receive port filters for messages containing the originating orchestration's port.

There are three flavors of direct binding. You can use direct binding to bind a send or receive action to one of the following:

- A known port on a partner orchestration.
- A port with routing information defined by a message subscription on incoming messages, based on a filter expression in a **Receive** shape.
- A self-correlating port, in which the orchestration engine will generate a correlation token on a message that is particular to the orchestration instance, and you do not have to specify any correlation sets yourself.

Dynamic Binding

If you will not know the location of a communication until run time, you can use dynamic binding for a send port. The location might, for example, be determined from a property on an incoming message.

For information about how to dynamically assign values to ports, see Assigning to Dynamic Ports.

Web Ports

If your project contains a reference to a Web service, Orchestration Designer will detect it and will make available a corresponding Web port type. You simply add a port to your orchestration and assign it an existing Web port type, and you will have a complete Web port. For more information, see Creating Web Ports.

How to Run the Port Configuration Wizard

You use the Port Configuration Wizard to create and configure a port in Orchestration Designer. A port must have a port type associated with it, and you use the wizard to select an existing port type or to create a new port type. For more information about ports and port types, see Using Ports in Orchestrations.

To Start the wizard

1. Right-clicking a port (on the design surface or in the Orchestration View window) and clicking **Configure Port**.

2. Running Smart Tag items whose associated actions cause a port to be created.

3. Selecting the **New Configured Port Parameter** command from the shortcut menu of the Orchestration Parameters folder in the Orchestration View window.

To Run the Wizard

1. Open the Port Configuration Wizard.

2. Specify port information. To help you, the wizard displays several pages. After completing each page, move to the following one by clicking **Next**, or move to the preceding one by clicking **Back**.

   - **Port Properties**. Type in a name for the port.

   - **Select a Port Type.** On this page, you first select whether you want a **New Port Type** or an **Existing Port Type**. If you select **Existing Port Type**, you then use a tree control to choose which existing port type to assign.

     If you select **New Port Type**, you then need to type the name of the port type in the **Name** text box, or accept the suggested default name. You also select the port type's communication pattern (one-way or request-response) and any access restrictions to impose on the new port type.

   - **Port Binding**. On this page you specify the direction of communication, also known as the *polarity*, and the binding type of the port.

     The polarity choice you make depends in part on the communication pattern of the port type that you selected on the preceding page of the wizard, **Select a Port Type**. Your choices are summarized in the following table:

Port direction	**Communication pattern**	**Direction of communication to choose on Port Binding page**
Send	One-way	I will always be sending messages on this port.
Receive	One-way	I will always be receiving messages on this port.
Send-Receive	Solicit-response	I will be sending a request and receiving a response.

Receive-Send	Request-response	I will be receiving a request and sending a response.

- You have a choice of four different binding types: Specify later, Specify now, Dynamic, and Direct. Each choice displays a different set of configuration options, as summarized here:

  Specify later. After selecting this option, you make no further configuration choices in the wizard because binding information is not determined at design time. Typically it is added by an Administrator at deployment time.

  Specify now. You can specify at design time a URI that defines the entity to which the port is to be bound. You also need to select from a list of transport types that includes options such as BizTalk Message Queuing, FILE, and HTTP. Finally, you select a receive pipeline and a send pipeline from a list of available pipelines. The list for each includes all pipelines in the current project plus the option of selecting a pipeline from a referenced assembly, which displays the **Select Artifact Type** dialog box.

  Dynamic. This option has similar choices to **Specify now**, but it is available only for a send port. You can specify a send pipeline to use. You can specify the port separately in an **Expression** shape so that it is assigned at run time,

  Direct. For direct binding, you can either select a port to connect to, to do routing based on filter expressions on incoming messages in the Message Box database, or to make it a self-correlating port. You can select a port from a drop-down list box.

  For more information, see Port Bindings.

3. Then move through the wizard again and click **Finish** when the information displayed on the final page matches the changes you want to make.

Using Role Links

You can group together a set of ports through which you will communicate with a business partner or other party—such as a supplier, a shipper, or an internal cost center—that functions cooperatively with your orchestration to carry out a well-defined interaction. If your role link represents a supplier, and you decide to change suppliers, an administrator can use BizTalk Explorer to make the change, and your orchestration does not have to be redesigned, recompiled, and redeployed.

A role link is an abstraction for the interaction between your orchestration and the external party, and the roles that each side plays. This abstraction of endpoints frees you to do such things as routing messages to different partners based on message content, or the results of a database lookup in your orchestration.

Here are the basic tasks you will need to complete to use role links in your orchestration:

- Create parties and ports and associate them with each other.

  For information about how parties work, see Managing Business Processes Using BizTalk Explorer.

- Add role links to your orchestration and add the ports to the role links.

- Associate the ports with **Send** and **Receive** shapes.

- When your role link is configured as a "using" or "consumer" role link, you will also need to set a destination ID.

A role link is an instance of a role link type, which consists of either one or two roles.

For more information, see Role Link Wizard, Using Ports in Orchestrations, and Managing Business Processes Using BizTalk Explorer

Roles

Roles are collections of port types, and each role either *uses* a service (as a consumer) or *implements* a service (as a provider). A role link can include either a uses or implements role, or one of each.

A uses role consumes the services provided by an implements role. When you define a role link with one or both of these roles, it is assumed that the complementary role is being fulfilled by the partner with whom you are linking. If a role uses a service, it sends a message to a party that implements a service; conversely, if a role implements a service, it receives a message.

Role link properties

A role link has a **SourceParty** property, a **DestinationParty** property, and an initiating role.

An initiating role specifies on which role the first communication occurs, and which will therefore initiate the role link by setting the value of the **DestinationParty** property.

If the initiating role is a consumer or *uses* role, you explicitly set the **DestinationParty** property (once and only once) in your orchestration.

If the initiating role is an *implements* role, the **DestinationParty** property will be initialized automatically by the receive.

The **SourceParty** property is read-only, and will be supplied through a trusted pipeline component based on the NT identity of the sender or a certificate associated with the message.

Initializing a consumer role link

To initialize a consumer or "uses" role link for a send action, you set the value of the DestinationParty, as in the following example, where ConfirmOrder is a role link, and PartnerName and OrganizationName are parameters of a party:

```
ConfirmOrder(Microsoft.XLANGs.BaseTypes.DestinationParty)     =      new
Microsoft.XLANGs.BaseTypes.Party(PartnerName, OrganizationName);
```

In This Section

- Using the Role Link Shape

How to Use the Role Link Shape

Role Link shape contains placeholders for an implements role and a uses role. It can include one of either, or one of each.

You can add port types directly to a **Role Link** shape, using either existing roles or new roles, and existing or new port types.

To add a Role Link

1. Right-click a Port Surface and then click **New Role Link**.

   —Or—

   From the **BizTalk Orchestrations** tab of the Toolbox, drag the **Role Link** shape onto a Port Surface.

To remove a Role Link

1. Right-click the **Role Link** to remove, and then click **Delete**.

How to Use the Role Link Wizard

The Role Link Wizard enables you to create a new role link or modify an existing one. You can use it to set or view the name, type, and access restriction of the role link, as well as the implements role and the uses role that compose the role link type. To understand how role links work, see Using Role Links.

Role link name: The role link name is filled in for you and is either the current name of an existing role link that you are configuring, or an automatically generated name if you are creating a new role link. In either case you can modify the name.

Role link type: You can select an existing role link type, or create a new one. Whether you are configuring a new or existing role link type, you can specify how your orchestration participates in the service.

Role link usage: If you create a new role link type, both the implements and uses roles are automatically created for you, and called "Provider" and "Consumer", respectively. You can select the role that reflects how your orchestration participates in the service.

After you have completed the steps in the wizard, you can remove a role if you like—a role link type must contain one of either role type or one of each role type. When you click **OK**, unconfigured roles are created corresponding to each name. You can also select port types for the roles in the Types window.

How to Work with Messages in Orchestrations

A message is the basic unit of communication in an orchestration. It contains one or more parts along with context data that describes properties of the message and its contents.

To add a message variable

1. In the Orchestration View window, right-click **Messages** and then click **New Message**.

   The **Messages** folder expands, if collapsed, and a new message is added.

2. Name the message.

3. Specify the message type.

To remove a message variable

In the Orchestration View window, right-click the message variable you want to remove and then click **Delete**.

In This Section

- How to Consume Web Service Arrays
- Using Multi-part Message Types
- Constructing Messages
- Using Distinguished Fields and Message Properties
- Using Filters to Receive Messages
- Correlating Messages with Orchestration Instances

How to Consume Web Service Arrays

BizTalk Server 2006 provides the ability to consume arrays that are exposed in web services from a BizTalk Orchestration.

To configure an Orchestration to consume an array exposed in a web service:

1. Determine the URL for the web service that exposes arrays. This is typically an asmx web page that lists the operations supported by the web service. For example: http://localhost/ArrayWS/ArraySvc.asmx.

2. Add a web reference to this URL in the Visual Studio project that contains your orchestration:

   - In the Solution Explorer, right-click **References** and select the option to **Add Web Reference…**

   - Enter the URL for the web service into the **URL:** text box and then click **Go**.

   - Enter a name for the web reference into the **Web reference name:** text box and click the **Add Reference** button.

   - The web reference will appear under **Web References** in the Solution Explorer.

3. Add a web port to your orchestration:

   - Drag a **Port** shape from the toolbox to one of the port surfaces in the Orchestration Designer to launch the **Port Configuration Wizard**. Click the **Next** button in the **Port Configuration Wizard** to display the **Port Properties** dialog.

   - Enter a value into the **Name:** text box to identify the port, and click the **Next** button to display the **Select a Port Type** dialog.

   - Select the option to **Use an existing Port Type**, select the Web Port type that corresponds to the web reference you added, and click the **Next** button to display the **Port Binding** dialog.

   - In the Port Binding dialog select the appropriate **Port binding:** option and click the **Next** button, then click the **Finish** button. You should now have a web port displayed in the Orchestration Designer that includes the operations supported by the web service.

4. Add **Send** and **Receive** shapes to your Orchestration as appropriate:

- Drag a **Send** shape from the toolbox to a connecting line in the Orchestration Designer surface to configure the orchestration to send a request message to the web port. If you connect the **Send** shape to one of the web port request message connectors, BizTalk will automatically create a message of the appropriate type to be used when sending a request message to this port.

- Drag a **Receive** shape from the toolbox to a connecting line in the Orchestration Designer surface to configure the orchestration to receive a response message from the web port. If you connect the **Receive** shape to one of the web port response message connectors, BizTalk will automatically create a message of the appropriate type to be used when receiving a response message from this port.

The BizTalk Server 2006 Orchestration Engine provides support for consuming both one dimensional and jagged arrays that are exposed by web services. If you add a web reference to a web service that exposes arrays, the Orchestration Designer will generate a web message type that describes the array. You can then send and receive messages of this type like any other message. BizTalk Server does not limit the sending of web messages containing arrays to only web ports.

How to Use Multi-part Message Types

Each message has a multi-part message type, a description of the message structure that consists of zero or more message parts. The parts are defined by XML Schema Definition (XSD) language schemas or .NET classes. You can define your own multi-part message types, or you can use existing .NET classes and schemas.

You can access or assign message parts directly within your orchestration, or you can use individual elements of message parts that are exposed as distinguished fields or property fields. For more information, see Using Distinguished Fields and Message Properties.

To add a multi-part message type

1. In the **Orchestration View** window, expand the **Types** node.

2. Right-click **Multi-part Message Types** and then click **New Multi-part Message Type**.

   The **Multi-part Message Types** folder expands, if collapsed, and a new multi-part message type is added with one default message part.

3. Name the multi-part message type and the provided message part.

   If your multi-part message type requires more than one message part, you can add additional parts by assigning a name to the <New> message part.

4. Associate each message part with a type, such as a .NET class or schema.

To remove a multi-part message type

- In the **Orchestration View** window, right-click the multi-part message type you want to remove and then click **Delete**.

To remove a part from a multi-part message type

- In the **Orchestration View** window, right-click the part you want to remove and click **Delete**.

To set the type modifier for a multi-part message type

- In the **Properties** window, set the following property:

Property	Description
Type Modifier	Determines the scope of the multi-part message type: • **Private—**Access to this multi-part message type is limited to the containing module. • **Public—**Access to this multi-part message type is not limited. • **Internal—**Access to this multi-part message type is limited to modules within the same project.

To add parts to an existing multi-part message

- From your project, add a reference to Microsoft.XLANGs.BaseTypes.
- Create a variable (for example *xlangPart*) of type **Microsoft.XLANGs.BaseTypes.XLANGMessage**
- Call *xlangPart***.AddPart(...)** using the appropriate arguments from an Expression shape.

If a multi part message that contains greater than the number of declared parts is received, the orchestration engine reads how many parts there are in the message, then constructs the proper part types for the parts that match the number of parts in the declared message type and then constructs **XmlDocument** parts for the remaining parts.

Constructing Messages

In This Section

- Constructing Messages in User Code
- Object References in Message Assignments
- Appending to Messages in User Code
- Using the Construct Message Shape
- Using the Message Assignment Shape
- Using the Transform Shape
- Assigning to Transforms in the Expression Editor
- Using XPaths in Message Assignment

Constructing Messages in User Code

You can represent BizTalk messages at design time as either XSD schemas or .NET classes.

Messages represented as XSD schemas

A template XML instance of the XSD message type is defined at design time and then stored on disk. At run time, a .NET component picks up the XML from disk and returns it as an XmlDocument. The orchestration code can assign this XmlDocument result to the message instance declared in the orchestration.

The **Message Assignment** shape has a single line of code:

The Helper Component that creates the XmlDocument has a single static method:

Messages represented as .NET classes

This approach first involves creating a .NET class that defines your message type. A simple example of such a class is shown here.

Once the message type is defined, it is very easy to write code in the orchestration that will create a new message of this type. Within a **Construct Message** shape, you write simple expressions to create a new message of the **MsgClass** type shown above, and then assign values to the fields which are attributed as Distinguished Fields (if you wish to override the default values). Note that MyMsg is an orchestration message variable whose type is NetClass.MsgClass.

Object References in Message Assignments

When you first assign a .NET-based object to a message or message part, that message holds and maintains a reference to the object.

For efficiency and scalability, the orchestration engine does not do a "deep copy" of the object: that is, it does not copy the entire contents of the object to the message.

If you subsequently assign the object to another message or message part, any modifications to the original results in modifications to the second message or message part. You should avoid this practice, because results are unpredictable.

If you need your second message to have a distinct copy of the object, you should assign the first message or message part to the second message or message part.

Appending to Messages in User Code

Because of the way BizTalk Server handles messages, you cannot simply append a new node directly to an existing message. Instead, you must clone the existing message, as follows:

How to Use the Construct Message Shape

Construct Message shape

You specify the message variable that you want to construct, and make assignments to the message or its parts. All assignments to any given message must take place within the same Construct Message shape.

If you want to modify a property on a message that has already been constructed—such as a message that has been received—you must construct a new message instance by assigning the first to the second and then modifying the property on the new message instance; both the construction and modification occur within the same Construct Message shape.

To configure a Construct Message shape

1. In the Properties window, use the **Messages Constructed** property to specify which messages this shape will construct.
2. Add and configure one or more **Transform** or **Message Assignment** shapes inside the **Construct Message shape**.

How to Use the Message Assignment Shape

Message Assignment shape

To configure a Message Assignment shape

1. If BizTalk Expression Editor is not visible, right-click the **Message Assignment** shape and click **Edit Expression** or, in the Properties window, click the **Ellipsis** (**...**) button for the **Expression** property.

2. In BizTalk Expression Editor, create an expression to assign values. For more information, see BizTalk Expression Editor.

How to Use the Transform Shape

Transform shape

Transforms are only used in the construction of messages, so your **Transform** shape always appears inside a **Construct Message** shape. You can drop the **Construct Message** shape on the design surface and then drop the **Transform** shape inside it, or you can simply drop the **Transform** shape on the design surface, and Orchestration Designer will create the enclosing **Construct Message** shape for you. To configure a Transform shape

1. In the Properties Window, click the **Ellipsis** (**...**) button for the **Input Messages**, **Output Messages**, or **Map Name** property.

2. Use the **Transform Configuration** dialog box to configure the **Transform** shape. For more information, see Transform Configuration Dialog Box.

How to Use the Transform Configuration Dialog Box

You use the **Transform Configuration** dialog box to configure the **Transform** shape. Specifically, you use it to:

- Assign the map to the **Transform** shape. You can create a new map or use an existing one.

- Define source and destination messages to be used in the transform.

To open the Transform Configuration dialog box

1. Right-click a **Transform** shape (within a **Construct Message** shape) and click **Configure Transform** shape.

You enter data into the **Transform Configuration** dialog box using the following elements.

New/Existing Map File?

In this section, you can click either the **New Map** or the **Existing Map** option button to select a map to assign to the **Transform** shape.

Use the **Name** field below the selected option button to specify a map. If you selected **New Map**, you can type a designation for the map you want to assign. When you use the **New Map** option, you must specify the fully qualified name of the map in the text box. The text box displays an example of such a name by default, because it is pre-populated with a unique identifier name based on the project namespace and **Transform** shape name: <Project namespace>.<Transform shape name>_Map (for example, MyProject.Transform3_Map).

If you selected **Existing Map**, click the Down arrow in the **Name** field to select which map file to use. This list box displays an alphabetically sorted list of all the existing maps available in the project. In this list, if you click the text <Select from referenced assembly>, the **Select Artifact Type** dialog box is displayed. For more information about the selections it makes available, see Select Artifact Type Dialog Box.

Select Source and Destination Messages

Use this part of the **Transform Configuration** dialog box to configure the map you selected in the **New/Existing Map File?** section. If you selected **New Map** in that section, you create that map by configuring it in this section.

If you selected **Existing Map**, you can use this section to do one of two things:

- Select an existing map to reuse as-is in the current transform.
- Select an existing map in order to change (reconfigure) it, and then use it in its new configuration in the current transform.

Specify source and destination messages by using the **Source Messages** and **Destination Messages** grid controls. You can use these grid controls to change the map file in several ways. If you delete a message (a row in either grid control), add a message, or select a message of a different type, you alter the structure of the map. When you alter the structure of a map, all other transforms that use it must be changed to match the new structure of the map. Other changes, such as removing a message and inserting in its place a message of the same type, do not alter the structure of the map.

The **Source Messages** and **Destination Messages** grid controls are identical in appearance and behavior. Each grid control has two columns: Message and Type. You populate the grid controls by selecting messages in the Message column. (You add data only into the Message column, because the Type column is read-only.) The cells in the Message column have drop-down lists populated with message instances that are within scope for the current orchestration.

You can select a row in either grid control by clicking the *right arrow* (>) button at the left side of the grid control. After you have selected a row, you can delete it by pressing the DELETE key. Deleting a row (a message) alters the structure of the map file that contained it. You can modify only map files that are local to the project.

When I click OK, launch the BizTalk Mapper

Clicking **When I click OK, launch the BizTalk Mapper** opens BizTalk Mapper automatically when you click **OK** to close the **Transform Configuration** dialog box and save your changes. You cannot save changes, however, if required information is missing. In this case, finish filling out the fields in the dialog box and then click **OK**.

Assigning to Transforms in the Expression Editor

It is possible to dynamically assign a map to your transform at run time. To do this, you can put a **Message Assignment** shape in your orchestration, and use BizTalk Expression Editor to enter an expression that assigns the map you want.

The transform construct takes a fully-qualified map name or a **System.Type** variable. You can use either to select a map at run time.

For example, you could enter code similar to the code below into a message assignment expression to set the map for a transform:

Using XPaths in Message Assignment

You can use the **xpath** function to assign an XPath value to a message part, or to assign a value to an XPath that refers to a message part. For more information on assigning to messages and message parts, see Constructing Messages.

Using Distinguished Fields and Message Properties

Distinguished fields are message data of special interest that you use primarily to make decisions or to manipulate data in your orchestration.

Message properties are either data—contents of the message itself—or "metadata"—context information about the message such as time stamps or routing information. You can use system-defined message context properties or transport context properties, or you can define your own properties by making reference to schema fields from within a property schema. Properties are used in subscriptions and correlations.

- You can designate a field in a schema as a distinguished field or property field by using the **Promote Properties** dialog box from within the Editor. For more information, see Promoting Properties

- You can designate a field in a .NET type as a distinguished field by decorating it with the DistinguishedField attribute, or as a property by the Property attribute.

Using Distinguished Fields

Distinguished fields are referred to by the path to the field in the message, using periods to separate the message name, the names of any records that enclose the field, and the name of the field itself:

Using Message Properties

Once you have added a field to a property schema, its value can be accessed in the orchestration with code and in filter expressions. For more information about property schemas, see Property Schemas.

Message properties are referred to by the name of the message followed by the namespace (the schema) and property name in parentheses:

Property Sets

You can also assign all of the context properties of one message (a property set) to the context properties of another message. To assign a property set, you simply place an asterisk in parentheses after both message names, in the same way you would put a property in parentheses:

Using Filters to Receive Messages

A filter is an expression that you apply to an incoming message to determine whether the message meets your criteria (if any) for being processed. If it does, the message is accepted by the receive action and the orchestration is activated. You apply a filter to a receive action that activates an orchestration.

To create a filter expression, you compare a property of an incoming message on the left side of the expression with a constant on the right side of the expression. You can also create compound expressions by applying the AND and OR operators to two or more expressions. You can also leave blank the filter expression, in which case all messages will be accepted.

A filter expression might look like the following:

In this example, an incoming message is presented to the orchestration. The orchestration has an activation **Receive** shape (the **Activation** property is set to **True** so that receipt of a certain message will cause the orchestration to be run) with the preceding filter expression applied to it. The incoming message is expected to have property called Quantity in the namespace **InvoiceSchema**. The orchestration accepts only invoices for 1000 or more items, so the runtime engine checks the incoming message before it runs.

The following table shows operators that you can use in filter expressions.

Operator	**Description**	**Example**
==	equal to	ReqMsg(Total) == 100
!=	not equal to	ReqMsg(Total) != 100
<	less than	ReqMsg(Total) < 100
>	greater than	ReqMsg(Total) > 100
<=	less than or equal to	ReqMsg(Total) <= 100
>=	greater than or equal to	ReqMsg(Total) >= 100
exists	exists	ReqMsg(Description) exists

Correlating Messages with Orchestration Instances

Correlation is the process of matching an incoming message with the appropriate instance of an orchestration. You can create any number of instances of a given orchestration, and while each of them will perform the same actions, they will do so on different data.

If, for example, your orchestration is designed to issue a purchase order, receive an invoice, and send payment, you need to be sure that the invoice message is received by the orchestration instance that the corresponding purchase order was sent from. Imagine ordering a single item and getting back an invoice for 1000 items, or vice versa, and you will understand the importance of correlation. Likewise, if two related messages are meant to be kept together and sent one after the other, you use correlation to make sure that they are received in the same instance.

You must use correlation whenever your orchestration does not have an explicit way of associating a message with an instance, such as an activate receive, a request-response, or a self-correlating port.

Correlation Sets

You can achieve this sort of correlation of messages with orchestration instances by defining correlation sets. A correlation set is a set of properties *with specific values*. This is different from a correlation type, which is simply a list of properties. If an incoming message does not have all of these properties, with matching values for each, correlation will fail and the message will not be received by the orchestration instance.

Correlation types define a set of properties on which you will be correlating messages. These can be any properties which were previously defined in a property schema and deployed with some BizTalk Project including "system" properties deployed with the GlobalPropertySchemas which is installed as part of the base BizTalk install. A correlation set defines a set of properties and values for these properties that a message must contain to be processed by a particular orchestration.

For example a correlation type could consist of the following properties:

Correlation Type Property	**Possible XML Representation**
Social Security number	<SSN></SSN>
Date of Birth	<DOB></DOB>
Gender	<Gender></Gender>

While a correlation set derived from this correlation type could consist of the following properties and values:

Correlation Set Property/Value	**Possible XML representation**
Social Security number = 222112222	<SSN>222112222</SSN>
Date of Birth = “1/1/1995”	<DOB>”1/1/1995”</DOB>
Gender = Male	<Gender>M</Gender>

Initializing Correlation Sets

- **Correlation Sets initialized on a Receive action**

  Correlation sets initialized on a Receive action define the exact set of properties that must exist in a published message in order for it to be processed by the corresponding receive action(s) in an orchestration. An initializing correlation set will create a correlation set from a correlation type based on the corresponding values in a document.

- **Correlation Sets initialized on a Send action**

  Correlation sets initialized on a Send action are created from a correlation type based upon the corresponding values in a document and promote the correlation properties in the outbound document.

Following Correlation Sets

Following correlation sets can only be bound to a non-activating receive action or to a send action. Following correlation sets are specified in tandem with previously initialized correlation sets.

- **Following Correlation Sets bound to a Receive action**

  Following correlation sets bound to a receive action define the set of properties and values that the document must contain to be received. Receive actions with following correlation sets accept documents that contain properties from a previously initialized correlation set.

- **Following Correlation Sets bound to a Send action**

  Following correlation sets bound to a send action dictate that the set of properties in the correlation set are promoted in the outbound document.

Inspecting correlation sets

BizTalk Server 2006 provides the ability to inspect correlation sets. You can inspect a correlation set in an Expression Shape using code similar to the following:

The example above assumes that you have created a variable named **MsgLen** of type **System.Int16** and that your orchestration contains a correlation set named **Correlation_1**. The ability to inspect correlation sets may be useful if you need to inspect the value of a correlation that was passed to an orchestration by another orchestration.

Properties and correlation types

Each correlation set is based on a **correlation type**, which is simply a list of properties. These properties might be data properties, which are found in the message itself, or context properties, which describe details of the system or messages that are unrelated to the data being conveyed in the message.

You can use a correlation type in more than one correlation set. If you need to correlate on different values for the properties in a correlation type, you must create a new correlation set—each correlation set can be initialized only once.

You can promote properties in a property schema to declare that certain properties in a message are accessible to your orchestration. For more information, see Promoting Properties.

Convoys

Convoys are groups of related messages that are intended by a partner business process to be handled in a specific order within an orchestration. For example, an orchestration might need to receive five different messages before any processing can begin. For more information, see Working with Convoy Scenarios.

Validating message correlation on Send actions

You can validate the properties of a message that you send from your orchestration to ensure that it reflects the properties in its correlation set. By default, this validation is disabled. For information on how to enable it, see Runtime Validation.

Passing correlation sets as parameters to orchestrations

You can pass correlations as *in* parameters to other orchestrations.

In This Section

- Adding, Configuring, and Removing Correlation Types
- Adding and Removing Correlation Sets
- Configuring Correlation Sets
- Working with Convoy Scenarios

How to Add, Configure, and Remove Correlation Types

You can use the Orchestration View window to add correlation types to or remove them from your orchestration.

To add and configure a correlation type

1. In the Orchestration View window, right-click **Correlation Types** and then click **New Correlation Type**.

   The **Correlation Properties** dialog box comes up. Add properties that you want to include in your correlation type. For more information, see Correlation Properties Dialog Box.

To remove a correlation type

1. In the Orchestration View window, right-click the correlation type you want to remove and then click **Delete**.

How to Use the Correlation Properties Dialog Box

You use the **Correlation Properties** dialog box to add or remove properties from a correlation type. The correlation type lists the properties in a correlation set which are used by Send and Receive activities to make sure that incoming messages are properly correlated with the right instances of an orchestration at run time.

There are two panes in the dialog box, each containing a list of properties. The left pane, "Available Properties", contains a tree view of all of the properties that are defined in your project or referenced by it. The properties that appear by default are system properties, defined by BizTalk Server, but you can also define your own properties in a property schema. For more information about creating property schemas, see Property Schemas.

The right pane, "Properties to correlate on", contains a list of properties that you want to include in your correlation type.

- To add a property to your correlation type, select it in the left pane and click the **Add** button.

  The property appears in the right pane and disappears from the left pane.

- To remove a property from your correlation type, select it in the right pane and click the **Remove** button.

  The property reappears in the left pane and is removed from the right pane.

- When you have added all of the properties you want, click **OK**.

  The dialog disappears and your changes are saved to the correlation type.

How to Add and Remove Correlation Sets

You can use the Orchestration View window to add correlation sets to or remove them from your orchestration. A correlation set is an instance of a correlation type.

To add and configure a correlation set

1. In the Orchestration View window, right-click **Correlation Sets** and then click **New Correlation Set**.

   The **Correlation Properties** dialog box appears. Add properties that you want to include in your correlation set. The correlation set and a corresponding correlation type will be created. For more information, see Correlation Properties Dialog Box.

To remove a correlation set

1. In the Orchestration View window, right-click the correlation set you want to remove and then click **Delete**.

How to Configure Correlation Sets

To configure your correlation set, you can select an existing correlation type, create a new correlation type, or modify an existing correlation type.

To assign a correlation type to a correlation set

1. Select an existing correlation type.

   - Or -

   Right-click the new correlation type and select **Configure Correlation Properties**.

   - Or -

   Click the Ellipsis (**...**) button on **Correlation** Properties in the Properties window.

   The **Correlation Properties** dialog box comes up. Add properties that you want to include in your correlation set. For more information, see Correlation Properties Dialog Box. If a correlation type was not already assigned to your correlation set, a new correlation type will be created.

If a correlation type already exists for your correlation set, and you add or remove properties with the **Correlation Properties** dialog box, the existing correlation type will be modified.

To associate send and receive activities with a correlation set

1. Expand the **Correlation Set** node.
2. Select a send or receive activity from the drop-down.

   - Or -

   Select the correlation set in either the **Initializing Correlation Sets** property or the **Following Correlation Sets** property in the Properties window for each **Send** or **Receive** shape you want to associate with the correlation set. For more information, see Using the Send Shape and Using the Receive Shape.

To disassociate send and receive activities from a correlation set

1. In the Orchestration View window, expand the correlation set node if necessary, right-click the activity you want to remove, and then click **Delete**.

   - Or -

Uncheck the correlation set in either the **Initializing Correlation Sets** property or the **Following Correlation Sets** property in the Properties window for each **Send** or **Receive** shape you want to disassociate from the correlation set. For more information, see Using the Send Shape and Using the Receive Shape.

Working with Convoy Scenarios

When a group of correlated messages could potentially be received at the same time, a race condition could occur in which a correlation set in a particular orchestration instance must be initialized by one of the messages before the other messages can be correlated to that orchestration instance. To ensure that all of the correlated messages will be received by the same orchestration instance, BizTalk detects the potential for such a race condition and treats these messages as a **convoy**.

There are two main types of convoys: concurrent correlated receives and sequential correlated receives.

A **convoy set** is a group of correlation sets that are used in a convoy.

Note the following general restrictions on convoys:

- A convoy set can contain no more than three properties.
- Concurrent and sequential convoys can coexist in an orchestration, but they cannot use any shared correlation sets.
- Correlation sets passed into started orchestrations do not receive convoy processing in the started orchestration.
- You cannot initialize two or more correlation sets with one receive to be used in separate convoys. For example, if receive r1 initializes correlation sets c1 and c2, receive r2 follows c1, and receive r3 follows c2, then the orchestration engine will not treat these as convoys. It is a valid convoy scenario if both r2 and r3 follow both c1 and c2, both follow c1 only, or both follow c2 only.

For more information about correlation, see Correlating Messages with Orchestration Instances

In This Section

- Concurrent Correlated Receives
- Sequential Correlated Receives

Concurrent Correlated Receives

Concurrent correlated receives are correlated receive statements in two or more branches of a **Parallel Task** shape.

If a correlation is initialized in more than one parallel task, *each correlated receive must initialize exactly the same set of correlations*. The first such task that receives a correlated message will do the actual initialization, and validation will be done on the others.

Sequential Correlated Receives

Sequential correlated receives are receives that are correlated to previous receives.

Convoy processing takes place for cases in which the correlation sets for a receive are initialized by another receive.

For receives that require convoy processing, the following restrictions apply:

- The correlation sets that constitute a sequential convoy set for a particular receive must be initialized by one preceding receive.
- The port for a receive that requires sequential convoy processing must be the same as the port for the receive initializing the convoy set. Cross-port convoys are not supported.
- Message types for a particular receive that require convoy processing must match the message type for the receive initializing the convoy set, unless the receive statement is operating on an Ordered Delivery port.
- All receives participating in a sequential convoy must follow all the correlation sets that are initialized (or followed) by the initializing receive, unless operating on an ordered delivery port.
- If a sequential convoy is initialized by an activate receive statement, then the activate receive cannot have a filter expression unless operating on an Ordered Delivery port.
- If a sequential convoy is initialized by an activate receive, the following receives cannot be inside a nested orchestration. Otherwise the preceding rules apply for nested orchestrations.

Using Expressions and Variables

Orchestration Designer provides facilities for managing data in your orchestration. You can define data types in the Orchestration View window, create messages, variables, and parameters in the Orchestration View window, and use BizTalk Expression Editor to add logic to manipulate and test values.

In This Section

- Expressions
- Orchestration Variables
- Adding Parameters to Orchestrations

Expressions

You can create expressions to make decisions, set delays, make .NET calls, and test while loop conditions. You can use operators to build rich expressions. You can assign values to and from messages and their distinguished fields, assign values to dynamic ports, call .NET classes, and pass messages as parameters to .NET calls.

To open BizTalk Expression Editor

1. Right-click an **Expression** shape, a **Loop** shape, a **Message Assignment** shape, a **Delay** shape, or a branch of a **Decide** shape.
2. Click **Edit Expression**.

   BizTalk Expression Editor appears.

To create or edit an expression

1. Type or edit the expression in the text field of BizTalk Expression Editor.
2. Click OK. This saves the expression and applies it to the selected shape.

In This Section

- Using Expressions to Refer to Message Components
- Assigning to Dynamic Ports
- Using Expressions to Create Objects and Call Object Methods
- Using the Expression Shape
- Using BizTalk Expression Editor
- Shapes that Take Expressions

Using Expressions to Refer to Message Components

You can use expressions to manipulate messages in various ways in your orchestration.

Referring to message fields

You can refer to a distinguished field in a message by appending the field name to the message name as follows:

In this example, MyMsg is the message, and Amount is a field that has been identified as a distinguished field for the message type that MyMsg is based on.

Assigning to messages and message parts

You can assign a message directly to another message, or a message part to a message part:

In this example, Invoice is a message part based on a schema.

If you want to modify a property on a message that has already been constructed—such as a message that has been received—you must construct a new message by assigning the first to the second in a Construct Message shape, and modify the property on the new message within the same Construct Message shape.

Adding message parts

You can add additional parts to an existing multi part message through the use of the XLANGs.BaseTypes.XLANGMessage.AddPart method. This may be useful when sending SMTP e-mails with an indeterminate number of attachments.

This functionality is implemented as follows:

- From your project, add a reference to Microsoft.XLANGs.BaseTypes.
- Create a variable (for example *xlangPart*) of type **Microsoft.XLANGs.BaseTypes.XLANGMessage**
- Call *xlangPart*.**AddPart(...)** using the appropriate arguments from an Expression shape.

Message part context property access

A message part has context separate from the message context. You can construct message part context properties through the schema editor when you set the **Property Schema Base** property for the associated element to **PartContextPropertyBase.**

The access is similar to message properties, through an expression like:

For example:

Assigning to message properties

You can assign a value to a message property:

In BizTalk Server 2006 you can refer to and assign values to the MIME properties of a multi-part message:

Assigning to Dynamic Ports

If a send port is marked as dynamic, you can assign to it the value of some variable of type string—perhaps a message property—that contains the URI of the port you want to use. In this example, you would first assign the URI to the message property:

Using Expressions to Create Objects and Call Object Methods

You might need to use expressions to create objects or invoke methods.

Creating objects

To create a variable that has a type which is a .NET class, you construct an object in the **Expression** shape. The properties of your .NET class variable include a constructor. If you use the default constructor, you simply declare the variable directly as you would any other variable, like one of type bool or int.

If you use a constructor that takes parameters, you use the keyword **new**, followed by the object class and any parameters in parentheses:

Invoking methods

To invoke a method on a .NET class object, you append a period and the name of the method to the object reference, followed by any parameters in parentheses:

Passing and using messages as parameters

To pass a message as a parameter to a method call on a .NET class, you first add a reference to Microsoft.XLANGs.BaseTypes.dll in the project that defines the class, and then use the type XLANGMessage in the method signature.

Referencing the multi-part message type enables you to access the various parts of the message by using the type XLANGPart:

In the call itself, you simply supply the name of the message as you would any other parameter:

You can also pass a message part as type XLANGPart.

NET member invocation

BizTalk Server 2006 removes many of the BizTalk 2004 restrictions for accessing .NET user code from an Orchestration. BizTalk 2004 limits .NET user code access to static fields that are members of simple or distinguished types such as bool, int or string. With BizTalk Server 2006 you can access public members except in the case of direct access to members of a message part. To directly access a member of a message part it must be promoted as a distinguished field.

How to Use the Expression Shape

The **Expression** shape enables you to enter any expression you choose in your orchestration. For example, you can make a .NET call to run an external program, or simply manipulate the values of your orchestration variables.

While the **Expression** shape is quite flexible, it is not good practice to use it to perform high-level orchestration logic, which preferably would be visible in the orchestration drawing itself. In general, it is easier to understand and maintain your orchestrations if your **Expression** shapes contain simple and modular expressions.

To configure an Expression shape

1. If BizTalk Expression Editor is not visible, right-click the **Expression** shape and click **Edit Expression** or, in the Properties window, click the Ellipsis (**...**) button for the **Expression** property.

2. In BizTalk Expression Editor, create an expression to assign values. For more information, see BizTalk Expression Editor.

Using BizTalk Expression Editor

BizTalk Expression Editor enables you to create expressions to expand the capabilities of various Orchestration Designer shapes. It enables you to assign values to messages or message parts in the Message Assignment shape, and to make .NET calls and manipulate the values of variables in the **Expression** shape. You can also use it to construct complex Boolean expressions in the **Loop** and **Decide** shapes, and to set a delay in the **Delay** shape.

IntelliSense

The IntelliSense feature will help guide you in creating expressions—for example, it can display a list of class members for you when you type in a .NET class name followed by a period (.). For more information on Intellisense, and shortcut keys for using it, see "Using IntelliSense" in the Visual Studio .NET Combined Collection.

Shapes that Take Expressions

Several shapes in Orchestration Designer, including **Decide** and **Loop**, use Boolean expressions to form rules that control branching. Other shapes use expressions for other purposes. You can create or edit an expression for these shapes by using BizTalk Expression Editor.

The following table summarizes the shapes that use expressions in Orchestration Designer and lists the data types that are valid for those expressions.

Shape	**Description of expression use**	**Valid expression data types**
Decide	**Decide** shapes contain **Rule** shapes, which use Boolean expressions.	Boolean
Receive	**Receive** shapes that have the Activate property set to True use the **Filter Expression** property to filter incoming messages. The expression in this property must evaluate to a Boolean, whose value determines whether or not to accept an incoming message. The **Filter Expression** dialog box is used to create filter expressions.	Boolean
Loop	A **Loop** shape requires a **Rule** shape, which in turn must contain a Boolean expression.	Boolean
Rule	**Rule** shapes (displayed on the Process Area as "branch" shapes) are simple shapes that contain Boolean expressions and are used within other (complex) shapes to govern branching.	Boolean
Listen	Each branch of a **Listen** shape contains, at a minimum, either a **Receive** shape, which uses a Boolean expression only for filter expressions (see the entry for **Receive**), or a **Delay** shape, which uses a **System.DateTime** object or **System.TimeSpan** object.	Boolean, System.DateTime, System.TimeSpan
Delay	The expression used in a **Delay** shape must evaluate to a **System.DateTime** object, to express a deadline, or a **System.TimeSpan**	System.DateTime, System.TimeSpan

	object, to express duration.	
Message Assignment	The expression in a **Message Assignment** shape assigns a value to a message. The value assigned can be of any type, though typically a message is assigned.	Any

Orchestration Variables

The Orchestration View window enables you to manage an orchestration's properties (also known as **Service** properties), parameters, ports, messages, and other variables. In addition to ports and messages, you can create integer variables, Boolean variables, string variables, or variables of a .NET class.

You can also use the Orchestration View window to manage the variables that belong to your scopes.

Initializing variables

You can initialize the value of a variable by setting it in the Properties window. If the variable is an instance of a class, you can specify a constructor. Note that if you do not specify a value or a constructor, your variables will be assigned default values, or will be created with a default constructor, as soon as an instance of your orchestration is created.

The only circumstance in which you are required to explicitly initialize your variables is when your orchestration contains more than one activate receive, as is possible in a **Scope**, **Parallel Actions**, or **Listen** shape. In this case, automatic initialization is disabled, and you must use an **Expression** shape to initialize your variables. You must place an **Expression** shape after each activation receive, and before any variable is accessed in the orchestration.

To add a variable

1. In the Orchestration View window, right-click the **Variables** folder and then click **New Variable**.

   The **Variables** folder expands, if collapsed, and a new variable is added.

2. Name the variable by typing a name in the Identifier property in the Properties window.

3. Associate the variable with a type, such as a .NET class.

4. If you select a predefined variable type, you have the option of specifying an initial value for the variable. In the Properties window, set the **Initial Value** property.

Otherwise, if the selected type is a .NET class, you have the option of using a default constructor. In the Properties window, set the following property:

Property	**Description**
Use Default Constructor	If a default constructor is available for a .NET class, this property determines whether the default constructor will be called when you use the variable for the first time: True—The default constructor will be called. This is the default value when a default constructor is available. False—The default constructor will not be called; you must call a constructor in an expression or make an assignment to the variable before you can use it in your orchestration.

To remove a variable

1. In the Orchestration View window, right-click the variable you want to remove and then click **Delete**.

In This Section

- Variable Scope in Orchestrations

Variable Scope in Orchestrations

There are a few important points to understand about the visibility and state of variables and orchestration parameters in various places within your orchestration, including exception handlers and compensation blocks.

- Orchestration parameters are visible throughout the body of the orchestration as well as in its compensation block.

- Scope-level variables are visible in the body of the scope as well as in all of its exception handlers and compensation block. Inside exception handlers, the variables are considered un-initialized since it is indeterminate whether the exception that led to a given handler took place before or after any initialization in the body. For this reason, if you declare a variable in a scope and initialize it in the body, you cannot read from it in an exception handler, or you will get a compiler error. A workaround for that exists in the case of long running transactions. The following example shows how the succeeded() operator can be used to ensure proper runtime behavior:

- Inside a compensation block, all variables assume the values they had when the body of the scope committed. Note that the compensation block of a scope never runs immediately after its body completes; there is always intervening code. If any of that intervening code updates some outer-scope variables, and then the compensation block runs, those updates will not be seen inside the compensation block. Instead the variables will temporarily revert to the values they had at the time the scope which owns that compensation had committed.

- When a transaction is nested inside a loop, the transaction will be compensated as many times as the loop executes. Inside the compensation block for each iteration, outer-scope variables assume the values they had at the time of the iteration of the transaction committed.

- Any updates made to outer-scope variables or orchestration parameters inside compensation blocks of inner scopes will not be visible after the compensation block completes. In other words, the compensation block cannot be used to directly modify the values of outer-scope variables.

BizTalk Expression Editor

BizTalk Expression Editor in Orchestration Designer is a standard Visual Studio text editor, which means it offers IntelliSense. You use BizTalk Expression Editor to enter an expression in textual form.

Use the text box to enter a single expression in textual form. The expression can span multiple lines, but must end with single semicolon.

Though you can use BizTalk Expression Editor to enter complex expressions easily and quickly, you cannot use it to enter an arbitrary amount of code. The reason for this is to keep code for the business process separate from its implementation code.

For more information about using expressions in Orchestration Designer, see Expressions.

Using Transactions and Handling Exceptions

When you design an orchestration, you should consider carefully where problems might occur and how best to deal with them. Many orchestrations have several potential points of failure. Problems can arise for any number of other reasons; for example, a server might go down or a message might be badly formatted.

It is especially important for a long-running or complex orchestration to keep track of its state and to report errors as they occur, so that you can resolve problems accurately and with a minimum of effort. It is equally important for an orchestration to maintain the integrity of a set of closely related actions, so that if part of a transaction takes place but another does not, the entire transaction can be rolled back as though it never occurred.

BizTalk Orchestration enables you to guarantee the atomicity of work, that is, the integrity of related actions, even when external systems are participating in transactions. It gives you tools to handle errors, to maintain the state of an orchestration, and to fix problems as they occur through transactions, compensation, and exception handling.

As a framework for transactions and exception handling, Orchestration Designer provides the **Scope** shape. A scope can have a transaction type, a compensation, and any number of exception handlers.

The steps for setting up a transaction and exception handling are:

- Create a scope.
- Identify the kind of transaction you need.
- Determine what will need to be compensated.
- Identify potential errors.
- Add appropriate exception handlers and compensation code.

In This Section

- Scopes
- Transactions
- Compensation
- Exceptions
- Transacted Orchestrations

Scopes

A scope is a framework for grouping actions. It is primarily used for transactional execution and exception handling.

A scope contains one or more blocks. It has a body and can optionally have appended to it any number of exception-handling blocks. It may have an optional compensation block as well, depending on the nature of the scope. Some scopes will be purely for exception handling, and will not require compensation. Other scopes will be explicitly transactional, and will always have a default compensation handler, along with an optional compensation handler that you create for it. A transactional scope will also have a default exception handler and any number of additional exception handlers that you create for it.

Synchronized scopes

You can specify that scopes are synchronized or not synchronized. By synchronizing a scope, you ensure that any shared data that is accessed within it will not be written to by one or more parallel actions in your orchestration, nor will it be written to while another action is reading it.

Atomic transaction scopes are always synchronized.

All actions within a synchronized scope are considered synchronized, as are all actions in any of its exception handlers. Actions in the compensation handler for a transactional scope are not synchronized.

Nesting of scopes

You can nest **Scope** shapes inside other **Scope** shapes. The rules for nesting scopes are as follows:

- Transactional and/or synchronized scopes cannot be nested inside synchronized scopes, including the exception handlers of synchronized scopes.
- Atomic transaction scopes cannot have any other transactional scopes nested inside them.
- Transactional scopes cannot be nested inside nontransactional scopes or orchestrations.
- You can nest scopes up to 44 levels deep.
- **Call Orchestration** shapes can be included inside scopes, but the called orchestrations are treated the same as any other nested transaction, and the same rules apply.
- **Start Orchestration** shapes can be included inside scopes. Nesting limitations do not apply to started orchestrations.

Scope variables

You can declare variables such as messages and correlation sets at the scope level. You cannot use the same name for a scope variable as for an orchestration variable, however; name hiding is not allowed.

In This Section

- Using the Scope Shape

How to Use the Scope Shape

The **Scope** shape provides a contextual framework for its contents. The first block of a **Scope** shape is the context block, or body, in which the basic actions of the scope take place; it is analogous to the try block in a try/catch statement. Following the body, the **Scope** shape may also include one or more exception-handler blocks and a compensation block.

To configure a Scope shape as a transaction boundary

1. In the Properties window, set the **Transaction Type** property to **Atomic** or **Long Running**.

2. If the **Transaction Type** is set to **Atomic**, then in the Properties window, specify the following properties:

Property	Description
Batch	Boolean value that determines if this transaction can be batched with other transactions across multiple instances of the orchestration. The default value is **True**.
Isolation Level	Determines the degree to which data is accessible among concurrent transactions: • Read Committed—To prevent the selected transaction from accessing data modifications in concurrent transactions until they are committed. This option is the Microsoft SQL Server default setting. • Repeatable Read—To require read locks until the selected transaction is complete. • Serializable—To prevent concurrent transactions from making data modifications until the selected transaction is complete. This option is the most restrictive isolation level.
Retry	Boolean value that determines whether this transaction is retried when an error occurs. The default value is **True**. **Note** An atomic transaction will be retried if you throw Microsoft.XLANG.BaseTypes.RetryTransactionException, or if the orchestration engine is unable to store its state or commit the transaction.
Timeout	Determines the time in seconds until the transaction fails due to inactivity. If you do not want to use a timeout, set the value of this property to 0. **Note** This is a DTC timeout, and is not enforced by the orchestration engine. For atomic transactions only, the engine will not interrupt the transaction. It proceeds normally until commitment, at which point it fails

	to commit only if participating in a DTC transaction through one of the objects inside it.

3. If the **Transaction Type** is set to **Long Running**, then in the Properties window, specify the following property:

Property	Description
Timeout	Determines the time in seconds until the transaction times out and is considered a failed transaction. If you do not want to use a timeout, set the value of this property to 0.

To configure a Scope shape to contain local variables

1. Double-click on the scope in the Orchestration View window.
2. Right-click the Variables folder under the scope and then click **New Variable**.
3. Proceed from step 2 in "To add a variable" in Orchestration Variables.

Transactions

The BizTalk Server Orchestration Engine manages the state, applies business logic, and invokes the supporting applications of complex processes and/or transaction sets. Business processes can be composed as discrete pieces of work using atomic transactions that automatically roll back all changes in case of errors or long running, which can contain nested transactions and use custom exception handling to recover from error scenarios. These transactional semantics are typically managed through the Scope construct in the Orchestration Designer. The long running processes can span days, weeks, and longer time durations. The long running processes typically utilize correlation to correlate messages that are received to the messages that may be sent. The orchestration engine typically dehydrates these instances to conserve system resources and re-hydrates the process upon receipt of these correlated messages. The orchestration engine persists the orchestration state to the MessageBox database at known checkpoints to recover from any application or system exceptions.

The transactional programming model provided for the BizTalk Orchestration engine includes support for exception handling and recovery from failed transactions, atomic transactions that automatically roll back their actions if an error occurs, or long-running transactions that can contain other transactions as well as custom exception handling.⊟In This Section

- Atomic Transactions
- Long-Running Transactions

Atomic Transactions

BizTalk orchestrations can be designed to run discrete pieces of work, following the classic 'ACID' concept of a transaction. These discrete or atomic units of work, when performed, move the business process from one consistent state to a new, consistent and durable state that is isolated from other units of work. This is typically done by using the **Scope** construct that encapsulates the units of work with the transactional semantics. The entire orchestration can also be defined as an atomic transaction without the use of scopes. The scopes, however, cannot be marked as transactional unless the orchestration itself is marked as a long running or atomic transaction type. Atomic transactions guarantee that any partial updates are rolled back automatically in the event of a failure during the transactional update, and that the effects of the transaction are erased (except for the effects of any .NET calls that are made in the transaction). Atomic transactions in BizTalk orchestrations are similar to distributed transaction coordinator (DTC) transactions in that they are generally short-lived and have the four "ACID" attributes (atomicity, consistency, isolation, and durability):

- **Atomicity**

  A transaction represents an atomic unit of work. Either all modifications within a transaction are performed, or none of the modifications are performed.

- **Consistency**

  When committed, a transaction must preserve the integrity of the data within the system. If a transaction performs a data modification on a database that was internally consistent before the transaction started, the database must still be internally consistent when the transaction is committed. Ensuring this property is largely the responsibility of the application developer.

- **Isolation**

  Modifications made by concurrent transactions must be isolated from the modifications made by other concurrent transactions. Isolated transactions that run concurrently perform modifications that preserve internal database consistency exactly as they would if the transactions were run serially.

- **Durability**

  After a transaction is committed, all modifications are permanently in place in the system by default. The modifications persist even if a system failure occurs.

The following isolation levels are supported by the atomic transactions used in BizTalk Orchestrations:

- **Read Committed**

  Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data.

- **Repeatable Read**

  Locks are placed on all data that is used in a query, preventing other users from updating the data. This prevents non-repeatable reads, but phantom rows are still possible.

- **Serializable**

  A range lock is placed preventing other users from updating or inserting rows into the database until the transaction is complete.

BizTalk Server ensures that state changes within an atomic transaction—such as modifications to variables, messages, and objects—are visible outside the scope of the atomic transaction only upon commitment of the transaction. The intermediate state changes are isolated from other parts of an orchestration.

If you require full ACID properties on the data—for example, when the data must be isolated from other transactions—you must use atomic transactions exclusively.

When an atomic transaction fails, all states are reset as if the orchestration instance never entered the scope. The rule BizTalk has for atomic transactions is that all variables (not just local to the scope) participate in the transaction. All non-serializable variables and messages used in an atomic transaction should be declared local to the scope; otherwise, the compiler will give the "variable….is not marked as serializable" error. All atomic scopes are assumed to be "synchronized" and the orchestration compiler will provide a warning for redundant usage, if indeed the synchronized keyword is used for atomic scopes. The synchronization for the shared data extends from the beginning of the scope until the successful completion of the scope (including the state persistence at the end of the scope) or to the completion of the exception handler (in case of errors). The synchronization domain does not extend to the compensation handler.

Atomic transactions can be associated with timeout values at which point the orchestration will stop the transaction and the instance will be suspended. If an atomic transaction contains a Receive shape, a Send shape, or a Start Orchestration shape, the corresponding actions will not take place until the transaction has committed.

An atomic transaction retries when the user deliberately throws a **RetryTransactionException** or if a **PersistenceException** is raised at the time that the atomic transaction tries to commit. The latter could happen if for instance, the atomic transaction was part of a distributed DTC transaction and some other participant in that transaction stopped the transaction. Likewise, if there were database connectivity problems at the time when the transaction was trying to commit, there would also be a **PersistenceException** raised. If this happens for an atomic scope that has **Retry=True**, then it will retry up to 21 times. The delay between each retry is two seconds by default (but that is modifiable). After 21 retries are exhausted, if the transaction is still unable to commit, the whole orchestration instance is suspended. It can be manually resumed and it will start over again, with a fresh counter, from the beginning of the offending atomic scope.

The atomic scopes have restrictions on nesting other transactions that cannot nest other transactions or include **Catch - Exception** blocks.

DTC transactions

While atomic transactions behave like DTC transactions, they are not explicitly DTC transactions by default. You can explicitly make them DTC transactions, provided that any objects being used in the transaction are COM+ objects derived from System.EnterpriseServices.ServicedComponents, and that isolation levels agree between transaction components.

Nonserializable objects

If you want to use a nonserializable object within an orchestration, you must use it only within an atomic transaction. Any use of such an object outside of an atomic transaction risks data loss whenever the orchestration is persisted by the engine.

Scenarios using Atomic Transactions

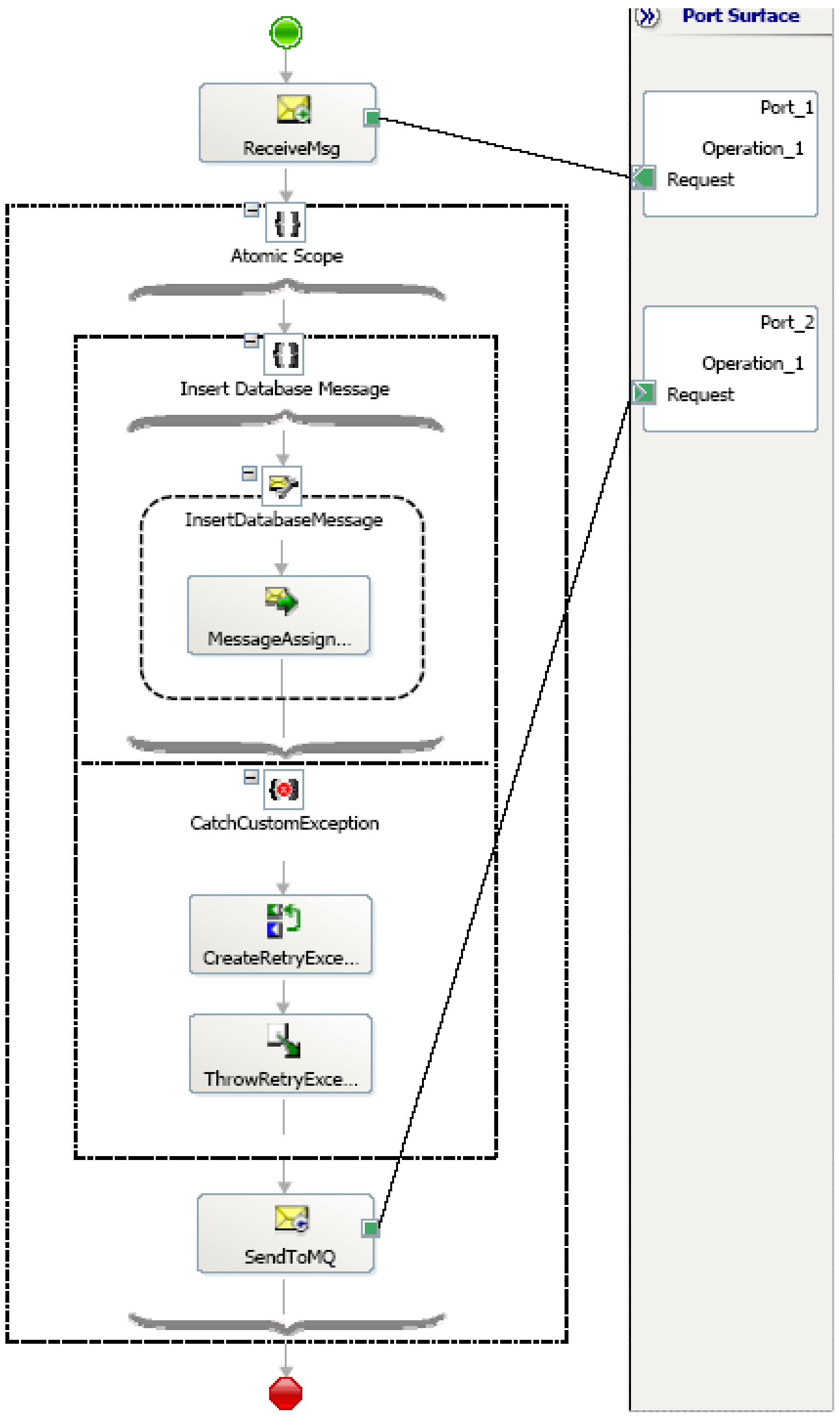
The following scenarios describe the use of atomic transactions.

Scenario 1 - An Atomic Transaction with COM+ ServicedComponent

The following orchestration shows how to use the **RetryTransactionException** with atomic transactions. Although exception handlers cannot be directly included for an atomic scope, the scope can include a non-transactional scope that can have an exception handler. The **ServicedComponent** enlists in the same DTC transaction and any exception raised by the component is caught and re-thrown as **RetryTransactionException**. (This assumes that the **Retry** property is set to **True** for the atomic scope).

Note that the orchestration would have been suspended and the action in the MessageAssignment shape would have been rolled back even if the **RetryTransactionException** is not thrown. This pattern, however, allows building resilience in the application where the retries occur automatically.

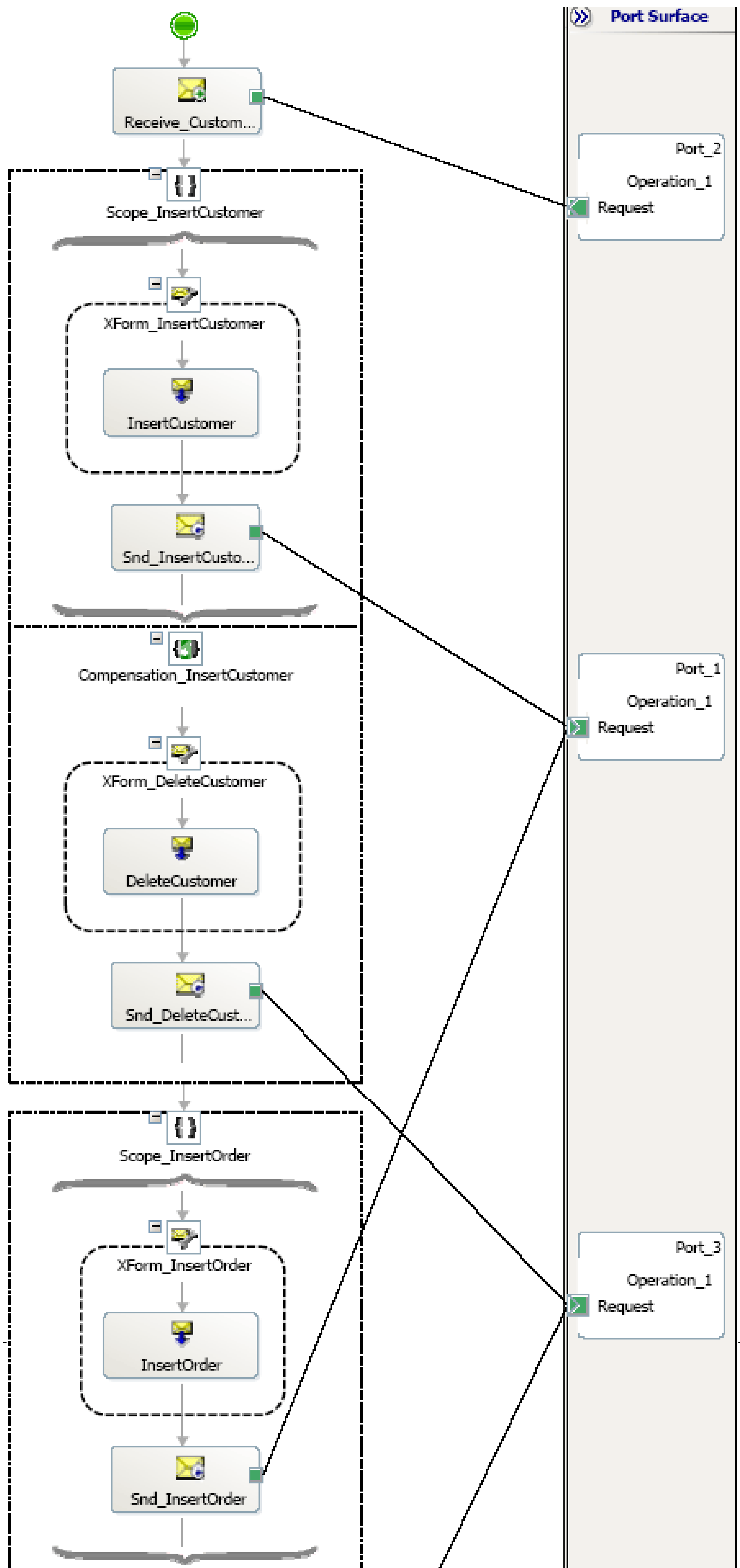
An atomic transaction with COM+ ServicedComponent



Scenario 2 - Using Transacted Adapters with Atomic Transactions

The following orchestration shows how to use the atomic transactions with the SQL adapter. The whole orchestration is marked as long running with individual atomic transactions for the two logical pieces of work, Inserting a new Customer and Insert Order details for the customer. If for whatever reason, the Order Insert fails, the Customer Insert should be rolled back. The sample uses the SQL adapter to do the database work. As mentioned earlier, the scope associated with an atomic transaction completes when the message is sent to the MessageBox database. This implies that after the engine is successful in sending the message in the Scope_InsertCustomer and Scope_InsertOrder scopes, each one of the scopes commits. The SQL adapter creates a new transaction for the actual Insert of the Customer or the order. The Ports have a property “Delivery Notification” for validating that the message has been successfully sent via the Sent Port. When the Delivery Notification property is set to “Transmitted”, a receive subscription is placed before the Transactional commit point of the Send Operation. However in case of Atomic Scopes, the receive subscription is placed in the enclosing Parent scope. In the scenario where the InsertOrder SQL transaction fails, a "Nack" will be sent back and the "Scope_InsertOrder" commits. After the Sent Port exhausts the configured retries, a DeliveryFailureException will be raised This exception will be caught by the default exception handler, which will run the default compensation process. This will invoke the compensation handler associated with the Scope_InsertCustomer, causing the undo operation of inserting the customer information.

Transacted adapters with atomic transactions


Port Surface
Receive_Custom...
Scope_InsertCustomer
XForm_InsertCustomer
InsertCustomer
Snd_InsertCusto...
Compensation_InsertCustomer
XForm_DeleteCustomer
DeleteCustomer
Snd_DeleteCust...
Scope_InsertOrder
XForm_InsertOrder
InsertOrder
Snd_InsertOrder
Port_2
Operation_1
Request
Port_1
Operation_1
Request
Port_3
Operation_1
Request

Long-Running Transactions

Long-running transactions are important, commonly used constructs in BizTalk orchestrations. They provide you with facilities for custom scope-based compensation, custom scope-based exception handling, and the ability to nest transactions, all of which give you great flexibility in designing robust transaction architecture.

You use a long-running transaction when the transaction might need to run for an extended time and you do not need full ACID properties (that is, you do not need to guarantee isolation of data from other transactions). A long-running transaction might have long periods of inactivity, often due to waiting for external messages to arrive.

Long-running transactions possess consistency, and durability, but not atomicity and isolation. The data within a long-running transaction is not locked; other processes or applications can modify it. The isolation property for state updates is not maintained because holding locks for a long duration is impractical.

Commitment of a long-running transaction is different than commitment of an atomic transaction. There is no implicit assumption of distributed coordination regarding the outcome (a long-running transaction exists only within a single orchestration instance). Instead, a long-running transaction is considered committed when the last statement in it has completed. There is no "auto" rollback of state in case of a transaction abort. You can achieve this programmatically through the exception and compensation handlers.

A scope can define its own state by declaring variables, messages, and .NET components. A long-running transaction has access to the state information of its own scope, any scope that encloses it, and any state information that is globally defined within the orchestration. It does not have access to the state information of any scopes that do not enclose it.

Nesting

Long-running transactions can contain atomic transactions or other long-running transactions. They can be nested to arbitrary depths. For example, your transaction might contain two other long-running transactions, each of which might contain atomic transactions.

Nesting is particularly useful when one or more components of the overall transaction need to be atomic while the overall transaction needs to be long-running. Consider the example of receiving and fulfilling a purchase order. The purchase order can arrive at any time, and the various steps in fulfilling the order might take time to occur, but you still want to treat the entire process as a transaction. The overall transaction in this case clearly needs to be long-running, but an individual step, such as acknowledging a payment, might need to be atomic.

Compensation

A long-running transaction can specify a compensation block that will be called to compensate for the transaction's activities, after it commits. It might simply undo the transaction where feasible, or perform some other function, such as notification, that helps mitigate the effects of the transaction in some way. If you do not add your own compensation code, the runtime engine will by default call the compensation blocks of the inner transactions, both long-running and atomic, in reverse order, starting with the last transaction committed and finishing with the first transaction committed.

Fault tolerance

Transactions support fault tolerance for recovery from both internal faults (such as machine failures and software faults) and external faults (such as cancel messages). Partial updates within long-running transactions are not rolled back automatically when a transaction failure occurs, as they are in ACID transactions.

The exception code block for the long-running transaction is called when a fault occurs. The exception code block contains a set of fault handlers that you write to deal with any of the faults that can arise during the execution of the transaction. You can rely on the last known state of the messages, variables, and objects in handling the fault.

You will typically want your exception handler to evaluate the state of the orchestration at the time the exception occurs, take any necessary action based on that state, and call the compensations of any nested transactions.

The execution of the long-running transaction is interrupted when a fault occurs. It cannot be resumed after a fault occurs.

Scenarios using Long Running Transactions

The following scenarios describe the use of long running transactions.

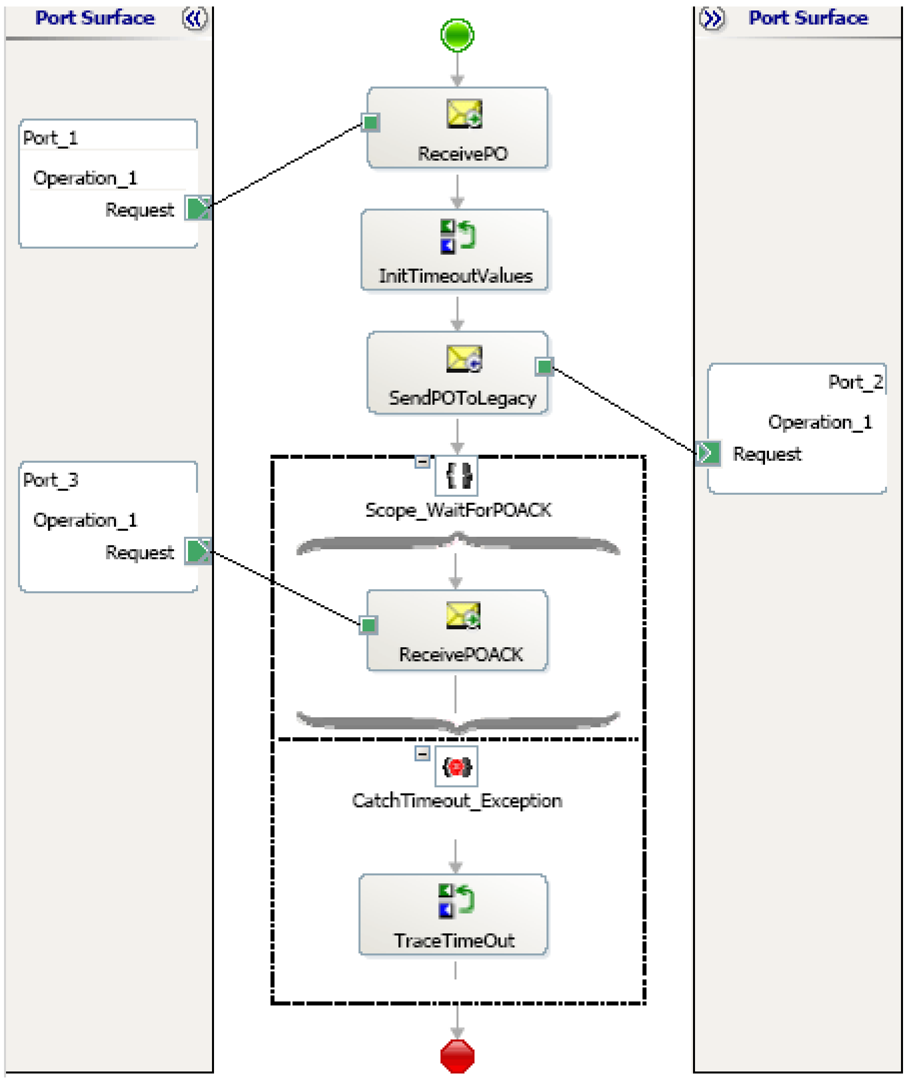
Scenario 1 - Using Long Running Transactions with Timeouts

Long running scopes can be associated with a timeout, which is a logical time within which the long-running work must complete. If the scope does not complete within the specified time, a pre-defined system exception **TimeoutException** is raised.

You can create long running processes by either marking the entire orchestration as long running or having an outer long running scope nest any other scopes. In the former scenario, a system-provided exception handler runs, whereas the latter allows associating specific exception handlers to the outer scope. The default system-provided exception handler will run the compensation handler for each of the successfully completed nested transactional scopes, if any, in reverse order of their completion. You can achieve the same through self compensating by using the Compensate shape in the exception handler for a long running transaction.

The following orchestration is a representation of how to associate timeouts with long running transactions. Sometime, you may need to interface with legacy systems that operate in a batch fashion. The scenario shows a purchase order being received and sent to the legacy system. The legacy system processes the purchase order and sends back a purchase order acknowledgement. The send operation initializes a correlation set using the purchase order number and the receive operation follows that correlation set. The receive operation is also in a long running scope with a timeout value. The orchestration engine will dehydrate the orchestration instance waiting for the receive. The correlation will ensure that the same orchestration instance is invoked after the message is received. If the purchase order acknowledgement does not arrive within the time interval specified by the timeout values, a **TimeoutException** will be thrown.

Long running transactions with timeouts

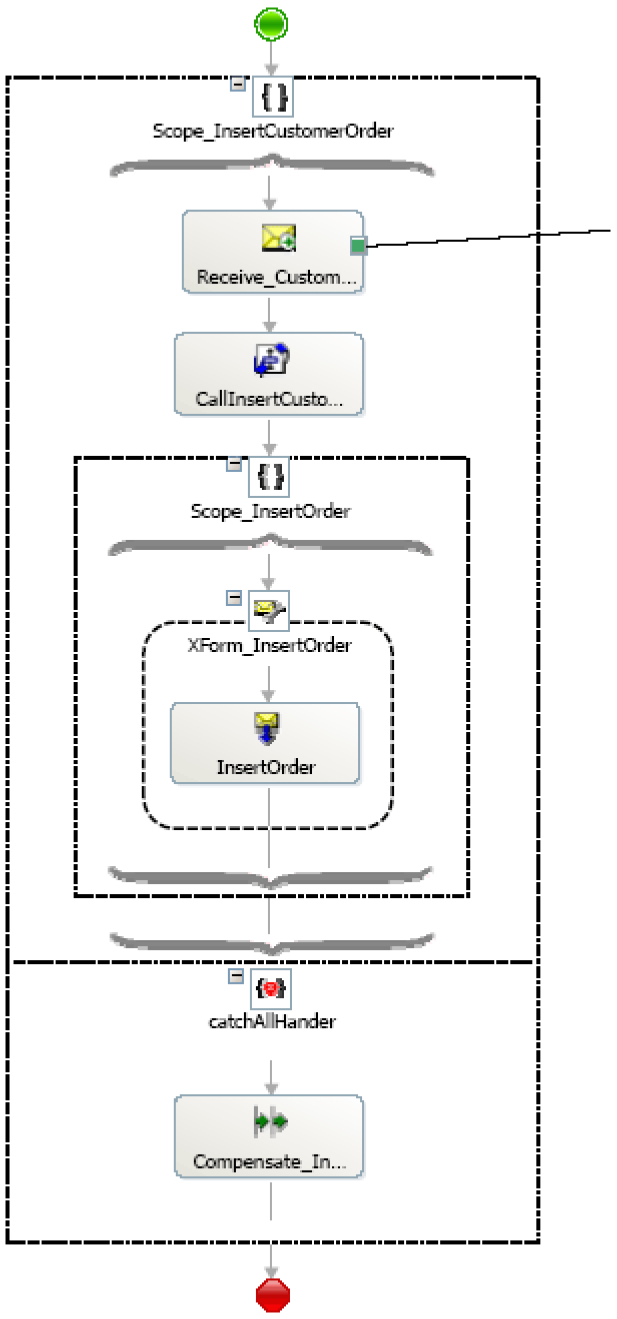


Scenario 2 - Using Long Running Transactions with Custom Compensation

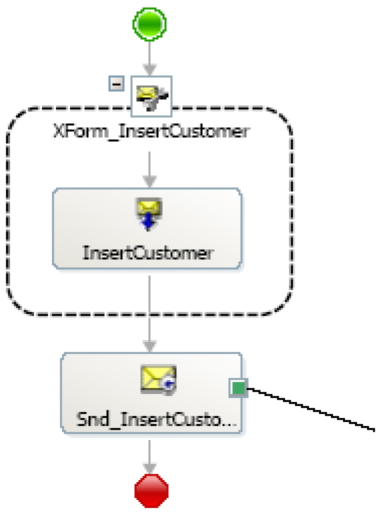
The following two orchestrations demonstrate how to associate and invoke custom compensations associated with entire orchestrations. This scenario inserts a new customer and inserts order details for the customer. The logic of the orchestration dictates that if the order insert fails, you should roll back the customer insert. The customer insert could be done by a legacy system, and is therefore, demonstrated in a separate callable orchestration. The called orchestration has the **Custom** property set for compensation, which provides a separate sheet to do the compensation process. The compensation is to delete the newly inserted customer.

The calling orchestration has a long running scope to do the order insert. This scope is nested within an outer long running scope. The outer scope has an exception handler associated to catch any exceptions. The handler uses the Compensate shape to invoke the custom exception associated with the called orchestration to roll back any changes that might have occurred in the call to the orchestration.

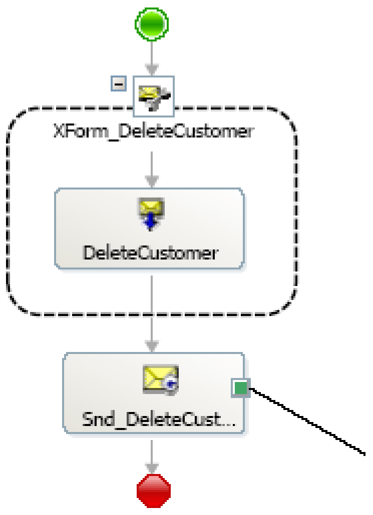
Long running transactions with custom compensation



Called orchestration (main)



Called orchestration (compensation)



Persistence in Orchestrations

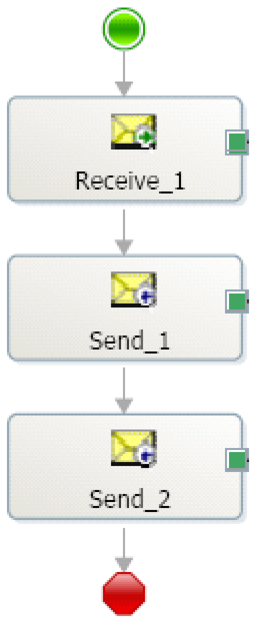
The orchestration engine saves the entire state of an orchestration instance at various persistence points to allow rehydration of the orchestration instance. The state includes any .NET-based components that may be used in the orchestration, in addition to messages and variables. The engine stores state at the following persistence points:

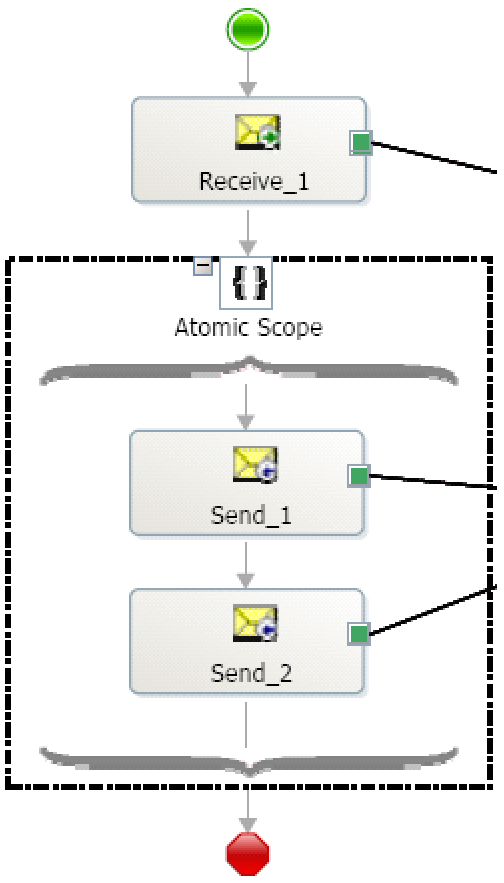
- End of a transactional scope (atomic or long running)
- At debugging breakpoints

- At the execution of other orchestrations through the Start Orchestration shape
- At the Send shape (except in an atomic transaction)
- When an Orchestration Instance is suspended
- When the system shutdowns in a controlled manner
- When the engine determines it wants to dehydrate
- When an orchestration instance is finished

The engine optimizes the number of persistence points as they are expensive, especially when dealing with large message sizes. As shown in the two orchestration instances below, in the orchestration with the Send Shapes in an atomic scope, the engine determines a single persistence point between the end of the transaction scope and the end of the orchestration. In the other orchestration, there will be two persistence points, one for the first Send shape and the second for the Send shape plus end of the orchestration.

Orchestration persistence





Any .NET-based objects you use in orchestrations, either directly or indirectly, must be marked as serializable unless they are invoked in atomic scopes, or if the objects are stateless and are invoked only through static methods. **System.Xml.XmlDocument** is a special case and does not need to be marked as serializable regardless of the transaction property for a scope.

How does the special handling for **System.Xml.XmlDocument** work:

When the user defines a variable **X** of type **T**, where **T** is **System.Xml.XmlDocument** or a class derived from **System.Xml.XmlDocument** then the compiler will treat **X** as a serializable object.

When serializing **X**, the runtime will keep the following pieces of information: (a) the actual type **Tr** of the object **X** is referring to (b) the **OuterXml** string of the document.

When de-serializing **X**, the runtime will create an instance of **Tr** (this assumes a constructor that does not take any parameters) and will call **LoadXml** providing the instance with the saved **OuterXml. X** will then be set to point to the newly created instance of **Tr**.

Compensation

If an error occurs and you need to undo or reverse the effects of a successfully committed transaction, you can do so by adding compensation code to your orchestration.

The compensation can be invoked after the transaction has completed its actions successfully. At that point, the state of the orchestration is known, and state information is available to the code in the compensation, which means that you can write code to act appropriately depending on the state of the orchestration when the transaction commits.

Compensations can also be provided on atomic transactions. These compensations can only be called after the atomic transaction commits. You need to write code to undo or reverse the path of the normal execution in the compensation.

The compensation block is flexible; it can contain any other shape, including another transaction scope.

Note You can do a compensation only once on a given scope.

Order of invocation

If you do not add your own compensation, the runtime engine performs a default compensation that invokes the compensations of any nested transactions within the current transaction. It first invokes the compensation of the most recently completed transaction, and works backward until all nested transactions have been compensated.

This holds true even when your compensation takes place inside a Loop shape: the compensations will be run in reverse order. First, the compensation for the last iteration of the loop will be invoked, then the compensation for the previous iteration, and so on.

If the default ordering does not fit your requirements, you can write your own compensation handler to explicitly call the compensation handlers of the nested scopes in the order you specify.

To add a Compensation Block

1. Right-click the **Scope** shape for the transaction to which you want to add a **Compensation Block**, and then click **New Compensation Block**.

2. A **Compensation Block** is added to the orchestration immediately following the associated **Scope** shape.

3. Inside the **Compensation Block** shape, add shapes to create the process for undoing a committed transaction.

In This Section

- Using the Compensate Shape

How to Use the Compensate Shape

If you are using nested transactions in your orchestration, you can add a **Compensate** shape in the compensation block or an exception block of a transaction scope. This enables your orchestration to explicitly perform compensation on a nested transaction. You specify which transaction you would like to be compensated in the **Compensate** shape, and any compensation code in the nested transaction will be run, provided the transaction committed successfully.

If you want to compensate more than one nested transaction, you add an additional **Compensate** shape for each transaction.

No **Compensate** shape is necessary if there is no other compensation code in an outer transaction; the compensation code of any nested transactions will be run automatically. The **Compensate** shape gives you control over the process by allowing you to decide whether or not you want a nested transaction to be compensated.

Procedures

To configure a Compensate shape

1. In the Properties window, select the compensation block to call from the **Compensation** drop-down list.

   The drop-down list will display all of the transactions which can be compensated, which includes the current transaction and any immediate child transactions of the current transaction. If you cannot see a transaction that you expect, it might be due to the relationship of the transactions.

   If you choose to compensate the current transaction, that means that the default handler will be invoked, and not an explicit compensation block (if there is one). This is a mechanism for automatically compensating directly nested transactions that have completed successfully.

Exceptions

Exceptions are raised when an error occurs in your orchestration. BizTalk provides various mechanisms for handling (and throwing) exceptions.

For a list of potential exceptions, see **Microsoft.XLANGs.BaseTypes Namespace**.

In This Section

- How Exceptions Are Handled

- Causes of Exceptions
- Catch Exception Block
- Using the Throw Exception Shape
- Fault Messages

How Exceptions Are Handled

When an exception occurs within a scope, each logical thread of execution in the scope is stopped. The runtime engine tries to find an exception handler for the appropriate exception.

If the exception handler is found that matches the specific type or one of its base types, control passes to that handler and its code runs.

Exception handlers are sequential; that is, they will be checked in order to see if they can handle a particular exception. To work properly, exception handlers must be placed so that those handling more specific types come first, followed by those handling more general types. This is so that you can ensure that an exception of a specific type is handled by the appropriate handler, rather than one designed to handle a base type.

If the exception handler completes normally, control passes to the surrounding scope. If no exception has been thrown in the surrounding scope, the orchestration continues to run. If the exception handler ends with a throw statement, the original exception is thrown again for the surrounding scope to act upon, unless you specify a different exception that you want to be thrown.

If no exception handler can be located, the default exception handler will run. The default exception handler for a scope will call the compensations for any nested transactions, and then throw the exception again. If you write your own exception handler, you can choose not to propagate the exception.

Causes of Exceptions

Exceptions can be generated within an orchestration in the following ways:

- By a **Throw Exception** shape, which throws an exception immediately and unconditionally. Control passes from the **Throw Exception** shape directly to the appropriate exception handler.
- By a time-out expiring in a long-running transaction. A predefined system exception—**Microsoft.XLANG.BaseTypes.TimeOutException**—is thrown in this case.

- By some other transaction failure. The runtime engine throws a system-defined message such as Microsoft.XLANG.BaseTypes.PersistenceException for these failures.

- By a user code exception. When calls to external user code are made within an orchestration, common language runtime classes that are called can throw exceptions. If these exceptions are not handled in the user code, they eventually propagate up into the scope in which the call to the user code is made.

- By some other system exception (for example, a persistence failure, another .NET or system exception such as type loader exception, or a data conversion error).

- By a sibling branch in a surrounding scope halting execution. Microsoft.XLANG.BaseTypes.ForcedTerminationException is thrown to each branch in this case, and you might want to add an exception handler to each. Such an exception handler cannot re-throw its exception, nor should it attempt to throw any other type of exception.

- By receipt of an external message that indicates a fault.

How to Add a Catch Exception Block

The **Catch Exception** block represents an exception handler. **Catch Exception** blocks are attached to the end of a **Scope** shape in Orchestration Designer. You can attach as many **Catch Exception** blocks as you need.

You can set up exception handlers to handle different kinds of exceptions. On each exception handler, you specify an exception type, which must be either a fault message or an object derived from the class **System.Exception**. If you do not specify an exception type, the exception block will be treated as a general exception handler, and can catch exceptions that do not derive from **System.Exception**.

If an exception is thrown that matches the specified type in an exception handler, that exception handler will be called. If some other exception is thrown, it will be handled by the default exception handler.

To add a Catch Exception block

1. Right-click the **Scope** shape to which you want to add a **Catch Exception** block, and then click **New Exception Handler**.

   A **Catch Exception** block is added to the orchestration immediately following the associated **Scope** shape.

2. In the Properties window, specify the following properties:

Property	Description
Exception Object Name	Assigns a name to the exception object caught by the exception handler.
Exception Object Type	Determines the object type (derived from System.Exception) that this exception handler will catch.

3. Inside the **Catch Exception** block, add shapes to create the process for handling the exception.

Using the Throw Exception Shape

You can explicitly throw exceptions in an orchestration by using the **Throw Exception** shape. When the throw is performed, the runtime engine will search for the nearest exception handler that can handle the type of exception being thrown.

First, the current orchestration is searched for an enclosing scope, and the associated exception handlers of the scope are considered in order to locate the appropriate handler for the type of exception that has been thrown.

If no appropriate exception handler is found, the orchestration that called the current orchestration is searched for a scope that encloses the point of the call to the current orchestration. This search continues until an exception handler is found that can handle the current exception.

An exact match for the exception is an exception class that is of the same class as, or a base class of, the run-time type of the exception being thrown.

After a matching exception handler is found, control is transferred to the first statement of the exception handler.

If the search for matching exception handlers fails, the orchestration halts. Transactions can help you minimize the impact of such an occurrence.

To configure a Throw Exception shape

1. In the Properties window, select an available object type to throw from the **Exception Object** drop-down list.

Fault Messages

Request-response ports can have fault messages associated with them, so that if something goes wrong after a request is sent, the responding service can communicate the error to the requester, in lieu of the response.

Each operation of a request-response port can handle an arbitrary number of different faults. A fault message can have any message type, but the message type must be unique for the operation, and it must not be the type used by the response in that port operation.

Receiving fault messages

If your port operation sends a request, then receives a response, you can use it to receive one or more different fault message types.

You can configure a **Catch Exception** block to handle an incoming fault message by selecting the appropriate fault from the request-response port operation as its Exception Object Type.

Sending fault messages

If your port operation receives a response, then sends a request, you can use it to send one or more different fault message types.

If for example your orchestration encounters an error condition and throws an exception, you can send a fault message from within the **Catch Exception** block that handles the exception. You construct a fault message of an appropriate type to convey the situation to the participating service, and specify that fault message on a Send shape that will use the corresponding fault in the port operation.

Transacted Orchestrations

Orchestrations can be transactional, just like scopes. In fact, an orchestration can itself be considered a scope. In general, the same rules apply for transacted orchestrations as for transacted scopes.

Orchestration compensation

If the **Transaction Type** property for your orchestration is set to long-running or atomic, you can also select a value for the **Compensation** property, which can be Default or Custom.

If you select **Custom** for the compensation, a **Compensation** tab will appear alongside the Orchestration Design Surface. It will look the same as the Orchestration Design Surface, and you can add shapes and ports to it in the same way.

Compensations only take place on orchestrations that are called by other orchestrations. You can compensate a specific orchestration instance by using the **Identifier** property on the **Call Orchestration** shape.

Using Business Rules in Orchestrations

You can create an instance of a Business Rules policy and execute it in your orchestration. To do this, you simply add a Call Rules shape inside an atomic transaction scope, and configure a policy on it.

For more information on the Business Rules engine and how to work with it, see Developing with Business Rules. For more information on orchestrations, see Creating Orchestrations.

In This Section

- Using the Call Rules Shape

How to Use the Call Rules Shape

The Call Rules shape enables you to create and execute an instance of a Business Rules policy. To configure the policy, click on the ellipsis button on the Configure Policy property in the Properties window.

Executing a Pipeline from within an Orchestration

new capability in BizTalk 2006 is the ability to synchronously call a pipeline from within an Orchestration. This allows orchestrations to leverage the message processing encapsulated within a pipeline (either send or receive) against a body of data without having to send that data through the messaging infrastructure. Envisioned uses of this feature include such things as allowing an orchestration to call a send pipeline in order to aggregate several messages into a single outgoing interchange. Conversely, an orchestration could call a receive pipeline to decode and disassemble an interchange obtained outside of the messaging infrastructure, without incurring the processing costs of going through the message box, etc.

Details

Orchestrations use methods in the **XLANGPipelineManager** class (in the **Microsoft.XLANGs.Pipeline** namespace) to call send or receive pipelines. A Receive pipeline consumes either a single message or an interchange and yields zero or more messages, just as when the pipeline executes in the context of receiving a message within BizTalk messaging. A Send pipeline consumes one or more messages and yields a single message or interchange, again, just as when the pipeline executes in the context of sending a message within BizTalk messaging.

Calling a Receive Pipeline

In order to call a receive pipeline from within an orchestration, the application calls the **ExecuteReceivePipeline()** method of the **XLANGPipelineManager** class. This method consumes a single interchange and returns a collection of zero or more messages (contained in an instance of the **ReceivePipelineOutputMessages** class). The syntax of this method is detailed in the .NET class library reference for the **XLANGPipelineManager** class.

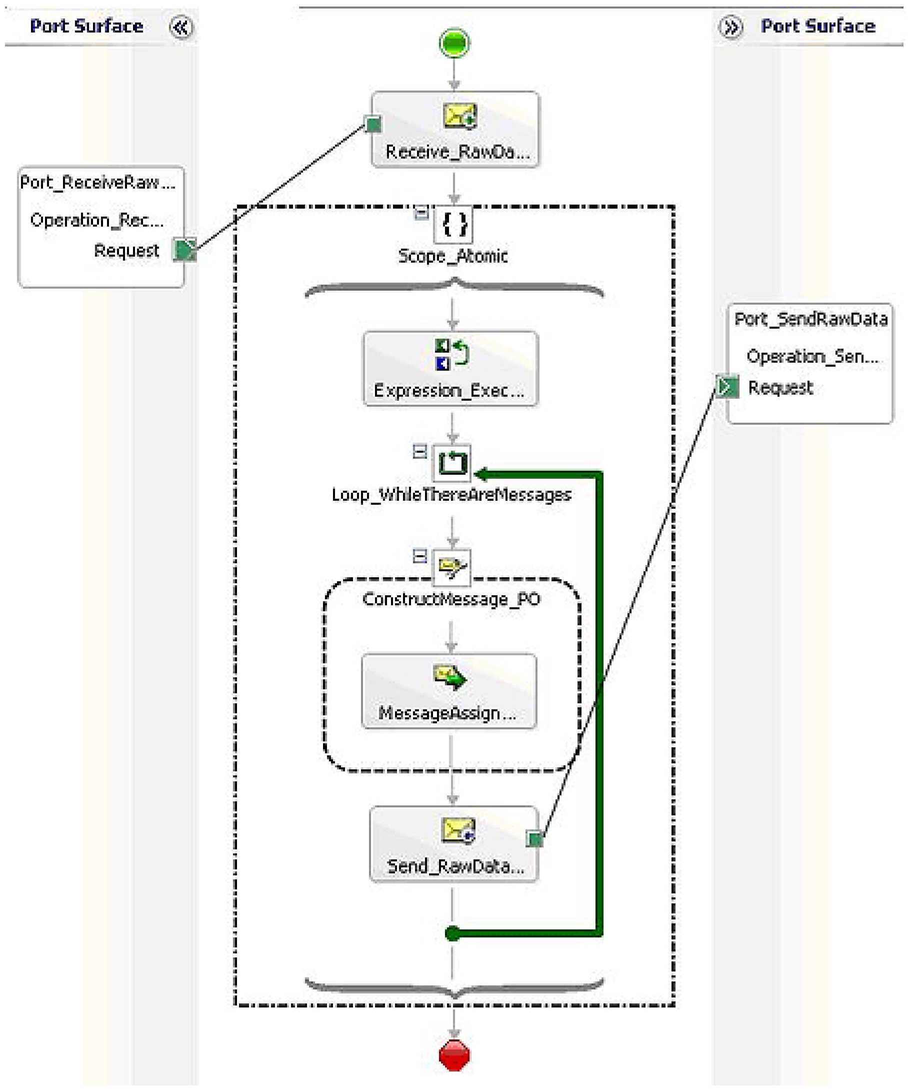
The prototype API for executing a receive pipeline from within an orchestration is:

```
// Execute receive pipeline

static public ReceivePipelineOutputMessages
ExecuteReceivePipeline(System.Type receivePipelineType, XLANGMessage
msg);
```

A call to a receive pipeline would typically be done in an **Expression** shape within the orchestration.

In order to call a receive pipeline from within an orchestration, the developer must reference the pipeline assembly in the orchestration project. An example of an orchestration that calls a receive pipeline:



For a more detailed example, see the SDK sample **Composed Message Processor (BizTalk Server Samples Folder)**

Calling a Send Pipeline

In order to call a send pipeline from within an orchestration, the application calls the **ExecuteSendPipeline()** method of the **XLANGPipelineManager** class. This method consumes a collection of one or more messages (contained in an instance of the **SendPipelineInputMessages** class) and returns a single interchange. The syntax of this method is detailed in the .NET class library reference for the **XLANGPipelineManager** class. Because execution of a send pipeline yields a new interchange, the call to **ExecuteSendPipeline()** method must be made within a message assignment shape, as such:

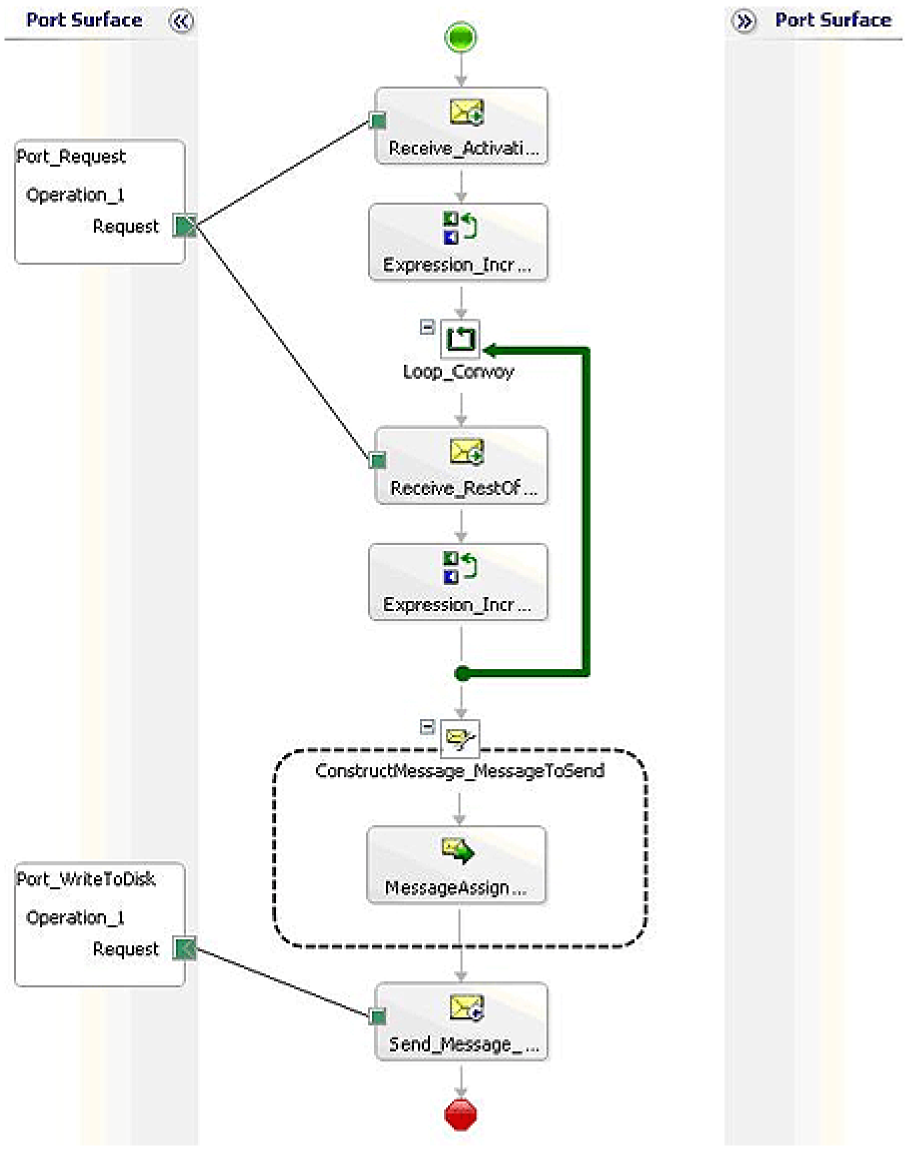
The prototype API for executing a send pipeline from within an orchestration is:

```
// Execute a send pipeline

static    public    ExecuteSendPipeline(System.Type    sendPipelineType,
SendPipelineInputMessages inputMsgs, XLANGMessage msg);
```

A call to a send pipeline must be done in a **Message Assignment** shape within the orchestration.

In order to call a send pipeline from within an orchestration, the developer must reference the pipeline assembly in the orchestration project. An example of an orchestration that calls a send pipeline:



For a more detailed example, see the SDK sample Aggregator.

Pipeline Execution - Behavioral Differences

The execution of a send or receive pipeline when called by an orchestration is predominantly the same as when the same pipeline executes within the messaging infrastructure (i.e. at receive location or send port). However, there are certain behavioral differences that are noted below.

Differences Within Pipeline Stages

The execution of the stages within a send or receive pipeline that is called from within an orchestration is nearly identical to the execution of those stages when the pipeline is called from the BizTalk messaging infrastructure, with the exceptions noted by stage below.

- Assembler/Disassembler: The assembler and disassembler stages will not process **Tracking Profile** data.

- Encoder/Decoder: The MIME encoder digitally signs messages using the certificate configured at the host with which the host is associated. The SMIME encoder encrypts messages using the certificate on the context of the message passed into the pipeline.

Schema Resolution

There are two schema lookup algorithms that are supported when executing a pipeline from an orchestration:

- Resolution by type

- Resolution by name

In cases where duplicate schemas are deployed, the algorithm's logic for selecting the appropriate schema is identical to that used when executing in the context of the messaging infrastructure.

Transactional Pipelines

Pipelines whose stages call transactional components will not have a transactional context available. Any call to **IPipelineContext.GetTransaction()** will throw **NotSupportedException**. This does not preclude execution of such a pipeline from an orchestration, but it does mean that the pipeline will have to detect and handle this situation.

Message Destination

Controlling message destination by pipeline components is not supported in this context. Setting the context properties **MessageDestination** or **SuspendOnRoutingFailure** will cause an **XLANGPipelineManagerException** to be thrown.

Pipeline Component Types

Pipeline components must be based on the following in order to be called from within an orchestration:

- .NET Framework v1.1
- .NET Framework v2.0
- COM

Restrictions

The following types of pipelines can **not** be executed from within an orchestration:

- Transactional pipelines
- Recoverable pipelines
- Pipelines which call the BAM interceptor API (a **NotSupportedException** will be thrown).
- The same pipeline instance cannot be executed in different branches of the parallel shape unless it is placed in a synchronized scope in every branch.
- Existing pipelines (assemblies) that were built against the BizTalk Server 2006 SDK. If this same pipeline is rebuilt against the BizTalk Server 2006 SDK, and it is not restricted from execution by an orchestration for any other reasons, then it can be called from within an orchestration.

Failure Modes and Effects

Any failure in pipeline execution which would have resulted in a suspended message were this pipeline to be called from within the BizTalk Server Messaging Infrastructure will instead result in an exception being thrown. The exception thrown is of type **Microsoft.XLANGs.Pipeline.XLANGPipelineManagerException**. This thrown exception can be handled in a catch block within the calling orchestration. If the orchestration does not catch the thrown exception, the XLANGs engine reports an error the text of which includes the exception information in the thrown exception.

The exception performs formatting of the error messages generated by the pipeline components.

The **Message** property of the **XLANGPipelineManagerException** class contains details of the pipeline's execution error. This detail is in the following format:

- There was a failure executing pipeline <pipeline type>. Error details <formatted error message>.

In this message, <pipeline type> is the name of the pipeline class and <formatted error message> is a description of the specific failure that occurred during pipeline execution.

For example, if an orchestration calls a receive pipeline and that pipeline's execution fails because none of the pipeline's components recognizes the message, the values of the **XLANGPipelineManagerException**'s properties would be:

XLANGPipelineManagerException property	**Value**
Message	There was a failure executing the receive pipeline "MyPipelines.ReceivePipeline". Error details: "Probe stage: No component in the disassemble stage recognized the message format.
Component	String.Empty

As another example, if an orchestration calls a send pipeline and that pipeline's execution fails because there is a validation failure, the text in the **Message** property of **XLANGPipelineManagerException** would be:

XLANGPipelineManagerException property	**Value**
Message	There was a failure executing the send pipeline "MyPipelines.SendPipeline". Error details: "Failed to validate the document: "The <element name> element is invalid - The value <element value> is invalid according to its datatype 'String' - The Pattern constraint failed.""
Component	“Microsoft.BizTalk.Component.XmlValidator”

Building and Running Orchestrations

After you have designed your orchestration, you can compile and run it to test it for completeness and correctness.

Once your orchestration has compiled without errors, you can deploy it in a test environment before making any necessary modifications and eventually deploying it to run in a production environment.

In This Section

- Building Orchestrations
- Running Orchestrations
- Debugging Orchestrations

How to Build Orchestrations

After you have completed an orchestration drawing, you build your BizTalk project into an assembly that encapsulates an executable orchestration.

During the build process, BizTalk Orchestration Designer examines each shape to determine whether it is complete and correct, and reports an error in the task list if it is not.

You have several options for building in Visual Studio:

- You can build the entire solution in which your orchestration resides.
- You can build a single project within the solution.
- You can skip the orchestration when building the project or solution.

If you want to build other components, including other orchestrations, but do not want to build a particular orchestration, you can indicate in the file properties for the orchestration's .odx file that you do not want to build it, and it will be skipped.

To build an orchestration

- After creating your orchestration, you need to build the BizTalk project that contains it before you can test or use the orchestration. You can build the entire solution, or a single project within the solution.

To build a solution

- In Solution Explorer, right-click the solution and select **Build Solution**.

To build a project

- Right-click a project and select **Build**.

To build a project or solution without compiling a particular orchestration

- Click on the .odx file corresponding to the orchestration, and in the Properties window, click on the **Build Action** property and select **None**.

In This Section

- Insufficient Configuration
- Build Errors in the Task List
- Importing BPEL4WS
- Exporting BPEL4WS

Insufficient Configuration

Insufficient configuration Smart Tag

Build Errors in the Task List

When you build your project, or solution, the results will appear in the Output window, while individual errors and warnings will appear in the task list.

Errors and warnings appear in the task list. You can double-click on the error, and the focus will be applied to the object that is not correctly configured.

How to Import BPEL4WS

You can import from existing BPEL4WS to create an orchestration.

To import BPEL4WS into an orchestration

1. Create a new project.
2. From the BizTalk Project types, double-click BizTalk Server BPEL Import Project, or select BizTalk Server BPEL Import Project and press OK.
3. In the wizard, select the BPEL, WSDL and XSD files that should be imported to form the new BizTalk project. Include all files that are referenced via import and include statements.
4. Select WSDL files for invoked Web services.

   You can now modify or deploy the new orchestration.

Import restrictions on BPEL4WS

- When importing BPEL and WSDL, make sure that the Name property of the WSDL definition node and the BPEL process node do not match.

- Do not use XLANG/s reserved words in BPEL4WS that you import. For a complete list, see **XLANG/s Reserved Words**.

- Only XSD predefined simple types are supported.

- xsd:QName is not supported; it is imported as System.String. Use xsd:string instead.

- Consider using canonical XPaths when importing BPEL4WS.

  It is good practice to import only canonical XPaths in order to achieve optimal performance. The entire path from root to the promoted node must be spelled out by using '/*[local-name()="someName" and namespace-uri()="someUri"]'.

  If you import a non-canonical XPath, you can remove a promotion and repromote the same field in order to have the Schema Editor create the correct canonical XPath.

  Example: (targetNamespace = http://BizTalk_Server_Project3.Schema1)

- XPath - /*[local-name()='Root' and namespace-uri()='http://BizTalk_Server_Project3.Schema1']/*[local-name()='promotedField' and namespace-uri()='']

Canonical XPath	**Non-Canonical XPath**
BizTalk Editor displays a special icon ( ) to denote that the field has been promoted. Usage of canonical XPath expressions to promote fields improves performance through more efficient traversal of the XML	BizTalk Editor does not display a special icon. Both the compiler and the promotion dialog give warnings. There is a linear but nontrivial effect on performance as the message size increases.

How to Export BPEL4WS

You can export an existing BizTalk orchestration to BPEL4WS.

If you are exporting, BPEL4WS compliance for compilation requires that orchestrations contain only features that are common between XLANG/s and BPEL4WS, or features that can be translated into BPEL4WS without affecting behavior.

Export restrictions on orchestrations for BPEL4WS compliance

- You cannot use the Call Orchestration shape or the Start Orchestration shape.

- You cannot use the Transform shape.

- You cannot invoke methods on custom .NET components.

- You cannot apply a timeout to a long-running transaction.
- Your orchestration cannot take parameters.
- Callable compensation handlers cannot have parameters.
- Variable types must be supportable in XPATH.
- You cannot use the Suspend shape.
- Literal values must be one of the following types:

  boolean, char, byte, sbyte, int32, uint32, int64, uint64, single, double, string

- Arithmetic operators are allowed only on operands of following numeric types:

  byte, sbyte, int32, uint32, int64, uint64, single, double

- Relational operators can not be applied to type char.
- You cannot make a reference to a servicelink property in an expression.
- You cannot perform any actions between a **Send** shape and a **Receive** shape that use the same outbound request-response port.
- You cannot indirectly reference a web service, as through a reference to another project that contains a reference. You must explicitly reference the web service in your project.
- You cannot specify a constant DateTime or TimeSpan in a delay. Instead, use one of the conversion classes in the System.Xml namespace:

  For a constant DateTime: System.Xml.XmlConvert.ToDateTime, e.g System.Xml.XmlConvert.ToDateTime("2004-04-15")

  For a constant TimeSpan: System.Xml.XmlConvert.ToTimeSpan, e.g System.Xml.XmlConvert.ToTimeSpan("2004-04-15")

⊟To export an orchestration to BPEL4WS

1. Add a new item of type BizTalk Orchestration to your project.
2. Click on the design surface to bring up the Orchestration Properties window.
3. Set **Module Exportable** to True.
4. Type in the namespace you want for **Module XML Target Namespace**.
5. Set **Orchestration Exportable** to True.

6. Type in the namespace you want for **Orchestration XML Target Namespace**.

7. In Solution Explorer, right-click on the .ODX file for your orchestration.

8. Select **Export to BPEL**.

Your orchestration will be exported to BPEL4WS. See the Output Window and Task List to confirm success or diagnose problems. Once your export succeeds, a .WSDL file and a .BPEL file will be created in your project directory.

Running Orchestrations

Orchestration instances are designed to be triggered either by an explicit call from another orchestration—using a **Call Orchestration** shape or **Start Orchestration** shape—or by receipt of an activation message. You should design your orchestration accordingly, and either set the **Activate** property on a **Receive** shape to true, or make sure that a calling orchestration exists and is configured properly to run the new orchestration.

Before any instances can run, you must use BizTalk Server Administration to direct the orchestration engine to begin processing the orchestration. See Starting an Orchestration with the BizTalk Administration Console. When an orchestration is invoked from another orchestration, or a message is presented to the engine that matches the criteria in an activation receive, the engine creates a new instance of the orchestration and runs that instance. It can run many different instances concurrently.

Calling and starting orchestrations

The **Call Orchestration** shape and **Start Orchestration** shape can be used to activate another orchestration. In both cases, the caller can pass in parameters to exchange information with the other orchestration. For more information, see Adding Parameters to Orchestrations.

Using activation receives

The **Receive** shape might also use a filter expression to require further criteria for activation. If the message is of the correct type and some property or properties of the message meet all of the criteria in the filter expression, the **Receive** shape accepts the message and the orchestration is activated. Such a Receive shape is referred to as an *activation receive*.

Debugging Orchestrations

Your primary vehicle for debugging and monitoring orchestrations will be BizTalk Health and Activity Tracking (HAT), a powerful tool that enables you to flexibly view and report on high-level progress or low-level details of your business process and the components that implement it.

In This Section

- Tracking Orchestrations with HAT
- Remote Debugging Configuration
- Custom Debugging

Tracking Orchestrations with HAT

Health and Activity Tracking (HAT) is a Web-based user interface that you can access over HTTP to see tracking data through different views and queries, depending on your role. You can monitor the progress of your business process, you can examine state as your business process runs, and you can create custom views to get just the information that is important to you. For details on how to use HAT to debug your orchestration, see Orchestration Debugger User Interface.

You can use HAT to query a suspended instance, resume the instance in debug mode, and add appropriate breakpoints using the Technical Details View. This will enable you to trace the activities and debug messages step by step.

You can set breakpoints to track the following events:

- The start and finish of an orchestration
- The start and finish of a shape
- The sending or receipt of a message
- Exceptions

At each breakpoint, you can examine information about local variables, messages and their properties, ports, and role links.

Remote Debugging Configuration

Microsoft.XLANGs.BizTalk.Client.dll.config. The server configuration is specified in BTSNTSvc.exe.config. The following is a listing of the default configuration for each.

Client (Microsoft.XLANGs.BizTalk.Client.dll.config)

Server(BTSNTSvc.exe.config)

Configurable Parameters

The default ensures maximum security configuration. However it is left to the user to change these defaults and these files are ACL'ed since they are in the program files folder.

The element <provider/> is optional and if not provided will cause the channels not to be mutually authenticated using the custom sinks. However this is a dangerous option to turn off as it will open up the channels. This can be done for better performance and when security attacks are not a concern.

The channel element can have property rejectRemoteRequests = true which will enable only local calls and reject remote requests.

The securityPackage attribute in the <serverProviders/> element can have any of the following values:

- negotiate
- ntlm
- Kerberos

The authenticationLevel attribute in the <serverProviders/> element can have any of the following values:

- packetPrivacy - the messages will be encrypted/decrypted
- packetIntegrity – the messages will be signed/verified
- call - the messages will be sent as is

The ref attribute in the <channel/> element can be changed to tcp or http. The port and name attribute in the <channel/> element can be changed as well to explicit values.

For more information, see .NET Framework Developer's Guide (Channel and formatter configuration properties).

Custom Debugging

If your orchestration is going to be exercised in a test environment or you are creating a prototype and wish to modify the values of message fields and orchestration variables, you can write a simple custom debugger.

You can do this by creating a debugging DLL with a class that includes a method which takes as input a message with a format defined in your orchestration and referenced in the debug DLL. For more information on passing a message as a parameter, see Expressions.

This method can bring up a debug dialog that includes a combo box or other control to allow user modification of values, puts the edited message back together, and passes it back out as the return value.

Set up a Boolean variable to indicate whether or not your orchestration is in debug mode, then wherever you have a point in your orchestration at which you would like to be able to modify values, you can add a **Decide** shape with one live branch that runs only when your debug mode variable is set to True, or when a particular condition arises that you want to examine. You call your method from an **Expression** shape in the live branch of the **Decide**. When you no longer need to debug, you set your debug mode variable to False, or remove the **Decide** shape(s) altogether and recompile.

To Debug a .NET component called by an Orchestration

The following steps demonstrate how to debug a .NET component called by an Orchestration:

1. Open the Visual Studio project for your component.
2. Set a breakpoint in your component on the method that is called by the orchestration.
3. Click the **Debug** menu and select **Attach to Process…** to display the **Attach to Process** dialog.
4. Click the **Select…** button next to the **Attach to:** text box to display the **Select Code Type** dialog.
5. Click to select the option to **Debug these code types:** and select **Managed** and then click the **OK** button.
6. Click to select the **BTSNTSvc.exe** process from **Available Processes** and then click the **Attach** button.
7. Send a message to your orchestration through a receive port.
8. The .NET component should be stopped in the breakpoint.
9. You can perform debugging as usual with Visual Studio 2005.

Working with the Orchestration Engine

The orchestration engine creates instances of orchestrations, runs them, maintains their state, and performs various optimizations to maximize scalability, throughput, and efficient resource utilization.

The orchestration engine also implements a reliable system shutdown and recovery model through strategic use of persistence, dehydration and rehydration.

In This Section

- Scalability and Performance
- Reliability
- Runtime Validation

Scalability and Performance

The orchestration engine has been designed with performance in mind: orchestration code is now compiled rather than interpreted, context-switching is eliminated by flow management and load balancing, and the engine is fully integrated with .NET, which enables you to take advantage of .NET and eliminate the overhead of COM interopability issues.

Orchestration scalability is enhanced because an orchestration can be run on several servers in a BizTalk group, even dehydrating an orchestration instance—saving its state and removing it from memory until it is needed—and rehydrating it on another server.

In This Section

- Persistence
- Dehydration and Rehydration
- Performance Counters

Persistence

The orchestration engine saves to persistent storage the entire state of a running orchestration instance at various points, so that the instance can later be completely restored in memory.

The state includes:

- The internal state of the engine, including its current progress.

- The state of any .NET components that maintain state information and are being used by the orchestration.
- Message and variable values.

Persistence Points

The orchestration engine saves the state of a running orchestration instance at various points. If it needs to rehydrate the orchestration instance, start up from a controlled shutdown, or recover from an unexpected shutdown, it will run the orchestration instance from the last persistence point, as though nothing else had occurred. For example, if a message is received but there is an unexpected shutdown before state can be saved, the engine will not record that it has received the message, and will receive it again upon restarting. The engine will save the state of an orchestration in the following circumstances:

- The end of a transactional scope is reached.

  The engine saves state at the end of a transactional scope so that the point at which the orchestration should resume is defined unambiguously, and so that compensation can be carried out correctly if necessary.

  The orchestration will continue to run from the end of the scope if persistence was successful; otherwise, the appropriate exception handler will be invoked.

  If the scope is transactional and atomic, the engine will save state within that scope.

  If the scope is transactional and long-running, the engine will generate a new transaction and persist the complete state of the runtime.

- A debugging breakpoint is reached.
- A message is sent. The only exception to this is when a message is sent from within an atomic transaction scope.
- The orchestration starts another orchestration asynchronously, as with the **Start Orchestration** shape.
- The orchestration instance is suspended.
- The system shuts down under controlled conditions. Note that this does not include abnormal termination; in that case, when the engine next runs, it will resume the orchestration instance from the last persistence point that occurred before the shutdown.
- The engine determines that the instance should be dehydrated.
- The orchestration instance is finished.

Serialization

All object instances that your orchestration refers to directly or indirectly (as through other objects) must be serializable for your orchestration state to be persisted. There are two exceptions:

- You can have a nonserializable object declared inside an atomic transaction. You can do this because atomic scopes do not contain persistence points.
- System.Xml.XmlDocument is not a serializable class; it is handled as a special case and can be used anywhere.

Dehydration and Rehydration

When many long-running business processes are running at the same time, memory and performance are potential issues. The engine addresses these issues by "dehydrating" and "rehydrating" orchestration instances.

Dehydration

The engine might determine that an orchestration instance has been idle for a relatively long period of time. It calculates thresholds to determine how long it will wait for various actions to take place, and if those thresholds are exceeded, it will dehydrate the instance. This can occur under the following circumstances:

- When the orchestration is waiting to receive a message, and the wait is longer than a threshold determined by the engine.
- When the orchestration is "listening" for a message, as it does when you use a **Listen** shape, and no branch is triggered before a threshold determined by the engine. The only exception to this is when the **Listen** contains an activation receive.
- When a delay in the orchestration is longer than a threshold determined by the engine.
- Between retries of an atomic transaction.

The engine then dehydrates the instance by saving the state, and frees up the memory required by the instance. By dehydrating dormant orchestration instances, the engine makes it possible for a large number of long-running business processes to run concurrently on the same machine.

Rehydration

The engine can be triggered to rehydrate an orchestration instance by receipt of a message, or by the expiration of a timeout. It will then load the saved orchestration instance into memory, restore its state, and run it from the point where it left off.

Rehydration on a different machine

An orchestration can be configured to run on more than one server. After an orchestration instance has been dehydrated, it can be rehydrated on any of these servers. If one server goes down, the engine will continue to run the orchestration on a different server, continuing from its previous state. The engine will also take advantage of this feature to implement load balancing across servers.

Reliability

The orchestration engine maintains the state of orchestration instances, shuts down in a controlled manner, and recovers flexibly to offer a very reliable model for business processes. For more information, see Persistence and Working with the Orchestration Engine.

In This Section

- System Shutdown
- Recovery

System Shutdown

When the orchestration engine is asked to shut down, it saves control information as well as the current state of all running orchestration instances, so that it can resume running them when it is started again.

The engine will continue to run the instance until it reaches the next persistence point, when it will save the orchestration state and shut down the instance before shutting itself down.

Recovery

The engine regularly saves to persistent storage the state information of an orchestration instance, and it also saves state in the event of a system shutdown. When an orchestration instance fails abnormally for whatever reason, the instance can be recovered from the last persisted state, and it can continue to run as if there were no interruption. This is true even if the original server that the instance ran on goes out of service for some reason; the instance can simply resume running on a separate machine. Because of this multi-server recovery model, you no longer need clustering.

Runtime Validation

You can configure the orchestration engine to perform various runtime validations that can help you test your orchestration and diagnose configuration or data errors that might arise.

You can set flags in BTSNTSvc.exe.config, a configuration file that you can create or edit in the same directory as BTSNTSvc.exe (normally in the BizTalk deployment directory). These flags will direct the engine to validate assemblies, schemas, and correlations.

If you want to disable a validation, remove the flag entirely from the configuration file.

When all validations are on, the engine will validate assemblies, schemas, and correlation.

For more information and examples of BTSNTSvc.exe.config, see **Orchestration Engine Configuration**.

Validate Assemblies

The orchestration engine will verify that any assemblies referenced by the orchestration are available. For the validation to succeed, all referenced assemblies must be in the Global Assembly Cache (GAC) when the first instance of the orchestration is activated. If the validation fails, an error will be logged to the application log and the orchestration will be suspended.

Validate Schemas

Any time an XSD part is assigned, the orchestration engine will validate the part's data against its schema. If the validation fails, the error will be logged to the application log and an exception will be thrown.

Validate Correlation

The orchestration engine will confirm that the property values that are specified for correlation with a given instance of an orchestration are reflected in any messages that are sent from that orchestration instance. If **validateCorrelation** is not set, the engine will assume that the sent message contains the correct correlation values, and no check will be performed.

If any correlation validation fails, the engine will log an error to the application log and throw an exception of type **CorrelationValidationException**.

By default, **validateCorrelation** is not set.

Creating Business Rules Using Business Rules Editor

Business rules (or business policies) define and control the structure, operation, and strategy of an organization. Business rules may be formally defined in procedure manuals, contracts, or agreements, or may exist as knowledge or expertise embodied in employees. Business rules are dynamic and subject to change over time, and can be found in all types of applications. Finance and insurance, e-business, transportation, telecommunications, Web-based services, and personalization are just a few of the many business domains that are governed by business rules. Each of these business domains shares the need to convey business strategies, policies, and regulations to information technology (IT) personnel for inclusion into software applications.

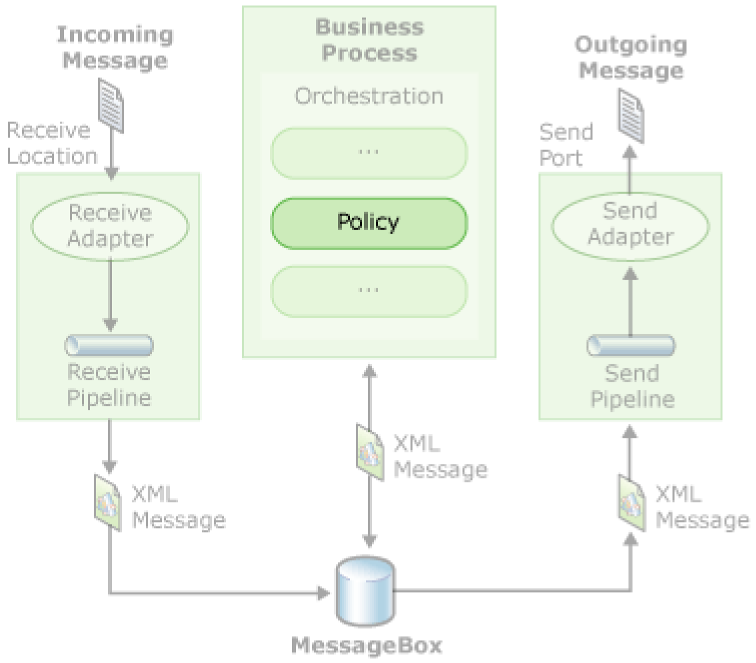
Traditional procedural and object-oriented programming languages, such as C, C++, and Microsoft® Visual Basic®, are oriented towards programmers. Even advanced object-oriented languages, such as Java and C#, are still primarily programmers' languages. The traditional software development cycle of design, develop, compile, and test requires substantial time and coordination, and does not enable nonprogrammers to participate in the maintenance of automated business policies. The Business Rules Framework addresses this problem by providing a development environment that allows rapid application creation without the lengthy cycle of traditional application programming. For example, business policies constructed by using this framework can be updated without recompiling and redeploying the associated orchestrations.

The Business Rules Framework is tightly integrated with Microsoft® BizTalk® Server 2006, and developers can use the following features to build and manage business rules:

- A high-performance rule engine that implements an inference mechanism to evaluate the business rules.
- A rich set of application programming interfaces (APIs) for developing rule-based applications.
- A graphical user interface, the Business Rule Composer, which developers, business analysts, and administrators can use in various ways to efficiently develop and apply rules and policies.
- A seamless integration with BizTalk orchestrations, which enables you to invoke a business policy or a set of business rules from a BizTalk orchestration.
- The Rule Engine Deployment Wizard, which enables you to rapidly import or export business rules or the vocabularies used by the rules, as well as to deploy or undeploy these rules.

The business rules (policy) you create by using the Business Rules Framework can be used in an orchestrated business process, as shown in the following figure.

Business Policy



This section provides conceptual information about how you can leverage the Business Rules Framework and use the tools that BizTalk Server 2006 provides to develop business rules.

In This Section

- About the Business Rules Framework
- Using the Business Rule Composer
- Creating Business Rules
- Business Rules Framework Security
- Programming with Business Rules Framework
- Rule Engine Configuration and Tuning Parameters

About the Business Rules Framework

The Business Rules Framework is a Microsoft® .NET-compliant class library. It provides an efficient inference engine that can link highly readable, declarative, semantically rich rules to any business objects (.NET components), XML documents, or database tables. Application developers can build business rules by constructing rules from small building blocks of business logic (small rule sets) that operate on information (facts) contained in .NET objects, database tables, and XML documents.

This design pattern promotes code reuse, design simplicity, and modularity of business logic. In addition, the rule engine does not impose on the architecture or design of business applications. In fact, you can add rule technology to a business application by directly invoking the rule engine, or you can have external logic that invokes your business objects without modifying them. In short, the technology enables developers to create and maintain applications with minimal effort.

In planning development of a rule-based application, you first need to determine how rules will fit into your business processes. Your application will create an instance of a policy and supply it with data, or facts, on which to operate. The policy object encapsulates the rule engine and provides a single point of entry through which to run it.

You also will need to plan for the development and testing of your rules design. You must consider how you are going to deploy and update your policies. You will likely want to track the progress of your rule engine's execution and monitor its current state.

Account for the following steps as you plan your rules development:

1. Plan how to incorporate your rules into your application.

2. Identify the business logic that you want to represent with rules in your application. The term "business logic" can refer to many things; an example of business logic is "Purchase orders for more than five hundred dollars must be approved by a manager."

3. Identify data sources for your rule elements. You can optionally define and publish vocabularies (domain-specific nomenclature that represents underlying bindings).

4. Define rules from vocabulary definitions or directly from data bindings, and from them compose a policy that represents your business logic.

5. Test and debug the policy with sample facts. You can either use the Test Policy functionality in the Business Rule Composer or use **Policy** or **PolicyTester** classes to execute from an application, command-line program, or orchestration.

6. Publish the policy version to the rule store.

7. Deploy the policy version.

8. Instantiate and build the short-term fact list in the hosting application. Use the **Call Rules** shape in an orchestration to execute your business policy or programmatically instantiate a policy version in your hosting application.

9. Monitor and track rule execution as needed.

In This Section

- **Business Rules Concepts**
- Business Rules Framework Architecture
- Facts
- Rule Engine

Rules

Business rules are declarative statements that govern the conduct of business processes. A rule consists of a condition and actions. The condition is evaluated, and if it evaluates to **true**, the rule engine initiates one or more actions.

Rules in the Business Rules Framework are defined by using the following format:

IF *condition* THEN *action*

Consider the following example:

IF amount is less than or equal to available funds

THEN conduct transaction and print receipt

This rule determines whether a transaction will be conducted by applying business logic, in the form of a comparison of two monetary values, to data or facts, in the form of a transaction amount and available funds.

You can use the Business Rule Composer to create, modify, version, and deploy business rules. Alternatively, you can perform the preceding tasks programmatically.

Conditions

A condition is a true/false (Boolean) expression that consists of one or more predicates that are applied to facts.

In our example, the predicate *less than or equal to* is applied to the facts *amount* and *available funds*. This condition will always evaluate to either **true** or **false**.

Predicates can be combined with the logical operators **AND**, **OR**, and **NOT** to form a logical expression that is potentially quite large, but will always evaluate to either **true** or **false**.

Actions

Actions are the functional consequences of condition evaluation. If a rule condition is met, a corresponding action or actions are initiated.

In our example, "conduct transaction" and "print receipt" are actions that are carried out when, and only when, the condition (in this case, "IF amount is less than or equal to available funds") is true.

Actions are represented in the Business Rules Framework by invoking methods or setting properties on objects, or by performing set operations on XML documents or database tables.

Facts

Facts are the data upon which rules operate. In our example, "amount" and "available funds" are facts. For more information, see Facts.

Policies

A policy is a logical grouping of rules. You compose a version of a policy, save it, test it by applying it to facts, and, when you are satisfied with the results, publish it and deploy it to a production environment.

Policy composition

You can compose policies in the Business Rule Composer by constructing rules from facts and definitions. A policy can contain an arbitrarily large set of rules, but typically you compose a policy from rules that pertain to a specific business domain within the context of the application that will be using the policy.

Policy testing

You can effectively perform a test run of your policy before it is published and deployed in a production environment. The Business Rule Composer allows you to supply instances of facts to a policy, run the policy, and view its output. The output includes fact activity, rule execution, condition evaluation, and updates to the agenda.

Policy versions

After you have defined all the rules in your policy, you can publish the policy version. In this way the policy is locked down, and its behavior is well-defined.

A given policy version can be used under a given set of circumstances in your business environment, and replaced by another version when those circumstances change. Also, both new and old versions can be used simultaneously by different applications.

Policy deployment

When your policy is ready to be run in a production environment, you can deploy it to make it available to a hosting application.

Dynamic policy updates

Dynamic policy updates allow you to modify policies independently of a running business process. You can create and deploy an updated version of the policy, and the hosting application can incorporate the update in near real time. This update does not require you to change any code, and thus you can avoid the overhead of redeveloping and redeploying the application.

Vocabularies

The terms used to define rule conditions and actions are usually expressed by domain or industry-specific nomenclature. For example, an e-mail user writes rules in terms of messages "received from" and messages "received after," while an insurance business analyst writes rules in terms of "risk factors" and "coverage amount."

Underlying this domain-specific terminology are the technology artifacts (objects, database tables, and XML documents) that implement rule conditions and rule actions. Vocabularies are designed to bridge the gap between business semantics and implementation.

For example, a data binding for an approval status might point to a certain column in a certain row in a certain database, represented as an SQL query. Instead of inserting this sort of complex representation in a rule, you might instead create a vocabulary definition, associated with that data binding, with a friendly name of "Status." Subsequently you can include "Status" in any number of rules, and the rule engine can retrieve the corresponding data from the table.

A vocabulary is a collection of definitions consisting of friendly names for the facts used in rule conditions and actions. Vocabulary definitions make the rules easier to read, understand, and share by people in a particular business domain.

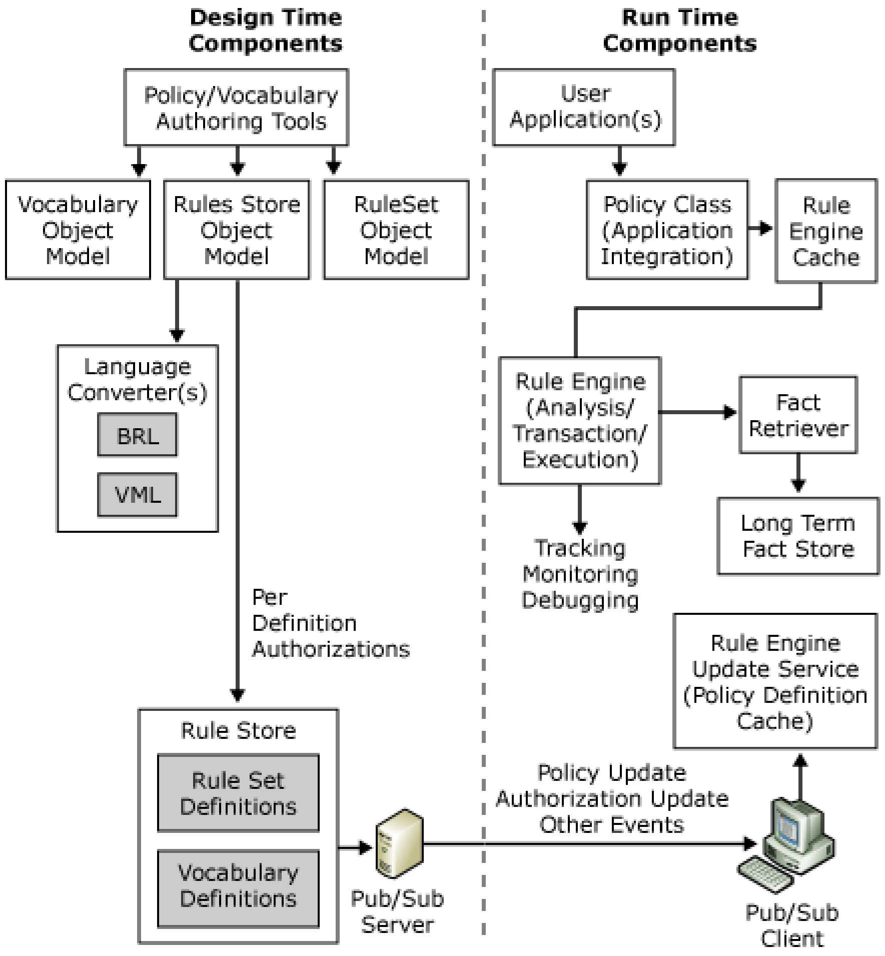
You can use the Business Rule Composer to define vocabularies that are then persisted into the shared rule store. Vocabularies can also be consumed by tool developers responsible for integrating rule authoring into new or existing applications.

Before you can use a vocabulary, it must be stamped with a version and published in your rule store. This is to guarantee that the definitions in the vocabulary will not change, to preserve referential integrity. This means that any policies that use that particular version of the vocabulary will not fail unexpectedly due to changes in the underlying vocabulary.

Business Rules Framework Architecture

The following figure shows the component architecture of the Business Rules Framework.

Microsoft Business Rules Framework Architecture



Some of the components of the framework are described in the following paragraphs.

Policy class

A **Policy** object is a single instance of a business policy, and provides the interface that is used by rule-based applications. It provides an abstraction that frees the application developer from concern about the location of the rule store, extracting rule sets from the rule store, instantiating instances of the underlying rule engine, and ensuring that long-term facts are asserted into the engine. In many scenarios, a rule-based application uses concurrent instances of the **Policy** object. These concurrent instances can represent the same policy, different versions of the same policy, or different versions of different policies.

RuleEngine class

The **RuleEngine** object serves as the execution engine for business policies. It uses four plug-in components (translator, inference engine, and tracking interceptor) for implementation. A **RuleEngine** object takes a **RuleSet** object representing a business policy as input and uses the analyzer, translator, inference engine, and tracking interceptor configured for the rule set to implement the business policy defined by the rule set.

Fact retriever

A fact retriever is an optional, user-defined, plug-in component that is responsible for gathering long-term facts for use by business policies. For more information, see Creating a Fact Retriever.

Before execution, a **Policy** object instance provides its **RuleEngine** instance to the fact retriever, giving it the opportunity to update the set of Facts in the rule engine's working memory. For more information, see Short-Term Facts vs. Long-Term Facts.

Rule Engine Update service

The Rule Engine Update service provides dynamic business policy updates in a distributed environment. It is an autostart Microsoft® Windows NT® service application that is responsible for subscribing to policy deployment and undeployment events that occur when business policies are changed.

In a typical enterprise scenario, business policies are deployed after being updated, tested, and versioned. Deployment consists of adding the updated policy to a secure, persistent rule store and optionally executing logic on the store to publish information about the updated policy to all interested parties (note that information about the policy is published and not the policy itself).

The Rule Engine Update service, running on a server hosting rule-based applications, receives the policy update notification and caches the information for subsequent use. The use of a pub/sub model for policy updates enables you to change business policies in near real time without service downtime or interruption.

Policy/vocabulary authoring tools

The Business Rule Composer in Microsoft® BizTalk® Server 2006 provides policy and vocabulary authoring capabilities to both end users and developers.

Rule store

A rule store is a repository for business policies and vocabularies. The repository can be a simple file or a secure, scalable, persistent, and reliable database such as Microsoft® SQL Server™. (SQL Server is used as the default rule store for the framework).

Caching

The Business Rule Engine Framework provides a caching mechanism for **RuleEngine** instances. Each **RuleEngine** instance contains an in-memory representation of a specific policy version.

The following steps describe the process when a new **Policy** instance is instantiated (either with a call on the API or execution of the **Call Rules** shape):

1. The **Policy** object requests a **RuleEngine** instance from the rule engine cache.

2. If a **RuleEngine** instance for the policy version exists in the cache, the **RuleEngine** instance is returned to the **Policy** object. If a **RuleEngine** instance is not available, the cache will create a new instance. When a **RuleEngine** instance is instantiated, it will, in turn, create a new fact retriever instance if one is configured for the policy version.

When the Execute method is called on the **Policy** object, the following steps occur:

1. The Policy object calls the **UpdateFacts** methodon the fact retriever instance if a fact retriever exists. The fact retriever's implementation of the method may assert long term facts into the working memory of the **RuleEngine**.

2. The **Policy** object asserts the short term facts contained in the **Array** that was passed in the **Execute** call.

3. The **Policy** object calls **Execute** on the **RuleEngine**.

4. The **RuleEngine** completes execution and returns control to the **Policy** object**.**

5. The **Policy** object retracts the short term facts from the **RuleEngine**. The long term facts asserted by the fact retriever will remain in the working memory of the rule engine.

After the **Dispose** method is called on the **Policy** object, the **RuleEngine** instance will be released back to the rule engine cache.

The rule engine cache will have multiple rule engine instances for a given policy version if the load requires it, and each rule engine instance has its own fact retriever instance.

Facts

Facts are information about the world. Facts can originate from many sources (event systems, objects in business applications, database tables, and so on), and must be fed into the rule engine by using one of the following elements:

- NET objects (methods, properties, and fields)
- XML documents (elements, attributes, and document subsections)
- Database rowsets (values from table column)

In the Business Rule Composer, you can use the Facts Explorer to browse and bring data into your rules from various sources.

In This Section

- .NET Object Facts
- Typed Facts
- Short-Term Facts vs. Long-Term Facts

.NET Object Facts

In the Business Rule Composer, you can specify a .NET assembly as a data source. You can subsequently select a class or class member from the assembly, and drag it onto a vocabulary definition or rule.

Asserted .NET object processing is the most straightforward of the processing types. The engine processes each object as a unique instance. For more information about the **Assert** function, see Engine Control Functions.

Typed Facts

Typed facts are classes that implement the **ITypedFact** interface: **TypedXmlDocument**, **DataConnection**, **TypedDataTable**, and **TypedDataRow**.

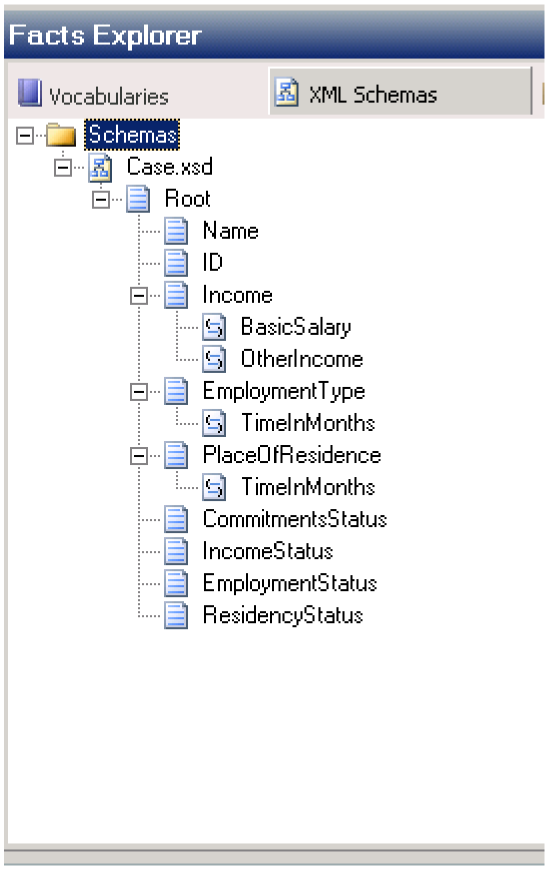
TypedXmlDocument

The **TypedXmlDocument** class represents the XML document type in the Business Rules Framework. When you use a node of an XML document as an argument in a rule, two XPath expressions are created: the Selector and Field bindings.

If the node has no child nodes, a Selector binding (also known as an XmlDocument binding) is created to the node's parent node and a Field binding (also known as an XmlDocumentMember binding) is created to the node itself. This Field binding is relative to the Selector binding. If the node has child nodes, a Selector binding is created to the node and no Field binding is created.

Suppose that you have the following schema.

Case.xsd



If the Income node is selected, only a Selector binding is created, because the node has child nodes. The default XPath expression in the **XPath Selector** property of the Property pane contains:

/*[local-name()='Root' and namespace-uri()='http://LoansProcessor.Case']/*[local-name()='Income' and namespace-uri()='']

However, if the Name node is selected, both a Selector binding and a Field binding are created. The binding information looks like.

Property	**Value**
XPath Field	*[local-name()='Name' and namespace-uri()='']
XPath Selector	/*[local-name()='Root' and namespace-uri()='http://LoansProcessor.Case']

The default XPath expressions for the XML nodes can be changed before dragging the node into a rule argument, and the new binding information is persisted with the policy. Note, however, that any edits that are made to the XPath expressions must be reentered in the Business Rule Composer when the schema is reloaded.

When vocabulary definitions are created for XML nodes, the XPath expressions for the bindings are similarly defaulted based on the rules described earlier, but are editable in the Vocabulary Definition Wizard. Changes to the expressions are persisted with the vocabulary definition and are reflected in any rule arguments built from the definitions.

DataConnection

DataConnection is a .NET class provided in the **RuleEngine** library. It contains a .NET **SqlConnection** instance and a **DataSet** name. The **DataSet** name enables you to create a unique identifier for the **SqlConnection** and is used in defining the resulting type.

The **DataConnection** class provides a performance optimization to the rule engine. Rather than asserting into the engine very large database tables (**TypedDataTable** class), which may contain many database rows (**TypedDataRow** class) that are not relevant to the policy, you can assert a lightweight **DataConnection**. When the engine evaluates a policy, it dynamically builds a SELECT query based on the rule predicates/actions and queries the **DataConnection** at execution. For example, suppose you have the following rule:

The following SQL query is generated by from the rule:

The results of the query are asserted back into the engine as data rows.

When you select a database table/column to use in a rule condition or action, you can choose to bind to the object using either **DataConnection** or **TypedDataTable** by selecting "Data connection" or "Database table/row" from the **Database binding type** drop-down box in the Property Window for the **Databases** tab of Fact Explorer.

TypedDataTable

You can assert an ADO.NET **DataTable** object into the engine, but it will be treated as any other .NET object. In most cases you will instead want to assert the rule engine class **TypedDataTable**.

TypedDataTable is a wrapper class that contains an ADO.NET **DataTable**. The constructor simply takes a **DataTable**. Any time a table or table column is used as a rule argument, the expression is evaluated against the individual **TypedDataRow** wrappers, and not against the **TypedDataTable**.

TypedDataRow

This is a typed fact wrapper for an ADO **DataRow** object. Dragging a table or column to a rule argument in the Business Rule Composer results in rules built against the returned **TypedDataRow** wrappers.

Short-Term Facts vs. Long-Term Facts

Facts can be used for different durations of time. A short-term fact is specific to a single execution cycle of the rule engine. A long-term fact is loaded into memory for use over an arbitrary number of execution cycles. The only real distinction between the two is in implementation.

If you use long-term facts, you must configure your policy to know where to find them, and implement a fact retriever object that can fetch the facts from persistent storage and present them to the policy object. For more information, see Creating a Fact Retriever.

There are three ways to supply fact instances to the rule engine:

- The simplest way to submit fact instances to the rule engine is with short-term, application-specific facts that you pass in to the policy object as an array of objects at the beginning of every execution cycle.

- You can write a fact retriever—an object that implements standard methods and typically uses them to supply long-term and slowly changing facts to the rule engine before the policy is executed. The engine caches these facts and uses them over multiple execution cycles.

- You can use your rule actions to assert additional facts into the engine during execution, provided that the rules involved evaluate to **true**.

Rule Engine

This topic describes several components, functionalities, and operations of the rule engine. The rule engine provides the execution context for a rule set. The **RuleEngine** object uses the following plug–in components for implementation:

- **Ruleset executor (inference engine).** Implements the algorithm responsible for rule condition evaluation and action execution. The default ruleset executor is a discrimination network-based forward-chaining inference engine designed to optimize in-memory operation.

- **Ruleset translator.** Takes as input a **RuleSet** object and produces an executable representation of the ruleset. The default in-memory translator creates a compiled discrimination network from the ruleset definition.

- **Ruleset tracking interceptor.** Receives output from the ruleset executor (inference engine) and forwards it to ruleset tracking and monitoring tools.

In This Section

Condition Evaluation and Action Execution

The Business Rules Framework provides a highly efficient inference engine capable of linking rules to .NET objects, XML documents, or database tables.

The Business Rule Engine uses a three-stage algorithm for policy execution. The stages are as follows:

- **Match.** In the match stage, facts are matched against the predicates that use the fact type (object references maintained in the rule engine's working memory) using the predicates defined in the rule conditions. For the sake of efficiency, pattern matching occurs over all the rules in the policy, and conditions that are shared across rules are matched only once. Partial condition matches may be stored in working memory to expedite subsequent pattern-matching operations. The output of the pattern-matching phase consists of updates to the rule engine agenda.

- **Conflict resolution.** In the conflict resolution stage, the rules that are candidates for execution are examined to determine the next set of rule actions to execute based on a predetermined resolution scheme. All candidate rules found during the matching stage are added to the rule engine's agenda.

  The default conflict resolution scheme is based on rule priorities within a policy. The priority is a configurable property of a rule in the Business Rule Composer. The larger the number, the higher the priority; therefore if multiple rules are triggered, the higher-priority actions are executed first.

- **Action.** In the action stage, the actions in the resolved rule are executed. Note that rule actions can assert new facts into the rule engine, which causes the cycle to continue. This is also known as forward chaining. It is important to note that the algorithm never preempts the currently executing rule. All actions for the rule that is currently firing will be executed before the match phase is repeated.

However, other rules on the agenda will not be fired before the match phase begins again. The match phase may cause those rules on the agenda to be removed from the agenda before they ever fire.

The following example shows the three-stage algorithm of match-conflict resolution-action.

Rule 1: Evaluate income

- Declarative representation:

An applicant's credit rating should be obtained only if the applicant's income-to-loan ratio is less than 0.2.

- IF—THEN representation using business objects:

IF Application.Income / Property.Price < 0.2

THEN Assert new CreditRating(Application)

Rule 2: Evaluate credit rating

- Declarative representation:

An applicant should be approved only if the applicant's credit rating is more than 725.

- IF—THEN Representation using business objects:

IF Application.SSN = CreditRating.SSN AND CreditRating.Value > 725

THEN SendApprovalLetter(Application)

The facts are summarized in the following table.

Fact	**Fields**
Application – An XML document representing a home loan application	• Income = $65,000 • SSN = XXX-XX-XXXX
Property – An XML document representing the property being purchased	• Price = $225,000
CreditRating – An XML document containing the loan applicant's credit rating	• Value = 0 – 800 • SSN = XXX-XX-

	XXXX

Initially the rule engine working memory and agenda are empty. After the application adds the Application and Property facts, the rule engine working memory and agenda are updated as follows.

Working memory	Agenda
• Application • Property	Rule 1

Rule 1 is added to the agenda because its condition (Application.Income / Property.Price < 0.2) evaluated to **true** during the match phase. There is no CreditRating fact in working memory, so the condition for Rule 2 was not evaluated. Because the only rule in the agenda is Rule 1, the rule is executed and then disappears from the agenda. The single action defined for Rule 1 results in a new fact (CreditRating document for the applicant) being added to working memory. After the execution of Rule 1 completes, control returns to the match phase. Because the only new object to match is the CreditRating fact, the results of the match phase are as follows.

Working memory	Agenda
• Application • Property • CreditRating	Rule 2

At this point Rule 2 is executed, resulting in the invocation of a function that sends an approval letter to the applicant. After Rule 2 has completed, execution of the forward-chaining algorithm returns to the match phase. Because there are no longer new facts to match and the agenda is empty, forward chaining terminates and policy execution is complete.

Agenda and Priority

To understand how the rule engine evaluates rules and executes actions, you need to understand the concepts of agenda and priority.

Agenda

The agenda is a schedule where the engine queues rules for execution. The agenda exists for an engine instance, and acts on a single policy, not on a series of policies. When a fact is asserted into working memory and the conditions of a given rule are satisfied, the rule is placed on the agenda and executed according to priority. A rule's actions are executed in order from top to bottom, and then the actions of the next rule on the agenda are executed.

The actions belonging to a rule are treated as a block, so that all actions are executed before moving on to the next rule. All actions in a rule block will execute regardless of other actions in the block. For more information about assertion, see Engine Control Functions.

The following example demonstrates how the agenda works.

Rule1

IF

Fact1 == 1

THEN

Action1

Action2

Rule2

IF

Fact1 > 0

THEN

Action3

Action4

We assert the fact Fact1, which has a value of 1, into the engine. Because the conditions of both Rule 1 and Rule 2 are satisfied, both rules are moved to the agenda for execution of their actions.

Working memory	**Agenda**
Fact1 (value=1)	**Rule1** • Action1 • Action2

	Rule2 • Action3 • Action4

Priority

Priority for execution is set on each individual rule, with a default priority of 0 for all rules. The priority can range on either side of 0, with larger numbers having higher priority. Actions are executed in order from highest priority to lowest priority.

The following example shows how priority affects the order of execution for the rules.

Rule1 (priority = 0)

IF

Fact1 == 1

THEN

Discount = 10%

Rule2 (priority = 10)

IF

Fact1 > 0

THEN

Discount = 15%

The conditions for both rules have been met, but Rule2 is executed first because it has higher priority. The final discount is 10%, because it is the result of the action executed for Rule1, as shown in the following table.

Working memory	**Agenda**
Fact1 (value=1)	**Rule2** Discount = 15% **Rule1**

	Discount = 10%

Engine Control Functions

This section explains the behaviors associated with several engine control functions that allow an application or policy to control the facts in the rule engine's working memory. The presence of facts in the working memory drives the conditions that are evaluated and the actions that are executed.

This section examines the **Assert**, **Retract**, **RetractByType**, **Reassert**, and **Update** functions for different facts: .NET objects, **TypedXmlDocument**, **DataConnection**, and **TypedDataTable**.

In This Section

- Assert
- Retract
- RetractByType
- Reassert
- Update

Assert

Assertion is the process of adding object instances into the rule engine's working memory. The engine processes each instance according to the conditions and actions that are written against the type of the instance, using the match-conflict resolution-action phases.

The following paragraphs describe behaviors that result from using the **Assert** function for different fact types.

NET objects

Each object is asserted into the working memory as a separate instance. This means that the instance is analyzed by each predicate that references the object's type (for example, IF Object.Property = 1). It is also made available to rule actions that reference the type, dependent on the results of the rule conditions.

Consider the following example.

In Rule 1, only those instances of A that have a Value of 1 will have their **Status** property updated. In Rule 2, however, if the condition evaluates to **true**, all instances of A will have their status updated. In fact, if there are multiple instances of B, the A instances will be updated each time the condition evaluates to **true** for a B instance.

To assert a .NET object from within a rule, you can add the built-in **Assert** function as a rule action. Note that the rule engine has a **CreateObject** function, but it is not displayed as a separate function in the Business Rule Composer. Its invocation is built by dragging the constructor method of the object you wish to create from the .NET Class view of the Facts Explorer to the action pane. The Business Rule Composer will then translate the constructor method into a **CreateObject** call in the rule definition.

TypedXmlDocument

When a **TypedXmlDocument** is asserted, the Business Rule Engine creates child **TypedXmlDocument** instances based on the selectors defined in the rule.

Selectors and fields are both XPath expressions. You can think of selectors as a way to isolate nodes of an XML document, and fields as identifying specific items within the selector. All the fields inside one selector are grouped together as an object by the engine. When you select a node under the **XML Schemas** tab in the Facts Explorer, the Business Rule Composer automatically fills in the **XPath Selector** property for all nodes, and the **XPath Field** property for any node that does not contain child nodes. Alternatively, you can enter your own XPath expressions for **XPath Selector** and **XPath Field** if necessary.

If the selector matches multiple portions of the XML document, there will be multiple objects of this type asserted into or retracted from the rule engine working memory. Assume you have the following XML.

If you use the selector /root/order (or //order), two objects will be added to the working memory.

1)

2)

Within each selector, the individual fields are referred to by XPaths.

If you use the selector /root/order/item (or (//order/item or //item), four objects, shown in bold text, will be added to the rule engine working memory.

Each object has access to three fields—@name, @quantity, and @cost. Because the object is a reference into the original document, you can refer to parent fields (for example, "../@customer").

You can use multiple selectors within the same document. This enables you to view different parts of the document (for example, if one section is the order and another section contains the shipping address). However, keep in mind that the objects that are created are defined by the XPath string that created them. Using a different XPath expression, even if it resolves to the same node, will result in a unique **TypedXmlDocument**.

The rule engine supports basic .NET scalar types natively, as well as objects for reference types. XML documents are basically text, but based on the type that was specified when the rule was built, the field value may be of any type. Also, because fields are XPath expressions, they may return a nodeset, in which case the first item in the set is used as the value.

Behind the scenes, the rule engine can convert a text field value to any one of the supported types through the **XmlConvert** function. You can specify this by setting the type in the Business Rule Composer. An exception is thrown if a conversion is not possible. Types **bool** and **double** can be retrieved only as their respective type, strings, or objects.

TypedDataTable

When a **TypedDataTable** is asserted, all **DataRows** contained in the **DataTable** are automatically asserted into the engine as **TypedDataRow**s. Any time a table or table column is used as a rule argument, the expression is evaluated against the individual **TypedDataRow**s, and not against the **TypedDataTable**.

For example, suppose that you have the following rule built against a "Customers" table:

IF Northwind.Customers.CustomerID = 001

THEN Northwind.Customers.ContactTitle = "Purchasing Manager"

Suppose that you assert the following **DataTable** with three **DataRow**s into the engine (as a **TypedDataTable**).

CustomerID	ContactTitle
001	Supply Clerk
002	Supply Clerk
003	Supply Clerk

The engine inserts three **TypedDataRow**s: 001, 002, and 003.

Each **TypedDataRow** is evaluated independently against the rule. The first **TypedDataRow** meets the rule condition and the second two fail. The results appear as follows.

CustomerID	ContactTitle
001	*Purchasing Manager*
002	Supply Clerk
003	Supply Clerk

The **DataSetName.DataTableName** is considered to be a unique identifier. Therefore, if a second **TypedDataTable** is asserted with the same **DataSet** name and **DataTable** name, it supersedes the first **TypedDataTable**. All **TypedDataRow**s associated with the first **TypedDataTable** are retracted, and the second **TypedDataTable** is asserted.

DataConnection

As with a **TypedDataTable**, dragging a table or column as a rule argument in the Business Rule Composer results in rules built against the returned **TypedDataRow**s as opposed to the **DataConnection** itself.

Suppose that the following rule is created and a **DataConnection** is asserted that contains a **SqlConnection** to Northwind.Customers:

IF Northwind.Customers.CustomerID = 001

THEN Northwind.Customers.ContactTitle = "Purchasing Manager"

When the engine evaluates the rule used in the **TypedDataTable** section, it dynamically builds a query that looks like:

SELECT *

FROM Northwind.Customers

WHERE CustomerID = 1

Because only one row in the database meets this criterion, only one **TypedDataRow** is created and asserted into the engine for further processing.

Summary

The following table summarizes the assert behavior for the various types, showing the number of resulting instances created in the engine for each asserted entity, as well as the type that is applied to each of those instances to identify them.

Entity	Number of instances asserted	Instance type
.NET object	1 (the object itself)	Fully Qualified .NET Class
TypedXmlDocument	1-N TypedXmlDocument(s): Based on Selector bindings created and document content	DocumentType.Selector
TypedDataTable	1-N TypedDataRow(s): One for each DataRow in the DataTable	DataSetName.DataTableName
TypedDataRow	1 (the TypedDataRow asserted)	DataSetName.DataTableName
DataConnection	1-N (one for each TypedDataRow returned by querying the DataConnection)	DataSetName.DataTableName

Retract

You can use the **Retract** function to remove objects from the rule engine's working memory. The following paragraphs describe the behavior associated with retracting entities of different types from the rule engine's working memory.

NET objects

A .NET object is retracted in a policy by using the **Retract** function. This function is available in the Business Rule Composer as a **Functions** vocabulary item: drag the class (not the assembly or method) into the **Retract** parameter.

Retracting a .NET object removes it from the rule engine's working memory and has the following impact:

- Rules that use the object in a predicate have their actions removed from the agenda (if any exist on the agenda).
- Actions on the agenda that use the objects are removed from the agenda.
- The object is no longer evaluated by the engine.

TypedXmlDocument

You can either retract the original **TypedXmlDocument** that was asserted into the engine or retract one of the child **TypedXmlDocument**s created from a node of the parent **XmlDocument**.

Using the following XML as an example, you can either retract the **TypedXmlDocument** associated with order or one or both of the **TypedXmlDocument**s associated with orderline.

To retract the order object, you would drag the top node for the schema in the XML Schemas fact pane. This node ends in ".xsd" and represents the document root node (not the document element node); it has a "/" selector that refers to the initial **TypedXmlDocument**. When the parent **TypedXmlDocument** is retracted, all **TypedXmlDocument** instances associated with the **TypedXmlDocument** (all **TypedXmlDocument**s created by calling the **Assert** function based on selectors used in the policy) are removed from working memory.

To retract only an individual child **TypedXmlDocument** (that is an orderline), you can drag this node from the XML Schemas pane into the **Retract** function. It is important to note that all **TypedXmlDocument**s are associated with the top-level **TypedXmlDocument** that was originally asserted, and not with the **TypedXmlDocument** that appears above it in the XML tree hierarchy. For example, product is a **TypedXmlDocument** below the orderline object; therefore, it would be associated with the order **TypedXmlDocument**, and not with the orderline **TypedXmlDocument**. In most instances, this distinction is not important. However, if you retract the order object, the orderline and product objects are also retracted. If you retract the orderline object, only that object is retracted, and not the product object.

The engine only works with and tracks object instances (**TypedXmlDocument**s) that it created when the **TypedXmlDocument** was initially asserted. If you create additional nodes—for example, sibling nodes to a node that was selected through a selector in the policy—these nodes will not be evaluated in rules unless **TypedXmlDocument**s are created and asserted for them. Asserting these new, lower-level **TypedXmlDocument**s will cause them to be evaluated in rules, but the top-level **TypedXmlDocument** will not have knowledge of them. When the top-level **TypedXmlDocument** is retracted, the new, independently asserted **TypedXmlDocument**s will not automatically be retracted. As a result, if new nodes are created, it is typically most straightforward to retract and reassert the full **XmlDocument**.

The **TypedXmlDocument** class supports a number of useful methods that can be called within a custom .NET member as part of an action. These include the ability to get the **XmlNode** associated with the **TypedXmlDocument** or the parent **TypedXmlDocument**.

TypedDataTable

You can retract either individual **TypedDataRow**s or the entire **TypedDataTable**. If you retract a table, all the containing rows are retracted from working memory.

To retract the entire **TypedDataTable** you need to use a helper function to access the **Parent** property on **TypedDataRow**, for example:

Retract(MyHelper.GetTypedDataTable(TypedDataRow))

In the preceding action, you would drag the table into **TypedDataRow**. In **GetTypedDataTable** you would return the value of **TypedDataRow.Parent**.

As with **TypedXmlDocument**s, if you assert additional, new **TypedDataRow**s for the same **DataTable** after asserting the **TypedDataTable**, they are treated as individual entities and retracting the **TypedDataTable** will not result in the retraction of these extra **TypedDataRow**s. Only the **TypedDataRow**s contained in the **TypedDataTable** when it was asserted will be retracted.

DataConnection

When a **DataConnection** is retracted, all **TypedDataRow**s retrieved from the database through the query constructed by the **DataConnection** are retracted from working memory. The **DataConnection** itself is also retracted, meaning that no more **TypedDataRow**s will be retrieved through the **DataConnection** (that is, through use of the **DataConnection** in other predicates or actions).

When using a **DataConnection**, any retract operation on an individual **TypedDataRow** puts the engine into an inconsistent state. Therefore, operations are not allowed on individual **TypedDataRow**s associated with a **DataConnection**. If you drag the table (using the **DataConnection** parameter) into the **Retract** function, you will be retracting the **DataConnection**.

RetractByType

The **RetractByType** function retracts all instances of a certain type in the working memory, whereas the **Retract** function retracts only specific items of a certain type. The following paragraphs describe how the **RetractByType** function works with entities of different types.

NET objects

All objects for a given class type are retracted from working memory. You simply drag the class from the .NET Classes fact pane into the **RetractByType** function.

TypedXmlDocument

All instances are retracted. This means that all **TypedXmlDocument**s with the same **DocumentType.Selector** are retracted. You should drag the appropriate node from the XML Schemas fact pane into the **RetractByType** function. Consistent with the **Retract** function, if you perform a **RetractByType** on the document root node, not only will all **TypedXmlDocument**s asserted with that **DocumentType** be retracted, but all child **TypedXmlDocument**s (**XmlNode**s in the tree hierarchy) associated with those parent **TypedXmlDocument**s will also be retracted.

TypedDataTable and DataConnection

Retract and **RetractByType** are equivalent for both **TypedDataTable** and **DataConnection**. Because **DataSetName.DataTableName** is a unique identifier for both types, there is only one instance in the engine at any point in time. As with **Retract**, you drag the table into the **RetractByType** function.

Reassert

To reassert means to call the **Assert** function on an object that is already in the engine's working memory. Reassert is equivalent to issuing a retract command for the object, followed by an assert command.

NET objects

The object is first retracted and any actions on the agenda for rules that use the object (in a predicate or action) are removed. The object is then asserted back into working memory and evaluated as any object that is newly asserted. This means that any rules that use the object in a predicate are re-evaluated and their actions are added to the agenda as appropriate. Any rules that previously evaluated to **true** and only use the object in their actions will have their actions re-added to the agenda.

TypedXmlDocument

When a top level **TypedXmlDocument** (**TXD**) is reasserted, the child **TXD**sthat were created when the top level **TXD** was initially assertedwill have different behaviors depending on the state of the child **TXD**s. In the case of a new child node or a child node that is dirty, meaning at least one of its fields has been changed in the policy by using rule action, assert or reassert action will be performed on the child node. Any existing child node that is not dirty remains in the working memory. The following example is a simplified scenario that describes the behaviors of the child nodes when their parent node is reasserted.

Assume there are three **TXD**s currently in the working memory: **P**, **C**1, **C**2, and **C**3. **P** is the top level **TXD**, the parent node; each child node contains a field **x**.

Next, assume the following operations have been performed as a result of rule action:

- The field (**x**) value for **C**2 is updated.
- **C**3 is deleted by using user code.
- An additional child node, **D**, is added to **P** by using user code.

Note A node will not be marked dirty by the Business Rule Engine from operations, of which the engine is not aware. For example, adding, deleting, or modifying a node programmatically in an external application.

The new representation of the objects in working memory is as follows.

Now, reassert **P**. The following points summarize the behaviors of the child nodes:

- Node **C**2 is reasserted, because it has become dirty after its field being updated.
- Node **C**3 is retracted from the working memory.
- Node **D** is asserted into the working memory.
- Node **C**1remains unchanged in the working memory, because it was not updated before **P** was reasserted.

TypedDataTable

If **Retract** is issued on a **TypedDataRow**, that row is retracted and then asserted into working memory. If **Retract** is issued on the **TypedDataTable**, all associated **TypedDataRow**s are retracted and then asserted.

DataConnection

All **TypedDataRow**s retrieved through the **DataConnection** are retracted. All predicates that use the **DataConnection** are then re-evaluated, causing the **DataConnection** to be requeried to create the relevant **TypedDataRow**s.

Update

You can use the **Update** function to improve engine performance and prevent endless loop scenarios. In a typical scenario, an object is updated by a rule and then needs to be reasserted into the engine to be re-evaluated, based on the new data and state.

Take the two rules that follow as an example. Suppose that objects **ItemA** and **ItemB** already exist in working memory. Rule 1 evaluates the **Id** property on **ItemA**, sets the **Id** property on **ItemB**, and then reasserts **ItemB** after the change. When **ItemB** is reasserted, it is treated as a new object and the engine re-evaluates all rules that use object **ItemB** in the predicates or actions. This ensures that Rule 2 is re-evaluated against the new value of **ItemB.Id**, as set in Rule 1. Rule 2 may have failed the first time it was evaluated, but evaluates to **true** the second time it is evaluated.

Rule 1

IF ItemA.Id == 1

THEN ItemB.Id = 2

Assert(ItemB)

Rule 2

IF ItemB.Id == 2

THEN ItemB.Value = 100

This ability to reassert objects into working memory allows the user explicit control over the behavior in forward-chaining scenarios. A side effect of the reassertion in this example, however, is that Rule 1 is also re-evaluated. Because **ItemA.Id** was not changed, Rule 1 again evaluates to **true** and the **Assert(ItemB)** action fires again. As a result, the rule creates an endless loop situation.

You need to be able to reassert objects without creating endless loops, and the **Update** function provides this capability. Like a reassert, the **Update** function performs **Retract** and **Assert** of the associated object instances, which have been changed from rule actions, but there are two key differences:

- Actions on the agenda for rules where the instance type is only used in the actions (not the predicates) will remain on the agenda.
- Rules that only use the instance type in the actions will not be re-evaluated.

Therefore, rules that use the instance types in either the predicates only or both the predicates and actions will be re-evaluated and their actions added to the agenda as appropriate.

Changing the preceding example to use the **Update** function ensures that only Rule 2 is re-evaluated. Because **ItemB** is only used in the actions of Rule 1, Rule 1 is not reevaluated, eliminating the looping scenario.

Rule 1

IF ItemA.Id == 1

THEN ItemB.Id = 2

Update(ItemB)

Rule 2

IF ItemB.Id == 2

THEN ItemB.Value = 100

However, it is still possible to create looping scenarios. For example, consider the following rule.

Rule 1

IF ItemA.Id == 1

THEN ItemA.Value = 20

Update(ItemA)

Because **ItemA** is used in the predicate, it is re-evaluated when **Update** is called on **ItemA**. If the value of **ItemA.Id** is not changed elsewhere, Rule 1 continues to evaluate to **true**, causing **Update** to once again be called on A. The rule designer must ensure that looping scenarios such as this are not created.

The appropriate approach for this will differ based on the nature of the rules. The following is a simple mechanism to solve the problem in the preceding example.

Rule 1

IF ItemA.Id == 1 and ItemA.Value != 20

THEN ItemA.Value = 20

Update(ItemA)

Adding the check on **ItemA.Value** prevents Rule 1 from evaluating to **true** again after the actions of Rule 1 are executed the first time.

NET objects

The .NET object is retracted and asserted with the behavior documented earlier. The **Update** function may be used in the Business Rule Composer with a reference to the class, as with the **Assert**, **Retract**, or **RetractByType** functions.

TypedXmlDocument

All object instances that were created from the **TypedXmlDocument** and have been changed are retracted and asserted. Passing a **TypedXmlDocument** to the **Update** function is performed in the same manner as with the other functions.

As with reassert, the update mechanism may be carried out on child nodes. Consider a simple example in which the **Update** function is being performed on a node **P**, the parent of node **C**. If node **C** is dirty, which means that at least one of its fields has been changed as a result of rule actions, node **C** will be updated in this case. If node **C** is not dirty, it remains unchanged.

TypedDataTable

If **Update** is called on a **TypedDataTable**, **Update** is called by the engine on all associated **TypedDataRow**s. **Update** may also be called on individual **TypedDataRow**s.

DataConnection

Update of a **DataConnection** is not supported. Use **Assert** instead.

Data Access in Business Rule Engine

The rule engine supports only .NET objects natively. To handle data from a database, you can use the ADO.NET objects directly, but the engine provides some helper classes to simplify the use of database data from rules. The rule engine extends its support by exposing three database-related types: **TypedDataRow**, **TypedDataTable**, and **DataConnection**. This section describes these helper classes, gives recommendations about when to use each type, and discusses some performance implications when using them.

The helper classes are as follows:

- **TypedDataRow.** Constructed by using a reference to an ADO.NET **DataRow** instance. The **TypedDataRow** is an obvious choice for rules that only deal with data from one or a small number of rows from a particular table.

- **TypedDataTable.** Literally a collection of **TypedDataRow** objects. Each row in the database table will be wrapped as a **TypedDataRow** and asserted into the working memory by the rule engine.

  A **TypedDataTable** requires an in-memory ADO.NET **DataTable**, which can be a performance overhead if this particular **DataTable** contains a very large number of rows. If a small number of rows in the database table are relevant and you can determine these rows prior to calling the rules, use a **DataTable**, otherwise use **TypedDataRow**.The assumption is that a high number of rows in the DataTable are relevant to the rules.

- **DataConnection.** Represents a table in a database accessed through a database connection. The difference between **DataConnection** and **TypedDataTable** is that in addition to the dataset name and table name, **DataConnection** requires a usable database connection and optionally a database transaction context.

Some or all predicates used in rules with the **DataConnection** will become part of query constraints against the database connection. Only rows that satisfy the query constraints will be retrieved from the database and used by the engine. This mechanism provides better performance and consumes less memory than holding a very large **DataTable** in memory.

In This Section

- Considerations When Using DataConnection
- Using DataConnection and TypedDataTable

Considerations When Using DataConnection

The following considerations should be kept in mind when using DataConnection with the Rule Engine.

Use primary keys

When there is a primary key, the equality of two rows is determined by whether the rows have the same primary key, rather than by object comparison. If the rows are determined to be the same, only one copy is retained in memory, and the other is released. This results in less memory consumption.

When a **DataConnection** is asserted into the rule engine for the first time, the engine always tries to locate its primary key information from its schema. If a primary key exists, primary key information is then retrieved and used in all subsequent evaluations.

Note A primary key is mandatory if changes need to be made to the database.

Provide a running transaction to the DataConnection whenever possible

Without a transaction, each query and update on the **DataConnection** will initiate its own local transaction, and different queries might return different results in different parts of rule evaluations. Users may experience inconsistent behavior if there are changes in the underlying database table.

Although you can use a **DataConnection** without providing a transaction when the table does not change over time, it is recommended that a transaction be used even when the **DataConnection** is only being used for read operations.

However, a transaction should always be used when updating data.

Number of queries may grow linearly

As queries against the **DataConnection** are parameterized by other joined objects, the number of queries executed against the **DataConnection** corresponds directly to the number of joining objects reaching the **DataConnection**. Therefore, if the number of joining objects reaching the **DataConnection** object grows linearly, the number of queries against the **DataConnection** will grow linearly as well. Currently, there is no optimization in place to reduce the number of queries.

An example of this is the rule:

A represents an **ObjectBinding**, DC represents a **DataConnection**, and x and y represent attributes of A and DC.

For every instance of A that passes the test (x == 7), a query is generated by using the **DataConnection**. If there are many matching instances, the same number of queries results.

Use OR conditions with caution

If the rule uses only conjunctive (**AND**) conditions, tests and queries will be executed as early as possible, so instances of objects passing through will be reduced. As a result, the number of queries against the subsequent **DataConnection** will be reduced proportionally. If disjunctive (**OR**) conditions and a **DataConnection** are used together in a rule, all condition evaluations will be pushed to the final query. If more than one **DataConnection** is used in a rule, all queries except the last one will effectively become a Select-ALL query statement.

In general, it is better to split any rule with an OR condition into two or more discrete rules, because the use of OR conditions will decrease performance compared to the definition of more atomic rules. This is true whether DataConnections are used or not.

You may also consider using separate rules that consist only of conjunctive conditions instead of one rule with **OR** conditions. With **OR** conditions, the number of queries grows at the speed of multiplication of instances of all joining objects. This is shown in the following example.

In this example, A represents an **ObjectBinding**; DC represents a **DataConnection**, and x, y, and z represent attributes of A and DC. If A has 100 instances, and x is 1 in the first object, 2 in the second object, through 100 in the 100th object, 100 queries have to run against the **DataConnection**.

It is better to rewrite the preceding rule by splitting it in to two rules.

SQL does not support some predicates and functions

Some predicates and functions that the rule engine supports are not supported by SQL. If these unsupported predicates and functions are used in the rule conditions, it cannot be incorporated into the query. The following terms cannot be optimized as an SQL query:

- Some of the engine built-in functions have no equivalent in an SQL query. These are **Power**, **FindFirst**, and **FindAll**. Using a **DataConnection** column as one or more of their arguments cannot be optimized as part of a query; for example, Power(2, dc.Column1).

- Built-in predicate Match that uses a **DataConnection** column as its regular expression argument (the first argument). For example, Match("abc*", dc1.Column2) is valid, but Match(dc1.Column1, dc1.Column2) cannot be translated.

- Using a user function that has a **DataConnection** column as one of its parameters. For example, c1.M(dc.Column1) cannot be optimized because the user function cannot execute on the database server, but must be executed by the Business Rule Engine.

- User functions that represent set operations on the **DataConnection**; for example, dc.Column1(5).

- Functions that use an **ObjectReference** to the **DataConnection**; for example, ObjecRef(dc).

- If one or more of a function's arguments is untranslatable, the function call becomes untranslatable; for example, Add(c1.M(dc.Column1), 5).

Using DataConnection and TypedDataTable

In many scenarios, using **DataConnection** provides better performance and consumes less memory than using **TypedDataTable**. However, **TypedDataTable** may be required in some cases because of certain restrictions on using **DataConnection**. In some other cases, using **TypedDataTable** may yield better performance than using **DataConnection**. This topic describes the criteria and factors that you should consider for choosing the right approach.

When to use TypedDataTable instead of DataConnection

The following shows when to use TypedDataTable instead of DataConnection:

- Data changes need to be made but the table does not have a primary key. To make data changes by using **DataConnection**, a primary key is required. Therefore, if there is no primary key, **TypedDataTable** is the only viable approach.

- Selectivity is high, which means that a large percentage of rows in the table will pass the tests specified as rule conditions. In this case, **DataConnection** does not provide much benefit and it may perform worse than **TypedDataTable**.

- Table is small, typically, a table that contains fewer than 500 rows. Note that this number could be larger or smaller depending on the rule shape and the memory available to the rule engine.

- Rule-chaining behavior is expected in a rule set. Calling the **Update** function on a **DataConnection** is not supported, but you could invoke **DataConnection.Update** in a rule using a helper method. When rule chaining is required, **TypedDataTable** is a better choice.

- One or more columns in the table hold a very large amount of data that is not required by the rules. An example is an image database, where the columns hold the image (large amount of data), name, date, and so on. If the image is not required, it may be better to select only the columns needed by the rules. For example, issuing a query such as "SELECT Name, Date from TABLE" can be more efficient than using **DataConnection**.

- If many rules need or update the same database row, using a **TypedDataTable**, the row is shared between all rules, and if the condition is the same (for example, Table.Column == 5), the condition evaluation can be optimized. With a **DataConnection**, in general, a query is generated for each rule that uses the **DataConnection**. Although the rows are reused (if the table has a primary key), multiple queries could be generated to get the same data each time.

When to use DataConnection instead of TypedDataTable

The following shows when to use DataConnection instead of TypedDataTable:

- Table contains a large number of rows, but selectivity is low—only a small percentage of rows will satisfy the rule conditions.

- Only one database table is large; all other objects used in the rule have a small number of instances. In the worst case, the number of queries executed against the database is equal to the product of all the other instances used in the rule.

- Rules consist exclusively of conjunctive conditions and objects other than database table are used in these rules.

Rule Action Side Effects

If the execution of an action affects the state of an object or a term used in conditions, that action is considered to have a side effect on the object.

Assume we have the following rules.

Rule 1

IF OrderForm.ItemCount > 100

THEN OrderForm.Status = "important"

Rule 2

IF OrderList.IsFromMember = true

THEN OrderForm.UpdateStatus("important")

In this case, **OrderForm.UpdateStatus** is said to have a side effect on **OrderForm.Status**. This does not mean that **OrderForm.UpdateStatus** has side effects; instead, **OrderForm.Status** will potentially be affected by one or more actions.

By default, the **SideEffects** property for .NET class members is **true**, which prevents the rule engine from caching the member with side effects. In our example, the rule engine does not cache **OrderForm.Status** in the working memory; instead it gets the most up-to-date value of **OrderForm.Status** every time Rule 1 is evaluated. If the **SideEffects** property is set to **false**, the rule engine caches the value first time it evaluates **OrderForm.Status**, but for later evaluations (in forward-chaining scenarios), it uses the cached value.

Currently the Business Rule Composer does not provide a way for users to modify **SideEffects**, however, you can only set the **SideEffects** property programmatically through the Business Rules Framework. You set do this at binding, using **ClassMemberBinding** class to specify object methods, properties, and fields used in rule conditions and actions. **ClassMemberBinding** has a property, **SideEffects**, which contains a Boolean value indicating whether accessing the member changes its value.

Support for Class Inheritance in Business Rule Engine

One of the key features of Object Oriented Programming (OOP) languages is inheritance. Inheritance is the ability to use all of the functionality of an existing class, and extend those capabilities without rewriting the original class.

The Business Rules Framework supports two types of class inheritance: implementation and interface. Implementation inheritance refers to the ability to use a base class's properties and methods with no additional coding. Interface inheritance refers to the ability to use just the names of the properties and methods, but the child class must provide the implementation.

The rules can be written in terms of a common base class, but the objects asserted into the engine can be from derived classes. In the following example,

RegularEmployee and **ContractEmployee** are derived classes of the base class **Employee**.

Assume you have the following rule.

Rule 1

At run time, if the user asserts two objects, one an instance of **ContractEmployee** and the other an instance of **RegularEmployee**, both the object instances are evaluated against the rule described earlier.

The user can also assert derived class objects in actions by using an **Assert**. This causes the rules that contain the derived object and its base type in their conditions to be re-evaluated, as shown in the following example.

Rule 2

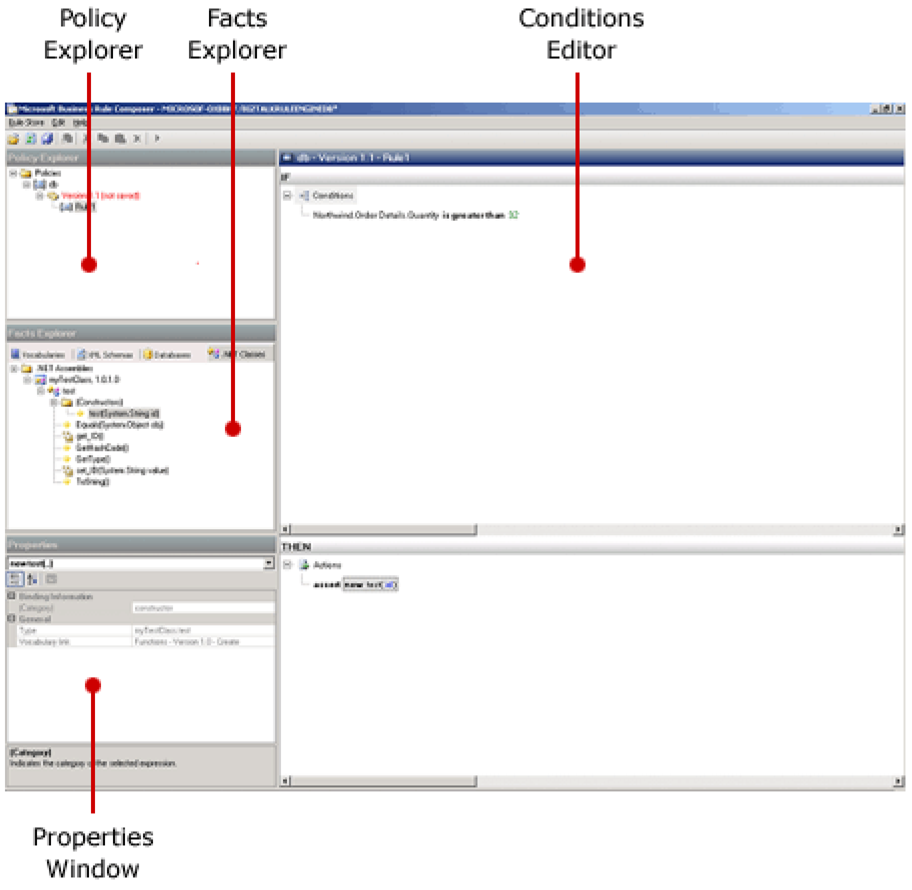
All rules containing the **Employee** type or the **ContractEmployee** type in their conditions get re-evaluated after the assertion. The type of inheritance in this example is implementation. Even though only the derived class is asserted, the base class also gets asserted if rules are written using methods in the base instead of the derived class.

Using the Business Rule Composer

In Microsoft® BizTalk® Server, the Business Rule Composer is a graphic tool used for authoring, versioning, and deploying policies and vocabularies. This section provides information about different windows of the Business Rule Composer.

The following figure shows the Business Rule Composer, and describes each window.

Business Rule Composer



In This Section

- Using Policy Explorer
- Using Facts Explorer
- Using Rule Editor

Using Policy Explorer

You can use the Policy Explorer to manage policies and rules in the rule store. You can create, modify, and delete policies and rules, and you can test, publish, deploy, and undeploy policies.

The following table describes the commands within the Policy Explorer that can be used in the process of developing new policies and rules.

Use this	To do this
Add New Policy	Create a new policy (rule set).
Add New Version	Create a new empty version of the selected policy. You can copy rules from other versions and paste them into the new version.
Copy (Policy Version)	Copy the selected policy version to the Clipboard.
Copy (Rule)	Copy the selected rule to the Clipboard.
Cut (Rule)	Copy the selected rule to the Clipboard and delete it.
Paste (Policy Version)	Paste a policy version and its contents into a selected policy.
Paste (Rule)	Paste a rule into the selected policy version.
Delete (Policy Version)	Delete the selected policy version.
Delete (Rule)	Delete the selected rule.
Delete	Delete the selected policy and all its versions.
Add New Rule	Create a new rule in the selected policy version.
Save	Save any changes made to the selected version and its rules.
Publish	Publish the selected policy version. Note that you cannot directly modify a published version. You can copy and paste it to a new version on the policy.
Reload	Reload the selected policy version and its rules, with the option to discard any current changes made in that version and restore the contents from the rule store.
Deploy	Deploy a published policy version. You must deploy a policy version to make it available to rule-based applications.
Undeploy	Undeploy a policy version. After a version is undeployed, it is no longer available to rule-based applications.
Test Policy	Test the selected policy version before deployment.

Properties window

The following table describes the properties for a policy version.

Property	Value
Name	Name of the policy. You can only change this value by changing the **Name** property of the policy (not policy version).
Fact Retriever	Information about the fact retriever for the policy version.
Maximum Execution Loop Depth	Maximum depth of the forward-chaining algorithm before an execution loop exception is thrown. The default loop count is 65536.
Translation Duration	Maximum amount of time to translate the rules before a translation time-out exception is thrown. The default is 60000 milliseconds.
Translator	Information about the translator used to translate the rules.
Current Version	The version of policy currently selected in the Policy Explorer.
Version Description	The description of the current version.

The following table describes the properties for a rule.

Property	Value
Active	Indicates whether the rule is enabled or disabled
Name	Name of the rule.
Priority	Priority of the rule within the policy. The higher the index, the higher the rule priority. Actions of a higher priority rule will fire first. The default is 0 and represents the middle priority.

Using Facts Explorer

The Facts Explorer contains four tabs: **Vocabularies**, **XML Schemas**, **Databases**, and **.NET Classes**.

Vocabularies tab

Use the **Vocabularies** tab in the Facts Explorer to manage vocabulary versions and definitions.

Use the shortcut menu to access the following options.

Use this	To do this
Add New Vocabulary	Create a new vocabulary.
Add New Version	Create a new empty version of the selected vocabulary. You can copy definitions from other versions and paste them into the new version.
Paste Vocabulary Version	Paste the contents of a previously copied vocabulary version as a new version in the selected vocabulary.
Delete	Delete the selected vocabulary and all its versions.
Add New Definition	Launch the Vocabulary Definition Wizard to create a new definition in the selected vocabulary version.
Save	Save the changes made to the selected version and its definitions.
Publish	Publish the selected vocabulary version. Note that you cannot directly modify a published version. You can copy and paste it to a new version of the vocabulary.
Reload	Reload the selected vocabulary version and its definitions, with the option to discard any current changes made in that version and restore the contents from the rule store.
Modify	Launch the Vocabulary Definition Wizard to change the selected definition. Note that you cannot modify a definition that is part of a published vocabulary version.
Go to source fact	For the selected definition, go to the corresponding source fact in a .NET assembly, XML schema, or database table.

XML Schemas tab

Browse through XSD schemas for the definitions of XML elements and attributes. You can drag items onto unpublished vocabulary versions on the **Vocabulary** tab to create definitions, or you can drag items to the Conditions Editor or Actions Editor to define predicates, actions, and arguments.

Use this	To do this
Select Root Node	Select a root node to load from an XML schema that contains multiple root notes.

Databases tab

Browse through database servers for databases and table definitions. You can drag items onto unpublished vocabulary versions on the **Vocabulary** tab to create definitions, or you can drag items to the Conditions Editor or Actions Editor to define predicates, actions, and arguments.

NET Classes tab

Browse through .NET assemblies for public .NET class definitions. You can drag items onto unpublished vocabulary versions on the **Vocabulary** tab to create definitions, or you can drag items to the Conditions Editor or Actions Editor to define predicates, actions, and arguments.

Use this	To do this
Browse	Load a .NET assembly from the global assembly cache
Remove	Remove a .NET assembly

Using Rule Editor

Use the Rule Editor to view and edit conditions in the Conditions Editor and actions in the Actions Editor for the selected rule.

Conditions Editor

Use the Conditions Editor (part of the Rule Editor) to view and edit conditions for firing rules. You can add built-in predicates by using the shortcut menu, drag items from the Facts Explorer to define arguments and predicates, and enter argument values inline by clicking an argument link.

Use the shortcut menu to access the following options.

Use this	To do this
Add logical AND	Add an operator to combine two or more predicates to form a logical **AND** expression.
Add logical OR	Add an operator to combine two or more predicates to form a logical **OR** expression.
Add logical NOT	Add the operator **NOT** to negate a logical expression or predicate.
Predicates	Add a predicate expression based on one of the built-in predicates provided by the Rule object model, such as the **Is Equal To** operator.
Delete logical operator	Delete the selected logical operator (**AND**, **OR**, or **NOT**).
Delete predicate	Delete the selected predicate.
Move Up	Move the predicate up one position or a level.
Move Down	Move the predicate down one position or a level.
Go to vocabulary	Locate the vocabulary definition in the Facts Explorer that corresponds to the selected predicate or argument.
Go to source fact	Locate the XML element, database column, or .NET method in the Facts Explorer that corresponds to the selected predicate or argument.
Reset argument	Delete the selected argument (and any nested arguments), and restore the initial definition.
Set to null	Replace the selected argument with a null constant definition.
Set to empty string	Replace the selected argument with an empty string value.

Actions Editor

Use the Actions Editor (part of the Rule Editor) to view and edit actions to execute when a rule is fired. You can add built-in actions by using the shortcut menu, drag items from the Facts Explorer to define actions and arguments, and enter argument values inline by clicking an argument link.

Use this	To do this
Delete action	Delete the selected action.
Go to vocabulary	Locate the vocabulary definition in the Facts Explorer that corresponds to the selected action or argument.
Go to source fact	Locate the XML element, database column, or .NET method in the Facts Explorer that corresponds to the selected action or argument.
Move Up	Move the action up one position or a level.
Move Down	Move the action down one position or a level.
Reset argument	Delete the selected argument (and any nested arguments), and restore the initial definition.
Set to null	Replace the selected argument with a null constant definition.
Set to empty string	Replace the selected argument with an empty string value.
Functions	Add an argument based on one of the built-in functions provided by the Rule object model, such as the **Add** operator.

Output window

Use the Output window to view results of test execution for a selected policy version.

Use the shortcut menu to access the following options.

Use this	To do this
Clear All	Clear all text from the Output window.
Copy	Copy the selected text in the Output window to the Clipboard.
Select All	Select all the text contained in the Output window.
Save to File	Save the text contained in the Output window to a specified file.

Creating Business Rules

This section provides task-specific information about using the Microsoft® BizTalk® Server Business Rule Composer to create business rules.

In This Section

How to Start the Business Rule Composer and Loading a Policy Selecting Facts How to Create Policies and Rules How to Modify Rules How to Maintain Policy Versions How to Configure a Fact Retriever for a Policy How to Develop Vocabularies How to Implement Negation as Failure How to Handle Null and DBNull l Tracking and Monitoring Policies How to Deploy and Undeploy Policies and Vocabularies Calling Business Rules in Orchestrations How to Import and Export Policies and Vocabularies

How to Start the Business Rule Composer and Loading a Policy

This section describes how to start the Business Rule Composer and load a policy.

To start the Business Rule Composer

1. On the **Start** menu, point to **All Programs**, point to **Microsoft BizTalk Server 2006**, and then click **Business Rule Composer**.

To load a policy

1. In the Business Rule Composer, on the **Rule Store** menu, click **Load**.
2. In the **Rule Store** dialog box, in the **SQL Server** drop-down list, select the SQL Server™ computer to which you want to connect.
3. In the **Database** drop-down list, select the database that contains the rule store you want to open.

   Note The default database for the rule store installed by BizTalk Server 2006 is **BizTalkRuleEngineDb**.

Selecting Facts

You can specify data source references that you can save in your rule store for subsequent use as facts in your rules and vocabularies. You can specify XSD schemas and their XML document elements and attributes, .NET assemblies and their classes and class members, or databases and their tables and table columns.

⊟**In This Section**

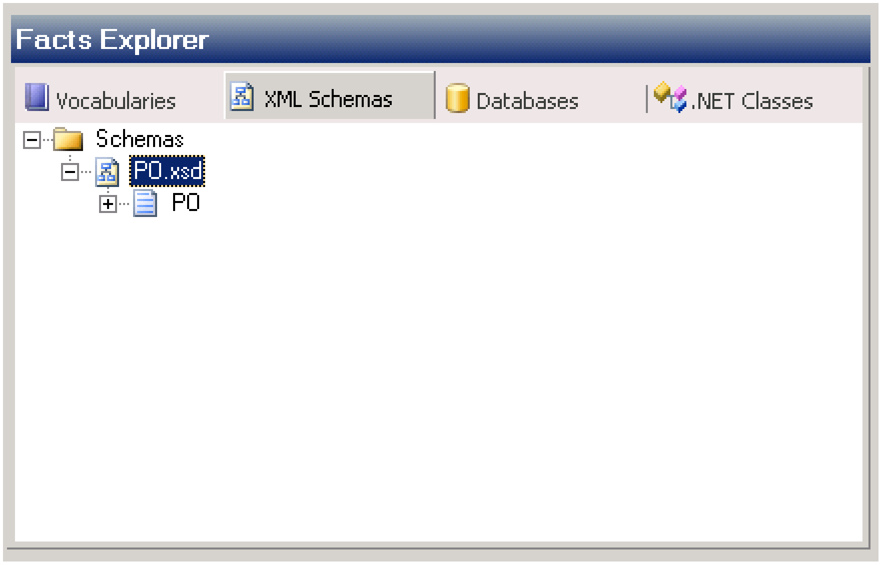
- How to Use an XML Schema as a Data Source
- How to Use a Database as a Data Source
- How to Use a .NET Assembly as a Data Source

How to Use an XML Schema as a Data Source

To specify an XML schema as a data source

1. In the Facts and Definitions window, click the **XML Schemas** tab.
2. Right-click the **Schemas** folder, and then click **Browse**.
3. In the **Schema Files** dialog box, select an XSD file, and then click **Open**.

Browsing an XML Schema



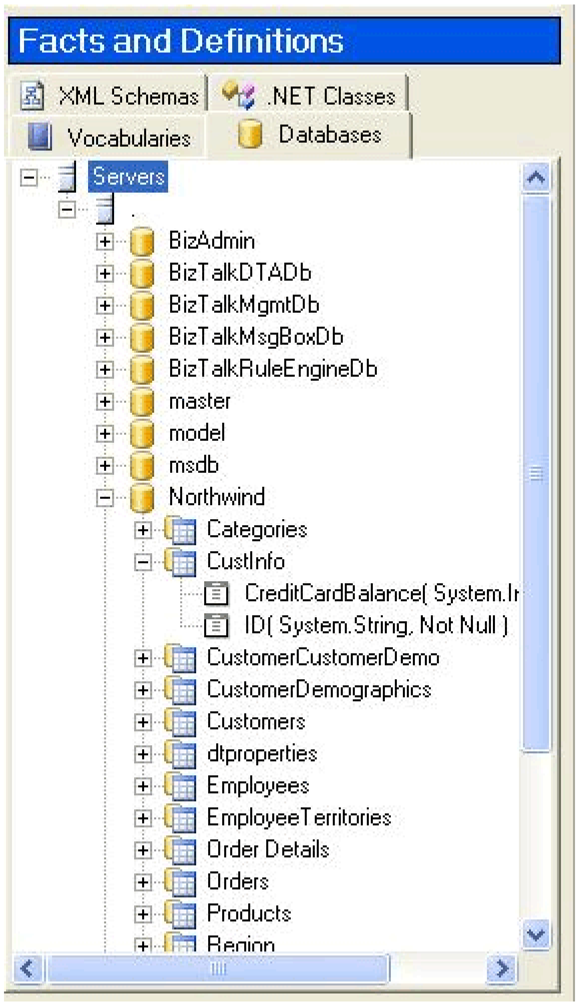
How to Use a Database as a Data Source

You can specify a database as a data source. You can subsequently select a table or table column from the database, and drag it onto a vocabulary definition or rule to use as a fact.

To specify a SQL database as a data source

1. In the Facts and Definitions window, click the **Databases** tab.
2. Right-click the **Servers** node, and then click **Browse**.
3. In the drop-down list, select an available database server.
4. Select an authentication type. If you select SQL authentication, enter a logon name and password. When you have entered your authentication information, click **OK**.

Browsing a database



How to Use a .NET Assembly as a Data Source

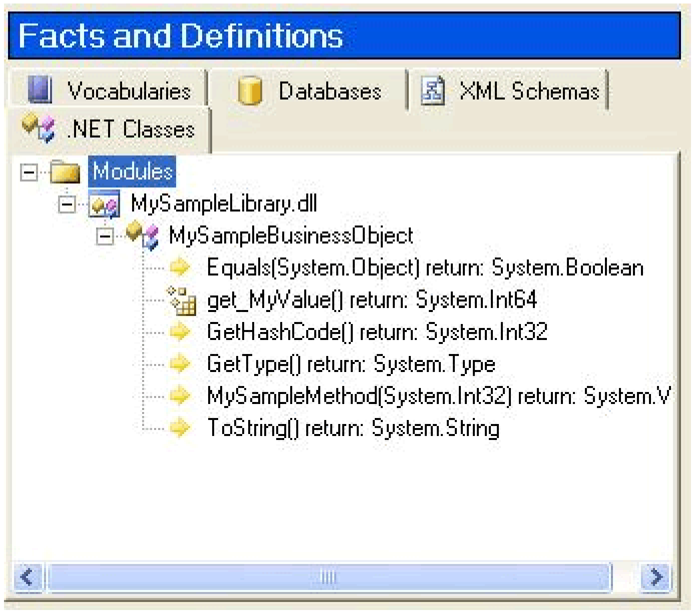
You can specify a .NET assembly as a data source. You can subsequently select a class or class member from the assembly, and drag it onto a vocabulary definition or rule.

⊟To specify a .NET assembly as a data source

1. In the Facts and Definitions window, click the **.NET Classes** tab.
2. Right-click the **Modules** node.
3. From the available assemblies, select a .NET assembly.

Note The assemblies have to be in the global assembly cache (GAC)

Browsing a .NET assembly



Modules
MySampleLibrary.dll
MySampleBusinessObject
Equals(System.Object) return: System.Boolean
get_MyValue() return: System.Int64
GetHashCode() return: System.Int32
GetType() return: System.Type
MySampleMethod(System.Int32) return: System.V
ToString() return: System.String

How to Create Policies and Rules

You can create rules with conditions that are logical groupings of logical operators (**AND**, **OR**, and **NOT**) applied to predicates (built-in or user-defined functions or operators) that take arguments (built-in or user-defined fact references). You can also right-click on **Conditions** or logical operators and select a logical operator or built-in predicate from the context menu.

You can define actions (built-in or user-defined functions) to be executed if the rule condition evaluates to **true**.

Note If you include more than one predicate in a rule, all predicates must appear as arguments to a logical operator. (The top level can be a single .Net member, db column, or XML field/attribute that is of boolean type.)

This topic contains procedures for the following tasks:

- To create a policy
- To add a rule to a policy
- To add a logical operator to a rule condition
- To add a built-in predicate to a rule condition or logical operator
- To add a built-in action to a rule

- To add an argument to a condition or action

To create a policy

1. In the Policy Explorer pane, right-click **Policies**, and then click **Add New Policy**.

   A new folder, **Policy1**, is created under **Policies**. By default, version 1 of a new policy is created for you.

2. Click **Policy1**.

3. In the Name property pane, type a name.

To add a rule to a policy version

1. In the Policy Explorer pane, expand [**your policy**], right-click **Version 1.0 (not saved)**, and then select **Add New Rule**.

To add a logical operator to a rule condition

1. In the Rule Definition window, right-click **Conditions**, and then click one of **Add Logical AND**, **Add Logical OR**, or **Add Logical NOT**.

To add a built-in predicate to a rule condition or logical operator

1. In the Facts and Definitions window, click the **Vocabularies** tab, and then click the **Predicates** folder.

2. Expand a published version of a predicate vocabulary, and click the predicate you want.

3. Drag the predicate onto the logical operator, or onto **Conditions** if your rule will contain only one predicate.

To add a built-in action to a rule

1. In the Facts and Definitions window, click the **Vocabularies** tab, and then click the **Functions** folder.

2. Expand a published version of the function vocabulary, and click the function you want.

3. Drag the function onto **Actions**. You can also right-click on **Actions**, and select a built-in action from the context menu.

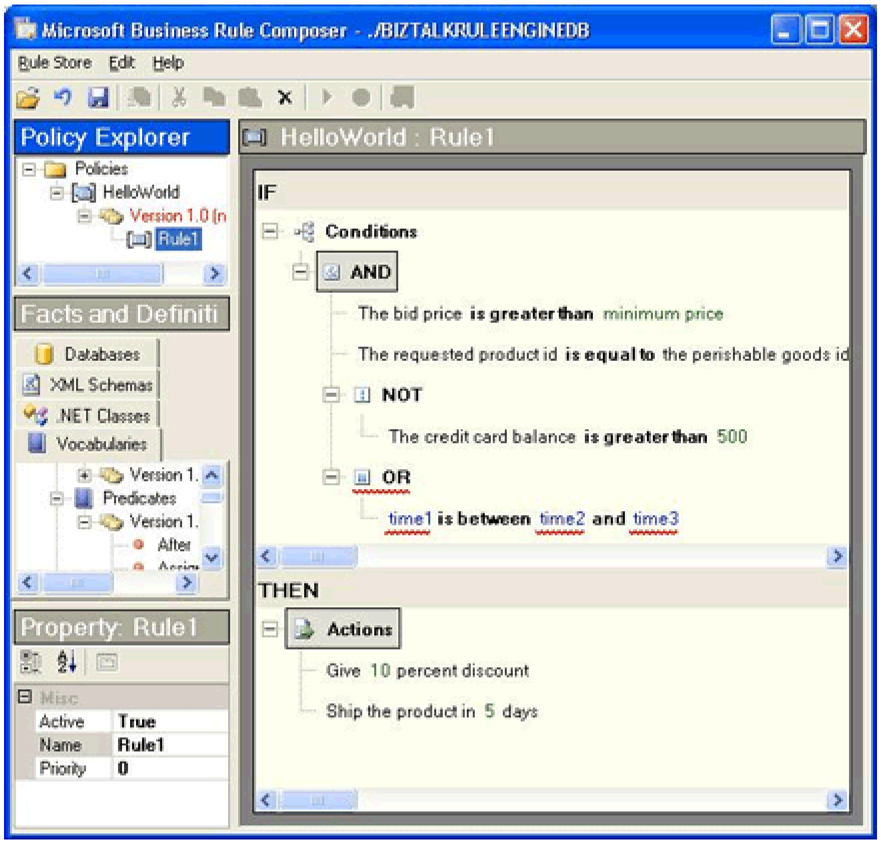
To add an argument to a condition or action

1. In the Facts and Definitions window, click the **Vocabularies** tab, and then click a vocabulary folder.

2. Expand a published version of the vocabulary, and click the term you want. The term must be of a type expected by the predicate or function.

3. Drag the term onto a predicate argument in a condition or a function argument in an action.

Sample rule



The bid price **is greater than** minimum price
The requested product id **is equal to** the perishable goods id
The credit card balance **is greater than** 500
time1 **is between** time2 **and** time3

Give 10 percent discount
Ship the product in 5 days

Active	True
Name	Rule1
Priority	0

How to Modify Rules

The ability to change rules is an important part of the business rules paradigm. You can modify rules within a policy in two ways: either by creating a new version of the policy, or by directly modifying an unpublished version of the policy.

You can modify rules individually, add new rules, or delete existing rules. You can delete predicates and logical operators from a rule condition, delete actions, move actions up and down in the display, or move predicates and logical operators within a condition. Keep in mind, however, the order in which the predicates and logical operators are displayed does not determine the order of evaluation.

You can set a rule to be inactive, so that the rule is not executed when the policy is executed, or you can reactivate a rule that has been deactivated.

You can set the priority on a rule so that its actions will be executed before or after the actions of any rule of different priority.

Caution If you need to stop your SQL Server computer, be sure to save any unsaved vocabulary versions or vocabulary definitions, and close the Business Rule Composer so that no changes are lost.

This topic contains procedures for the following tasks:

- To change an argument in a condition or action
- To move a predicate within a condition
- To move a logical operator within a condition
- To change the order of actions within a rule
- To delete a predicate, logical operator, or action
- To activate or deactivate a rule
- To set a priority on a rule

To change an argument in a condition or action

1. In the Facts and Definitions window, click the appropriate tab, and navigate to the term you want to use as an argument. The term must be of a type expected by the predicate or function.
2. Click the term and drag it onto a predicate argument in a condition or a function argument in an action.

To move a predicate within a condition

1. Click the predicate, and drag it to another logical operator.

To move a logical operator within a condition

1. Click the logical operator, and drag it onto another logical operator or onto **Conditions**.

To change the order of actions within a rule

1. Click the action and then click **Move action up** or **Move action down**.

To delete a predicate, logical operator, or action

1. Click the predicate, logical operator, or action, and then click **Delete**.

To activate or deactivate a rule

1. Click the rule, and then in the Properties window, set **Active** to either **True** or **False**.

To set a priority on a rule

1. Click the rule, and then in the Properties window, set **Priority** to an integer value.

How to Maintain Policy Versions

After you add rules to a version of your policy, you can save the version to the rule store for further development, or you can publish it to create a well-defined, immutable set of rules that can be deployed for use in a rule-based application.

This topic contains procedures for the following tasks:

- To create an empty new version of a policy
- To save a policy version
- To publish a policy version
- To create an updated version of a policy
- To update a policy to use an updated assembly

To create an empty new version of a policy

1. Right-click the policy folder, and then click **Add New Version**.⊟To save a policy version

1. Right-click the policy version, and then click **Save**.

To publish a policy version

1. Right-click the policy version, and then click **Publish**.

To create an updated version of a policy

1. Right-click an existing policy version, and then click **Copy**.
2. Right-click the policy folder, and then click **Paste**.

   A new version is created, with the same elements as the copied version.

To update a policy to use an updated assembly

1. Click **Start**, point to **Administrative Tools**, point to **.NET Framework Configuration**, and then click **Configured Assemblies**.
2. Go to properties for the target assembly and click the **Binding Policy** tab.
3. In the global assembly cache, change the version number to the newer version.

How to Configure a Fact Retriever for a Policy

You can store facts that do not change frequently, and then before the first execution cycle of your host application, you can retrieve these facts from storage, present them once to the rule engine for caching, and reuse them over multiple execution cycles.

There are two ways to associate a fact retriever with a policy. You can do this by using the Business Rule Composer or programmatically by using the **RuleSetExecutionConfiguration** object.

Note Only one fact retriever implementation can be associated with a policy version.

To associate a fact retriever with a policy in the Business Rule Composer

1. In the Policy Explorer window, click the policy version with which you want to associate the fact retriever.
2. In the Properties window, click the **ellipsis** button () in the **FactRetriever** property to browse in the global assembly cache for a fact retriever object.

   Important The **ellipsis** button () does not appear until you click the **FactRetriever** row in the Property Window.

How to Develop Vocabularies

Rule conditions and actions are based on sources that may have detailed, difficult-to-read binding information that tells the user little or nothing about what they refer to. The Business Rules Framework enables you to create vocabularies to simplify the development of rules by offering users intuitive, domain-specific terminology that users can associate with rule conditions and actions.

You can identify data sources to use, create a new vocabulary, and add vocabulary definitions to it. You can save a version of your vocabulary to the rule store, and when it is complete, you can publish it to provide users with a well-defined, immutable set of terms that are bound to data references.

If you need to make changes to your vocabulary in the future, you can simply create a new version of the vocabulary that reflects the changes.

To create a vocabulary

1. In the Facts and Definitions window, click the **Vocabularies** tab.
2. Right-click the **Vocabularies** folder, and then click **Add New Vocabulary**.

   A new **vocabulary** folder appears under the **Vocabularies** folder

3. Click the new **vocabulary** folder, and then edit the name in the Properties window.

Note You must save everything (all versions of the definitions) before you can rename a vocabulary or a policy.

How to Create Vocabulary Definitions

You can use the Vocabulary Definition Wizard to create vocabulary definitions. You can define a vocabulary definition as a constant value, a range of values, a set of values, or elements of a .NET assembly, XML document, or database table. If you select a public variable, there will be **Get** and **Set** options just like in Database and XML definition wizard.

Alternatively, you can create a new vocabulary definition by selecting a fact from one of the three tabs—for example, a database column, an XML node, or a member of a .NET class—dragging the fact over to the **Vocabularies** tab, and dropping it to an unpublished version of a vocabulary.

To define a vocabulary definition as a constant value

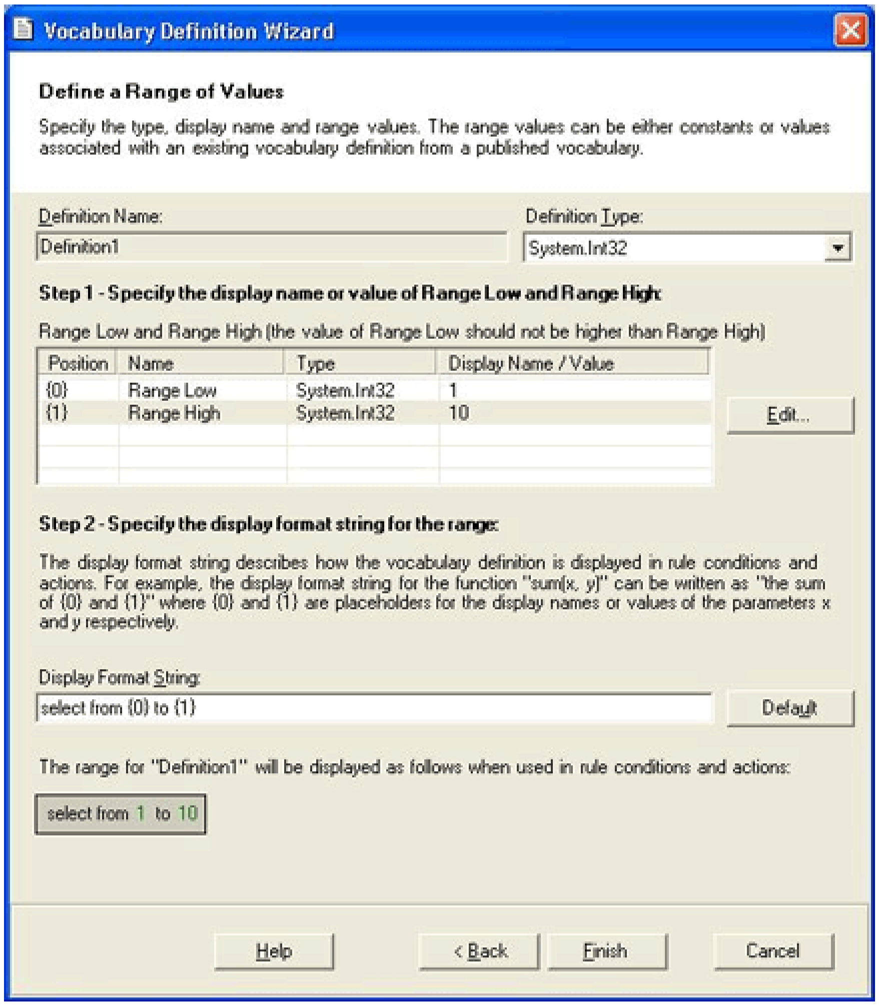
1. Right-click the vocabulary version, and then click **Add New Definition**.

2. In the Vocabulary Definition Wizard, select **Constant Value, Range of Values, or Set of Values**, and then click **Next**.
3. Edit the definition name and definition description.
4. Select **Constant Value**, and then click **Next**.
5. Select a definition type from the drop-down list of available system types.
6. Edit the display name and value, and then click **Finish**.

To define a vocabulary definition as a range of values

1. Right-click the vocabulary version, and then click **Add New Definition**.
2. In the Vocabulary Definition Wizard, select **Constant Value, Range of Values, or Set of Values**, and then click **Next**.
3. Edit the definition name and definition description.
4. Select **Range of Values**, and then click **Next**.
5. Select a definition type from the drop-down list.
6. Click **Range Low**, and then click **Edit** to specify the values for the lower range. The parameter definition dialog will be opened.
7. Select **Use Constant Value** and enter a constant value, or select **Use Definition from a Published Vocabulary** and browse to a vocabulary definition, and then click **OK**.
8. Repeat steps 6 and 7 for **Range High**.
9. Type the display format string or click **Default** to revert to the default display format string, and then click **Finish**.

10. **Range of values with formatted string**


Vocabulary Definition Wizard

Define a Range of Values

Specify the type, display name and range values. The range values can be either constants or values associated with an existing vocabulary definition from a published vocabulary.

Definition Name: Definition1

Definition Type: System.Int32

Step 1 - Specify the display name or value of Range Low and Range High:

Range Low and Range High (the value of Range Low should not be higher than Range High)

Position	Name	Type	Display Name / Value
{0}	Range Low	System.Int32	1
{1}	Range High	System.Int32	10

Step 2 - Specify the display format string for the range:

The display format string describes how the vocabulary definition is displayed in rule conditions and actions. For example, the display format string for the function "sum(x, y)" can be written as "the sum of {0} and {1}" where {0} and {1} are placeholders for the display names or values of the parameters x and y respectively.

Display Format String: select from {0} to {1}

The range for "Definition1" will be displayed as follows when used in rule conditions and actions:

select from 1 to 10

To define a vocabulary definition as a set of values

1. Right-click the vocabulary version, and then click **Add New Definition**.
2. In the Vocabulary Definition Wizard, select **Constant Value, Range of Values, or Set of Values**, and then click **Next**.
3. Edit the definition name and definition description.
4. Select **Set of Values**, and then click **Next**.
5. Enter the definition type and display name.
6. To add a member to the set, select **Use Constant Value** and enter a constant value, or select **Use Definition from a Published Vocabulary** and browse to a vocabulary definition, and then click **Add**.

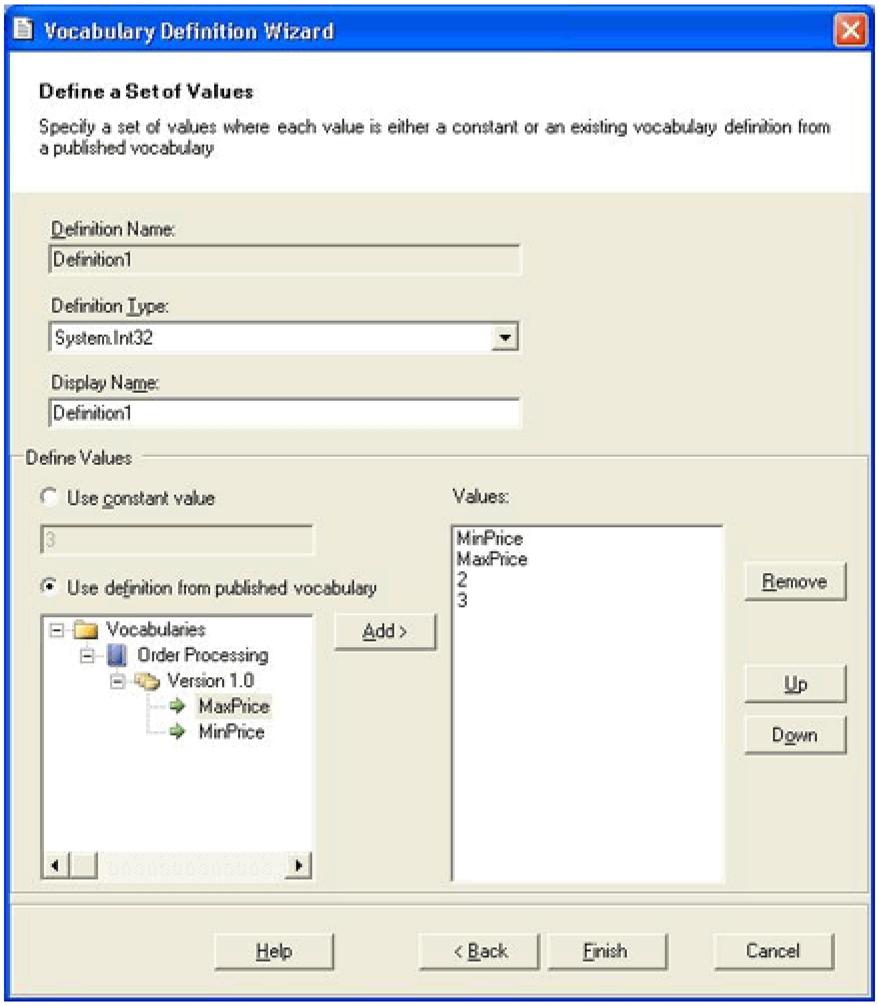
7. Repeat step 6, with any combination of constants or vocabulary definitions, for as many items as you want to include in your set.

8. To move a member within the relative order of the set, select it and then click **Up** or **Down**.

9. To remove a member from the set, select it and then click **Remove**.

10. When you have completed your set, click **Finish**.

Set of values


Vocabulary Definition Wizard
Define a Set of Values
Specify a set of values where each value is either a constant or an existing vocabulary definition from a published vocabulary
Define Values
Vocabularies
Order Processing
Version 1.0
MaxPrice
MinPrice
Add >
Remove
Up
Down
Help
< Back
Finish
Cancel

Definition Name: Definition1

Definition Type: System.Int32

Display Name: Definition1

Use constant value: 3

Use definition from published vocabulary

Values:
MinPrice
MaxPrice
2
3

To define a vocabulary definition as a .NET class or class member

1. Right-click the vocabulary version, and then click **Add New Definition**.

2. In the Vocabulary Definition Wizard, select **.NET Class or Class Member**, and then click **Next**.

3. Edit the **Definition Name** and **Description** fields.

4. Click **Browse**.

5. In the **.NET Assemblies** dialog box, select an assembly, and then click **OK**.

6. Expand the assembly node.

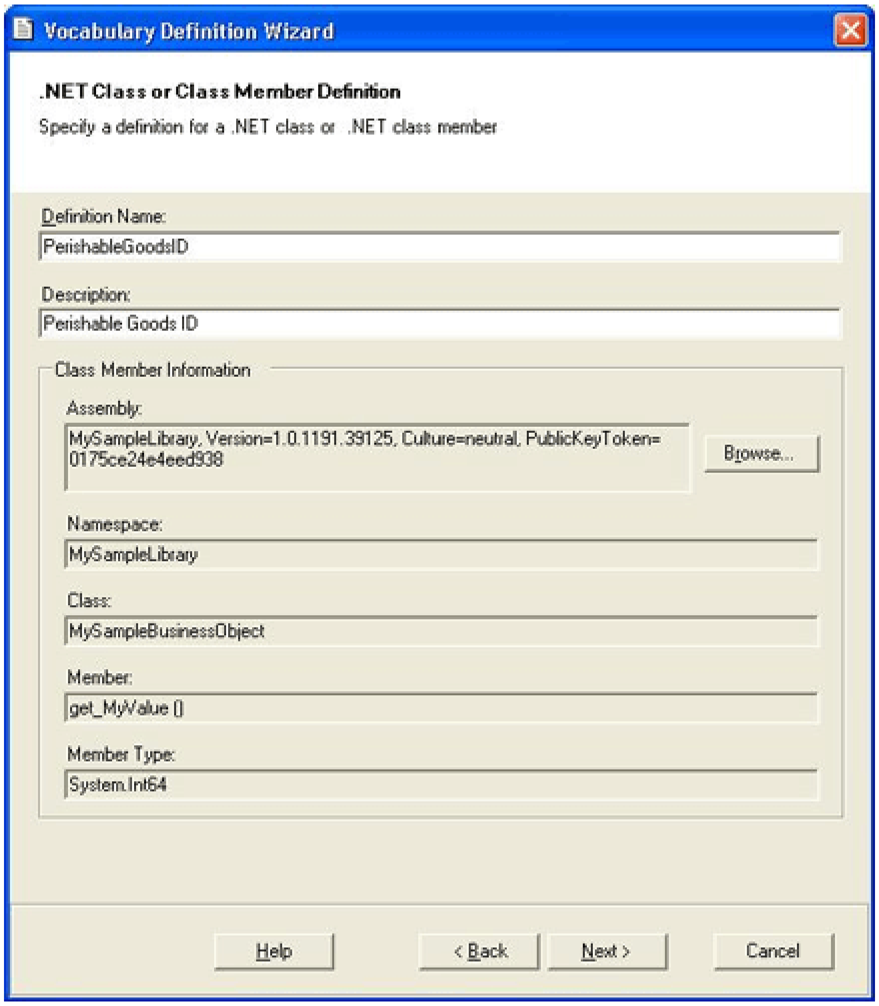
7. Select a class, or expand a class and select a class member, and then click **OK**.

8. Click **Next**, and specify the display name.

   For more information, see "To specify the display format of a vocabulary definition by using parameters" later in this topic.

9. Click **Finish**.

Net object vocabulary definition



.NET Class or Class Member Definition

Specify a definition for a .NET class or .NET class member

Definition Name: PerishableGoodsID

Description: Perishable Goods ID

Class Member Information

Assembly: MySampleLibrary, Version=1.0.1191.39125, Culture=neutral, PublicKeyToken=0175ce24e4eed938

Namespace: MySampleLibrary

Class: MySampleBusinessObject

Member: get_MyValue ()

Member Type: System.Int64

To define a vocabulary definition as an XML document element or attribute

1. Right-click the vocabulary version, and then click **Add New Definition**.

2. In the Vocabulary Definition Wizard, select **XML Document Element or Attribute**, and then click **Next**.

3. Type a definition name and a definition description.

4. Click **Browse** to locate a schema file and specify a document element or attribute.

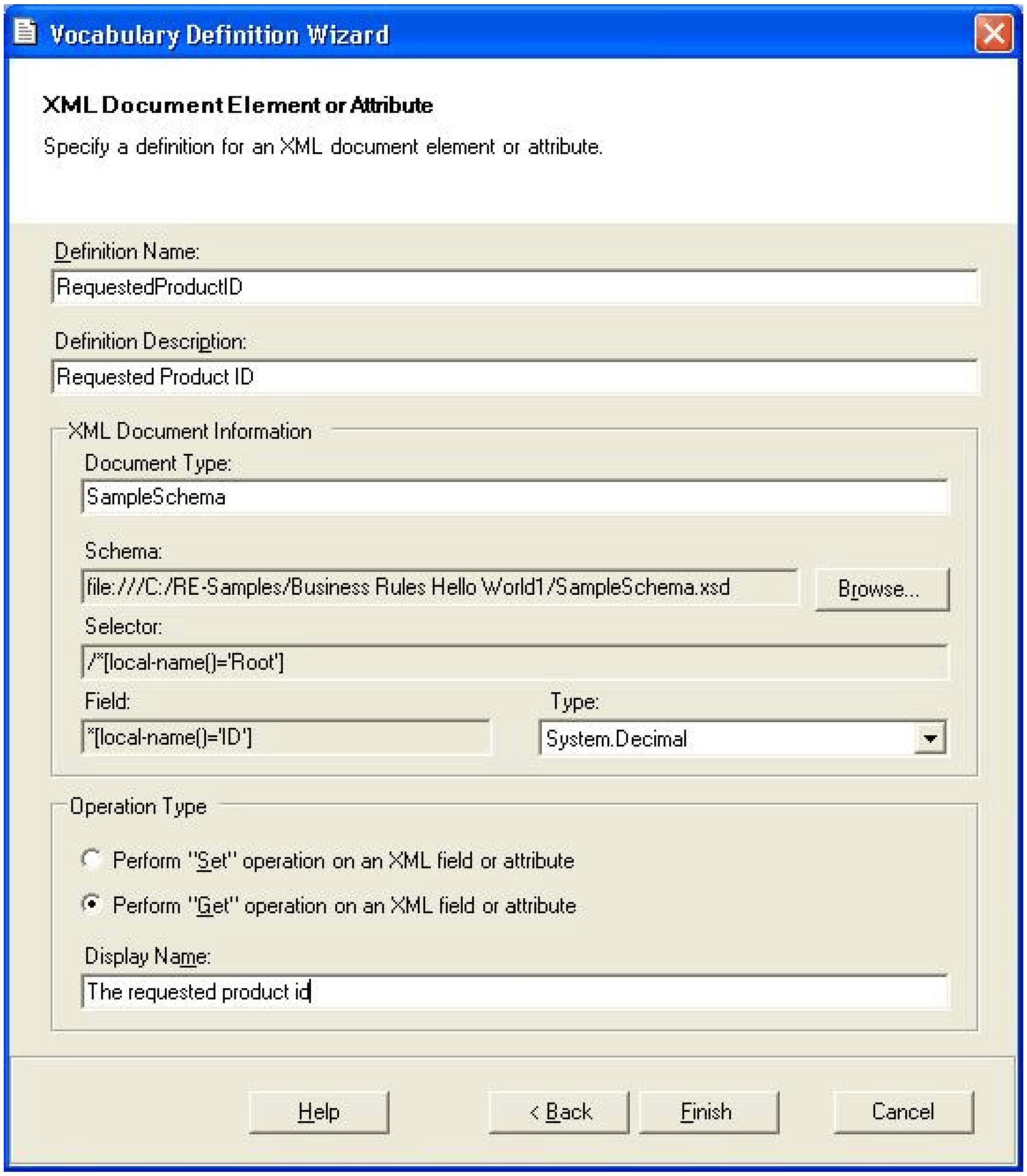
5. From the drop-down list, select a type that is compatible with the type of the element or attribute in the schema.

6. Select an operation type to indicate whether you plan to get the value of the element or attribute, or to set its value.

7. If you chose to set the value, click **Next**, and then specify the display format.

8. Click **Finish**.

XML vocabulary definition



Vocabulary Definition Wizard

XML Document Element or Attribute

Specify a definition for an XML document element or attribute.

Definition Name: RequestedProductID

Definition Description: Requested Product ID

XML Document Information

Document Type: SampleSchema

Schema: file:///C:/RE-Samples/Business Rules Hello World1/SampleSchema.xsd

Selector: /*[local-name()='Root']

Field: *[local-name()='ID']

Type: System.Decimal

Operation Type

Perform "Set" operation on an XML field or attribute [✓ Perform "Get" operation on an XML field or attribute]

Display Name: The requested product id

To define a vocabulary definition as a database table or database table column

1. Right-click the vocabulary version and then click **Add New Definition**.

2. In the Vocabulary Definition Wizard, select **Database Table or Database Table Column Definition**, and then click **Next**.

3. Type a definition name and definition description.

4. Click **Browse**.

5. Select a SQL Server computer to connect with. If the SQL Server computer is not currently started, select the **Start SQL Server if it is stopped** check box.

6. Select an authentication type. If you select SQL Server authentication, type your logon name and password, and then click **OK**.

7. Select the table or table column that you want to bind to, and then click **OK**.

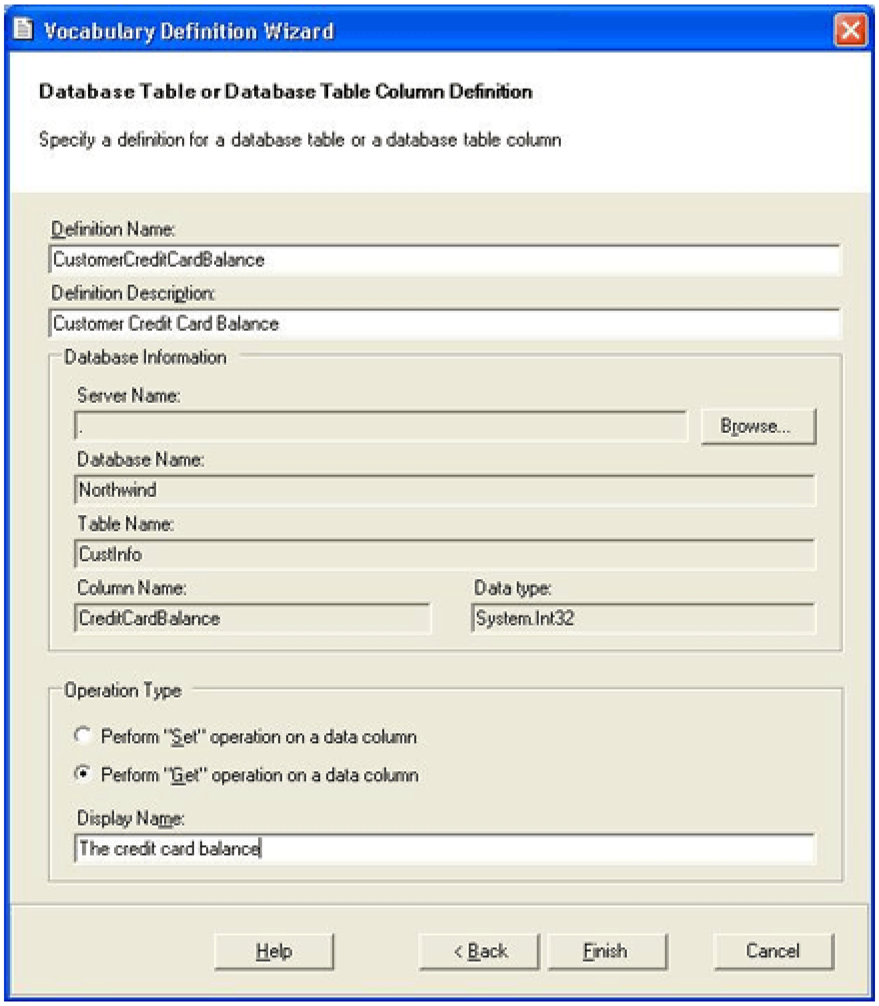
8. If you selected a table column, select whether you want to get its value or set its value. If you choose to get its value, type the display name. If you choose to set its value, click **Next** to specify the display format.

   For more information, see "To specify the display format of a vocabulary definition by using parameters" later in this topic.

9. If you selected a table, type a display name.

10. Click **Finish**.

11. **Database vocabulary definition**


Vocabulary Definition Wizard

Database Table or Database Table Column Definition

Specify a definition for a database table or a database table column

Definition Name: CustomerCreditCardBalance

Definition Description: Customer Credit Card Balance

Database Information

Server Name: .

Database Name: Northwind

Table Name: CustInfo

Column Name: CreditCardBalance

Data type: System.Int32

Operation Type

( ) Perform "Set" operation on a data column

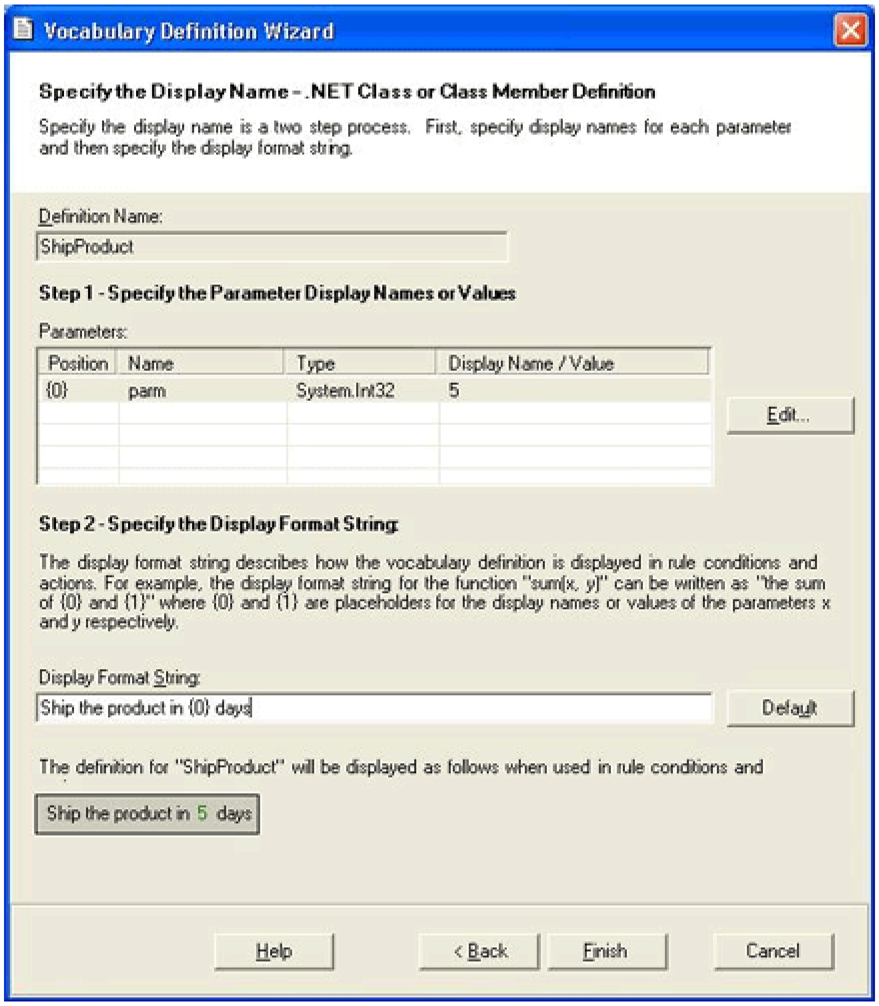
(●) Perform "Get" operation on a data column

Display Name: The credit card balance

To specify the display format of a vocabulary definition by using parameters

1. Select a parameter and then click **Edit**.
2. Select **Use Constant Value** and enter a constant value, or select **Use Definition from a Published Vocabulary** and browse to a vocabulary definition, and then click **OK**.
3. Type the display format string or click **Default** to revert to the default display format string, and then click **Finish**.

4. **Vocabulary definition with parameters**



Specify the Display Name - .NET Class or Class Member Definition

Specify the display name is a two step process. First, specify display names for each parameter and then specify the display format string.

Definition Name: ShipProduct

Step 1 - Specify the Parameter Display Names or Values

Parameters:

Position	Name	Type	Display Name / Value
{0}	parm	System.Int32	5

Step 2 - Specify the Display Format String:

The display format string describes how the vocabulary definition is displayed in rule conditions and actions. For example, the display format string for the function "sum(x, y)" can be written as "the sum of {0} and {1}" where {0} and {1} are placeholders for the display names or values of the parameters x and y respectively.

Display Format String: Ship the product in {0} days

The definition for "ShipProduct" will be displayed as follows when used in rule conditions and

Ship the product in 5 days

How to Modify Vocabulary Definitions

You can modify a vocabulary definition in an unpublished version of a vocabulary by using the Vocabulary Definition Wizard to change the display name or to change the binding associated with the display name.

This topic contains procedures for the following tasks:

- To modify a vocabulary definition
- To add a fact to a vocabulary or vocabulary definition

To modify a vocabulary definition

1. Right-click the vocabulary definition, and then click **Modify**.

2. Use the Vocabulary Definition Wizard as you would to create a new vocabulary definition.

To add a fact to a vocabulary or vocabulary definition

1. In the Facts and Definitions window, click the **Vocabularies** tab, and select the unpublished vocabulary version that you want to update.
2. Click **Databases**, **XML Schemas**, or **.NET Classes**.
3. Navigate through the data source hierarchy to the fact you want.
4. Click the fact and hold down the mouse button.
5. Drag the fact over the **Vocabularies** tab.

   The **Vocabularies** tab opens.
6. Drag the fact to the vocabulary version you want to modify.

How to Maintain Vocabulary Versions

When you have added vocabulary definitions to a version of your vocabulary, you can save the version to the rule store for further development, or you can publish the version to create a well-defined, immutable set of data-bound terms that are available to users to add to rules as they develop their policies. Note that fact that existing rules will still point to the old versions.

This topic contains procedures for the following tasks:

- To save a vocabulary version
- To publish a vocabulary version
- To create an updated version of a vocabulary
- To create an empty new version of a vocabulary

To save a vocabulary version

1. Right-click the version, and then click **Save**.

To publish a vocabulary version

1. Right-click the version, and then click **Publish**.

To create an updated version of a vocabulary

1. Right-click an existing version, and then click **Copy**.
2. Right-click the vocabulary folder, and then click **Paste**.

   A new version is created, with the same elements as the copied version.

To create an empty new version of a vocabulary

1. Right-click the vocabulary folder, and then click **Add New Version**.

How to Implement Negation as Failure

This section describes how to implement the **Negation as Failure** idiom. In your business logic, you often need to identify the actions to take when certain conditions do not hold. It is natural to write a rule like the following example:

If it is not raining, then walk to work.

Also, negation is often used to express an exception to a condition:

If a visitor buys a product on your Web site and the purchased product is not a coffee maker, then send a promotional e-mail message about the coffee maker.

In the Business Rules Framework, it is relatively easy to support exceptions by using the classical negation operator **NOT**. Describing the actions to take when certain conditions do not hold is somewhat more complicated, although it can also be expressed by using rule priorities and the classical **NOT** operator.

Start by writing a single rule that expresses the condition that does not hold and the actions to take when the condition is not true. Suppose that you have the following rules.

Rule 1—Priority 0

The first rule captures or expresses the idea without describing what it means to be able to send the form electronically. This is important because it isolates this rule from the mechanisms used to determine if a form can be sent electronically, which may change over time. After the first rule is in place you can add other, higher priority rules that describe the current meaning of SendFormElectronically.

Note that the rule engine has no way to reason about entities that are not in working memory. For example, there is no way to express a statement like this: "If an instance of a particular fact is not in the knowledge base, then perform a certain action." However, the **Exists** predicate can be used against XML documents to check the absence of document content. All facts referenced in rule conditions and actions need to be asserted into working memory to instantiate the rule. Therefore, there is no way to define and execute the following rules.

How to Handle Null and DBNull

This topic describes the expected behaviors when dealing with null values associated with different types, and discusses options for checking null or existence of a particular field or member.

XML

The following applies to XML:

- An XML value will never be returned from a document as null. It is either an empty string or a "does not exist" error. When it is an empty string, an error may occur for conversion of certain types, for example, fields specified as an integer type when building rules.

- The Business Rule Composer does not allow you to set a field to null or to set the type of a field to **object**.

- Through the object model you can set the type to **Object**. In this case, the value returned is the type to which the XPath evaluates— **float**, **boolean**, or **string**, depending on the XPath expression.

NET classes

The following applies to .NET classes:

- You are not allowed to compare to null if your return type is not an **object** type.

- You can pass null as a parameter for parameters that are not value types, but you may get a runtime error, depending on the implementation of the member.

- You can set fields of types derived from **Object** to null.

Data connection

The following applies to a data connection:

- You can compare any database table column type to null or set any column type to null, if the table allows nulls for the column.

- The Business Rule Composer automatically sets the member type in the binding to **object**, if you compare or set to null for a value type; it changes back to the original type if you reset or replace the argument.

- For a string type, it changes the type to **object** if you set it to null, but leaves it as a string if you compare against null.

- You cannot compare or set to **DBNull.Value** from the Business Rule Composer, because it will not change the column type to **object**.

- The engine will convert a DBNull value to null for comparison and will convert null to a DBNull value for insertion into the database.

- Tests will ignore null values (this is the way SQL Server works). For example, if you have the rule "IF db.column > 5 THEN .", only the rows that have a value in db.column are tested—rows with null are skipped.

TypedDataTable and TypedDataRow

The following applies to TypedDataTable and TypedDataRow:

- The Business Rule Composer changes the field type to **object** in the same manner as with **DataConnection**s.

- Tests will fail for null values, unless you are comparing to null. For example, there will be an error in the rule "IF db.column > 5 THEN ", if any row asserted has a null value.

  Note that because conditions are evaluated in parallel, tests like "IF db.column != NULL AND db.column > 5 THEN " will still fail, because both tests are potentially evaluated on every row.

Checking for null or existence

When writing business rules, it is natural to check for the existence of a field before comparing its value. However, if the field is null or does not exist, comparing the value will cause an error. Assume you have the following rule:

IF Product/Quantity Exists AND Product/Quantity > 1

An error will be thrown in the rule if Product/Quantity does not exist. One of the ways to circumvent this problem is to pass a parent node to a helper method that returns the element value if it exists or something else if it does not. See the following rules.

Rule 1

IF EXISTS(Product/Quantity) THEN ASSERT(CREATEOBJECT(typeof(Helper), Product/Quantity)

Rule 2

IF Helper.Value == X THEN...

Another possible solution is to create a rule like the following:

IF Product/Quantity Exist Then CheckQuantityAndDoSomething(Product/Quantity)

In the preceding example, the CheckQuantityAndDoSomething function will check the parameter value and execute if the condition is met.

Tracking and Monitoring Policies

This section describes how you can test policies and explains the tracking information.

In This Section

- How to Test Policies
- Policy Test Trace Output
- How to Use HAT to Monitor Rules

How to Test Policies

To test a policy, you need facts on which the rules can be executed. You can add facts by specifying values in XML documents or database tables that you will point to in the policy tester, or you can use a fact creator to supply to the engine an array of .NET objects as facts. For more information, see Creating a Fact Creator.

To test a policy in the Business Rule Composer

1. In the Policy Explorer window, click the policy version that you want to test.
2. Click the **Test Policy** button (green arrow) on the menu bar.

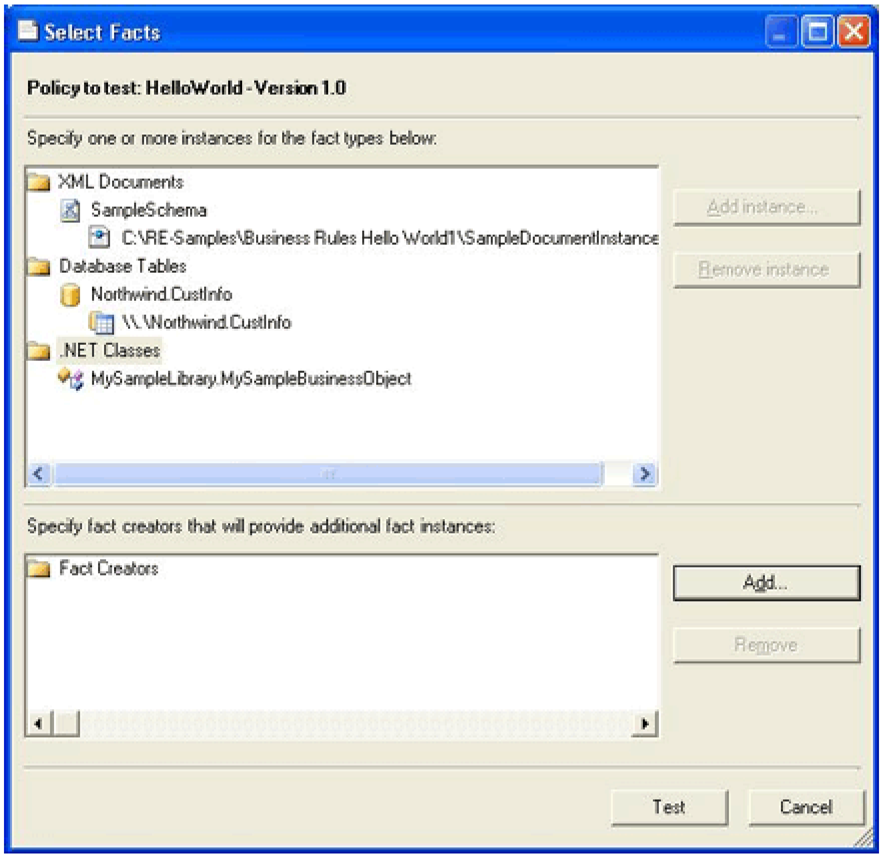
   The top pane displays the fact types that the policy rules reference.

3. To add a fact instance, click an **XML Document** or **Database Table** fact type, and then click **Add Instance**.
4. If you want to remove a fact instance, click the corresponding fact type, and then click **Remove Instance**.
5. If you want to add a fact creator that you have written, click **Add** in the fact creator pane.
6. Click **Test**.

The policy test trace output appears in the test output window.

7. Right-click in the output window to save, clear, select, or copy the output text.

Selecting fact to test a policy



Policy to test: HelloWorld - Version 1.0

Specify one or more instances for the fact types below:

XML Documents
SampleSchema
C:\RE-Samples\Business Rules Hello World1\SampleDocumentInstance
Database Tables
Northwind.CustInfo
\\.\Northwind.CustInfo
.NET Classes
MySampleLibrary.MySampleBusinessObject

Specify fact creators that will provide additional fact instances:

Fact Creators

Policy Test Trace Output

When you test your policy, the policy tester displays the output in the version information window. This section describes the activities included within the trace.

Note If a derived class is asserted but rules are written directly against members of the base class, an instance of the base class is asserted and conditions are evaluated against that instance.

This section contains:

- Policy Test Trace Output Information for Business Rules
- Policy Test Trace Output Examples

Policy Test Trace Output Information for Business Rules

This section provides information on the tracking information that is displayed when testing a policy in the Business Rule Composer. Very similar information is seen when viewing tracking results for policy execution in the Health and Activity Tracking tool.

There are four statement types that are displayed in the tracking output:

- Fact Activity
- Condition Evaluation
- Agenda Update
- Rule Fired

Each statement type is described below.

Fact activity

This statement indicates changes to the facts present in the working memory of the engine. The following is an example of a fact activity entry:

Rule Engine Instance Identifier

Unique identifier for the **RuleEngine** instance that provides the execution environment for the rule firing.

Ruleset Name

The name of the rule set (policy).

Operation

There are three types of operation that can occur in a fact activity:

Assert

The fact is being added to the working memory

Update

The fact is being updated by a rule and then needs to be reasserted into the engine to be re-evaluated, based on the new data and state.

Retract

The fact is being removed from the working memory

Object Type

The type of fact for a particular activity:

- DataConnection
- TypedDataTable
- TypedDataRow

  When a TypedDataTable is asserted all of the contained rows are asserted as TypedDataRows. TypedDataRows associated with a DataConnection are not asserted until a condition is evaluated and the resulting query is executed.

- TypedXmlDocument

  Assertions will be seen for both parent and child TypedXmlDocument instances.

Object Instance Identifier

Unique instance ID of the fact reference

Condition evaluation

This activity indicates the result of evaluating individual predicates. The following is an example of a condition evaluation entry:

The descriptions for some of the terms in the preceding example are as follows:

- **Test Expression.** A simple (unary or binary) expression within a rule.
- **Left Operand Value.** The value of the term on the left side of an expression.
- **Right Operand Value.** The value of the term on the right side of an expression.
- **Test Result.** The result of the evaluation, which is either True or False.

Agenda update

This activity indicates rules that are added to the rule engine agenda for subsequent execution. The following is an example of an agenda update entry:

The descriptions for some of the terms in the preceding example are as follows:

- **Operation.** Rules can be added or removed from the agenda.
- **Rule Name.** The name of the rule that is being added to the agenda.
- **Conflict Resolution Criteria.** The priority of a rule, which determines the relative order in which actions are executed (higher-priority actions are executed first)

Rule fired

This activity indicates the execution of a rule's actions. The following is an example of a rule fired entry:

Policy Test Trace Output Examples

This section contains examples of the policy test output for different types of facts.

Net Class

Example rule "TestRule1" in policy "LoanProcessing":

Output:

RULE ENGINE TRACE for RULESET: LoanProcessing 3/16/2004 9:50:28 AM

FACT ACTIVITY 3/16/2004 9:50:28 AM

Rule Engine Instance Identifier: 9effe3f9-d3ad-4125-99fa-56bb379188f7

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: MyTest.test

Object Instance Identifier: 872

CONDITION EVALUATION TEST (MATCH) 3/16/2004 9:50:28 AM

Rule Engine Instance Identifier: 9effe3f9-d3ad-4125-99fa-56bb379188f7

Ruleset Name: LoanProcessing

Test Expression: MyTest.test.get_ID > 0

Left Operand Value: 100

Right Operand Value: 0

Test Result: True

AGENDA UPDATE 3/16/2004 9:50:28 AM

Rule Engine Instance Identifier: 9effe3f9-d3ad-4125-99fa-56bb379188f7

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/16/2004 9:50:28 AM

Rule Engine Instance Identifier: 9effe3f9-d3ad-4125-99fa-56bb379188f7

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/16/2004 9:50:28 AM

Rule Engine Instance Identifier: 9effe3f9-d3ad-4125-99fa-56bb379188f7

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: MyTest.test

Object Instance Identifier: 872

DataConnection / TypedDataRow

Example rule "TestRule1" in policy "LoanProcessing":

Output:

```
RULE ENGINE TRACE for RULESET: LoanProcessing 3/16/2004 8:30:16 AM

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: DataConnection:Northwind:CustInfo

Object Instance Identifier: 874

CONDITION EVALUATION TEST (MATCH) 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Test Expression: select * from [CustInfo] where [CreditCardBalance] > 0

Left Operand Value:

Right Operand Value:

Test Result: True

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 177556

AGENDA UPDATE 3/16/2004 8:30:16 AM
```

```
Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 177559

AGENDA UPDATE 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 177558
```

```
AGENDA UPDATE 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83
```

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: DataConnection:Northwind:CustInfo

Object Instance Identifier: 874

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 177559

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 177558

FACT ACTIVITY 3/16/2004 8:30:16 AM

Rule Engine Instance Identifier: 1aad35bb-0599-470b-b0fa-73b3fa1dfb83

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 177556

The example above indicates that three rows in the CustInfo table met the condition in the rule. This caused three unique TypedDataRows to be asserted into the engine and an agenda update and rule firing to occur for each instance.

TypeDataTable / TypedDataRow

Example rule "TestRule1" in policy "LoanProcessing":

Output:

```
RULE ENGINE TRACE for RULESET: LoanProcessing 3/17/2004 11:27:35 AM

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataTable:Northwind:CustInfo

Object Instance Identifier: 377

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 376

CONDITION EVALUATION TEST (MATCH) 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Test Expression: TypedDataRow:Northwind:CustInfo.CreditCardBalance > 0

Left Operand Value: 500

Right Operand Value: 0

Test Result: True
```

```
AGENDA UPDATE 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 375

CONDITION EVALUATION TEST (MATCH) 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Test Expression: TypedDataRow:Northwind:CustInfo.CreditCardBalance > 0

Left Operand Value: 1000

Right Operand Value: 0

Test Result: True

AGENDA UPDATE 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Add
```

```
Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 374

CONDITION EVALUATION TEST (MATCH) 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Test Expression: TypedDataRow:Northwind:CustInfo.CreditCardBalance > 0

Left Operand Value: 35000

Right Operand Value: 0

Test Result: True

AGENDA UPDATE 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439
```

```
Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedDataTable:Northwind:CustInfo

Object Instance Identifier: 377

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Retract
```

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 375

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 374

FACT ACTIVITY 3/17/2004 11:27:35 AM

Rule Engine Instance Identifier: 0f7bcdf3-8103-4990-a740-acaeee386439

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedDataRow:Northwind:CustInfo

Object Instance Identifier: 376

TypedXmlDocument

Example rule "TestRule1" in policy "LoanProcessing":

Output:

RULE ENGINE TRACE for RULESET: LoanProcessing 3/17/2004 9:23:05 AM

FACT ACTIVITY 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedXmlDocument:Microsoft.Samples.BizTalk.LoansProcessor.Case

Object Instance Identifier: 858

FACT ACTIVITY 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Operation: Assert

Object Type: TypedXmlDocument:Microsoft.Samples.BizTalk.LoansProcessor.Case:/Root/EmploymentType

Object Instance Identifier: 853

CONDITION EVALUATION TEST (MATCH) 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Test Expression: TypedXmlDocument:Microsoft.Samples.BizTalk.LoansProcessor.Case:/Root/EmploymentType.TimeInMonths >= 4

Left Operand Value: 6

Right Operand Value: 4

Test Result: True

AGENDA UPDATE 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Operation: Add

Rule Name: TestRule1

Conflict Resolution Criteria: 0

RULE FIRED 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Rule Name: TestRule1

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedXmlDocument:Microsoft.Samples.BizTalk.LoansProcessor.Case

Object Instance Identifier: 858

FACT ACTIVITY 3/17/2004 9:23:05 AM

Rule Engine Instance Identifier: 51ffbea4-468f-4ce8-8ab7-977cadda2e2b

Ruleset Name: LoanProcessing

Operation: Retract

Object Type: TypedXmlDocument:Microsoft.Samples.BizTalk.LoansProcessor.Case:/Root/EmploymentType

Object Instance Identifier: 853

This example shows that a **TypedXmlDocument** was asserted into the engine with a document type of "Microsoft.Samples.BizTalk.LoansProcessor.Case". Based on the XPath selector defined in the rule, the engine then created and asserted a child TypedXmlDocument with type "Microsoft.Samples.BizTalk.LoansProcessor.Case:/Root/EmploymentType" based on the document type and selector string.

This child TypedXmlDocument evaluated to True in the condition, causing an agenda update and rule firing. The parent and child **TypedXmlDocument**'s were then retracted.

Update Function

Example policy "Order"

"InventoryCheck" Rule

"Ship" Rule

Output:

```
RULE ENGINE TRACE for RULESET: Order 3/17/2004 10:31:17 AM

FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Assert

Object Type: TestClasses.Order

Object Instance Identifier: 448

CONDITION EVALUATION TEST (MATCH) 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Test Expression: TestClasses.Order.inventoryAvailable == True

Left Operand Value: null

Right Operand Value: True

Test Result: False

FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Assert

Object Type: TestClasses.Shipment

Object Instance Identifier: 447
```

```
FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Assert

Object Type: TestClasses.Inventory

Object Instance Identifier: 446

CONDITION EVALUATION TEST (MATCH) 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Test Expression: TestClasses.Inventory.AllocateInventory == True

Left Operand Value: True

Right Operand Value: True

Test Result: True

AGENDA UPDATE 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Add

Rule Name: InventoryCheck

Conflict Resolution Criteria: 0

RULE FIRED 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Rule Name: InventoryCheck
```

```
Conflict Resolution Criteria: 0

FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Update

Object Type: TestClasses.Order

Object Instance Identifier: 448

CONDITION EVALUATION TEST (MATCH) 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Test Expression: TestClasses.Order.inventoryAvailable == True

Left Operand Value: True

Right Operand Value: True

Test Result: True

AGENDA UPDATE 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Add

Rule Name: Ship

Conflict Resolution Criteria: 0

RULE FIRED 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order
```

```
Rule Name: Ship

Conflict Resolution Criteria: 0

FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Retract

Object Type: TestClasses.Order

Object Instance Identifier: 448

FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Retract

Object Type: TestClasses.Shipment

Object Instance Identifier: 447

FACT ACTIVITY 3/17/2004 10:31:17 AM

Rule Engine Instance Identifier: 533f2fb6-a91f-49c1-8f36-e03a27ca9d72

Ruleset Name: Order

Operation: Retract

Object Type: TestClasses.Inventory

Object Instance Identifier: 446
```

In this example, the condition associated with the Ship rule evaluates to False the first time it is checked. However, when the InventoryCheck rule fires, the inventoryAvailable field on the Order is changed and an Update command is issued on the engine for the Order object. This causes the Ship rule to be reevaluated. This time the condition evaluates to true and the Ship rule fires.

How to Use HAT to Monitor Rules

You can use Microsoft BizTalk Server 2006 Health and Activity Tracking (HAT) to monitor rule activities and track the overall progress of an orchestration that uses the Business Rules Framework. The tracking information for rules that you see in HAT is the same as the output of policy testing in the Business Rule Composer.

For more information about using Health and Activity Tracking, see Using Health and Activity Tracking.

To configure tracking and monitoring for policies

- Click **Start**, point to **Program Files**, point to **Microsoft BizTalk Server 2006**, and then click **Health and Activity Tracking**.
- On the welcome page, click **Next**.
- On the menu bar, click **Configuration**, and then click **Rules**.
- Select the deployed policy version (Rule Set ID) on which you want to enable tracking.
- Select the appropriate check boxes from among **Track Fact Activity**, **Track Rule Firing**, **Track Condition Evaluation**, and **Track Agenda Update** to specify the activities that you want to track.

  Note Data shown for **Rules that did not fire** when tracking of **Rules that fired** is disabled is incorrect.

How to Deploy and Undeploy Policies and Vocabularies

You can use the Rule Engine Deployment Wizard to deploy or undeploy a policy.

In the Rule Engine Deployment Wizard, when you try to deploy, only published policy versions are shown in the drop-down list. When you try to undeploy, only deployed policy versions are shown in the drop-down list.

To deploy or undeploy a policy

- Click **Start**, point to **Program Files**, point to **Microsoft BizTalk Server 2006**, and then click Rule Engine Deployment Wizard.
- On the welcome page, click **Next**.
- Select either **Deploy Policy** or **Undeploy Policy**, and then click **Next**.
- From the drop-down lists, select an available SQL Server computer anddatabase, and then click **Next**.

- From the drop-down list, select a policy, and then click **Next**.
- Review the server, database, and policy information, and then click **Next**.
- Watch the progress of the deployment or undeployment. When it is finished, click **Next**.
- Review the completion status of the deployment or undeployment, and then click **Finish**.

Calling Business Rules in Orchestrations

You can invoke a policy (or rule set) from an orchestration by using the **Call Rules** shape. The policy invokes the rule engine, which operates on the rules in the policy.

In addition to using Call Rules, the rules engine can be programmatically called from expression code, for example, in an **Expression** or **Message Assignment** shape. For information about programmatically executing policies, see Executing Policies.

This section describes configuration information and the procedures required to call and execute a business policy from a BizTalk orchestration.

In This Section

1. Configuration Information
2. How to Use the Call Rules Shape [Duplicate1]

Configuration Information

To use the **Call Rules** shape, you must configure several areas. This topic describes the configuration issues you need to consider.

Transaction type

The parent scope where the **Call Rules** shape resides must be an **Atomic** transaction type.

Orchestration variables and fact types

In the orchestration or the scope where the **Call Rules** shape resides, you must have variables that match types used in the policy. If you do not have the correct types of variables, you cannot supply the correct parameters to the policy. Suppose that you have a **Call Rules** shape, CallMyRules, in MyScope scope, and you use CallMyRules to call MyPolicy. If a .NET class, MyClass, is used in MyPolicy, you must create a variable of a MyAssembly.MyClass type in MyScope. Similarly, if MyPolicy has **DataConnection** bindings, you must create a variable of a Microsoft.RuleEngine.DataConnection type in MyScope.

In addition to creating the variables in the scope that contains the **Call Rules** shape, you must also create instances for these scope variables. You can do this by adding an **Expression** shape to your scope. Using the preceding example, you should instantiate a MyAssembly.MyClass instance and a **DataConnection** instance. To instantiate the DataConnection instance, you do the following:

1. Create a **SqlConnection** instance and assign it to MySqlCon.

2. Create a **DataConnection** instance and assign it to dataConnection (the scope variable of **DataConnection** type), as shown in the following code fragment:

Message type and document type

You must ensure that the **DocumentType** property for XML nodes used in your business rule (that you implement in the Business Rule Composer) is the same as the **MessageType** for those XML nodes defined in the orchestration—it must match the **MessageType** of the message or message part that will be passed to the rule engine in the **Call Rules** shape.

For example, if you define a purchase order (PO) XML schema in a BizTalk project, the **MessageType** defined for this schema is **BTSProject.PO** (if **BTSProject** is the namespace or the project name using the default namespace).

In the case of the **PO\Amount** element, before you drop it into a rule definition, you must change its **DocumentType** property to **BTSProject.PO** in the properties window. You can access the properties window when any node in the schema is selected in the **XML Schemas** tab in the Fact Explorer. This change is applied for all elements in the schema, but is not persisted with the schema. It must be reset when the Business Rule Composer is launched or the schema is reloaded. This is required for vocabulary definitions based either on XML schema or rules built directly from XML schema. If you do not do this, you can still execute the policy, but the **Call Rules** shape will not work correctly, and message type will not appear in the **Specify policy parameters** list box in the **CallRules policy configuration** dialog box.

How to Use the Call Rules Shape [Duplicate1]

In Orchestration Designer, you can use the **Call Rules** shape to call a business policy.

To call a policy by using the Call Rules shape

1. From the Toolbox, drag the **Scope** shape to the orchestration.

2. Click the shape you have just created to select it.

3. Change the **Transaction Type** property to **Atomic**.

4. From the Toolbox, drag the **Call Rules** shape and drop it in the **Drop a shape from the toolbox here** box in the shape created in step 1.
5. Click the **CallRules_1** shape to select it.
6. In the **Name** property, type the new name.

To configure the Call Rules shape

- In Orchestration Designer, select the **Call Rules** shape you want to configure.
- Double-click the shape to display the **CallRules policy configuration** dialog box.
- In the **Select the business policy you wish to call** drop-down list, select the policy you need.
- In the **Specify policy parameters** list box, select a variable from the **Parameter Name** property.

How to Import and Export Policies and Vocabularies

You can use the Rule Engine Deployment Wizard to import or export a policy or vocabulary.

Note All vocabularies used by a policy or vocabulary must be imported first or you will get an error.

To import or export a policy or vocabulary

- Click **Start**, point to **Program Files**, point to **Microsoft BizTalk Server 2006**, and then click Rule Engine Deployment Wizard.
- On the welcome page, click **Next**.
- Select either **Export Policy or Vocabulary** or **Import and Publish Policy or Vocabulary**, and then click **Next**.
- From the drop-down lists, select an available SQL Server computer and database, and then click **Next**.
- From the drop-down list, select a policy or vocabulary, type a file name or click **Browse** to specify a file to export to or import from, and then click **Next**.
- Review the server, database, and policy or vocabulary information, and then click **Next**.
- Watch the progress of the import or export. When it is finished, click **Next**.
- Review the completion status of the import or export, and then click **Finish**.

Business Rules Framework Security

The Business Rule Engine operates in the security context of the hosting application. The identity of the rule engine instance during execution is that of the thread context that invokes the **Policy.Execute** method.

Default security configuration

When you install Microsoft® BizTalk® Server 2006, two Microsoft® Windows® groups are created by default, one for administrators and one for users: "BizTalk Server Administrators" and "BizTalk Application Users." These two Windows groups are members of BTS_ADMIN_USERS and BTS_HOST_USERS SQL Roles which are members of RE_ADMIN_USERS and RE_HOST_USERS SQL Roles respectively.

The default SQL roles are created whenever a rule store is created: BTS_ADMIN_USERS, BTS_HOST_USERS, RE_ADMIN_USERS and RE_HOST_USERS.

Default Windows groups	SQL roles
BizTalk Server Administrators	RE_ADMIN_USERS
BizTalk Application Users	RE_HOST_USERS

Only users in the RE_ADMIN_USERS role can execute the stored procedures that update the deployment and entity access protection configuration tables. This means that the deployment, undeployment, and protection configuration are all done only by rule engine administrators. Users in the RE_HOST_USERS role can execute the other stored procedures in the SQL rule store.

Regardless of the order of installation of the rule engine, the rule engine configuration process grants the Rule Engine Update Service account membership to the RE_HOST_USERS SQL role, if it does not already have database access. However, if the rule engine is installed after the initial BizTalk installation the BizTalk, host-specific users group is not added to the BTS_HOST_USERS SQL role in the Rule Engine database because host creation has already been completed. You need to perform this step manually.

Artifact level security

In addition to the default security configuration, the Business Rule Engine can also provide security at the artifact—policy and vocabulary level.

Each policy or vocabulary version has one or more authorization groups associated with it. An Authorization group is a named list of Microsoft Windows users, SQL users, SQL roles, and Windows groups, with a particular access level on each type.

When a new policy or vocabulary is created in the rule store, only the user who created it and the rule engine administrator have both read/execute and modify/delete access by default. The rule engine administrator can configure which users (processes operate under user credentials) have the access level, or rights, to perform different operations—read/execute, modify/delete, full permission, or no permission.

Artifact specific security is not enabled by default. Setting artifact level security is not currently available through the user interface. However, it can be set programmatically with the Business Rule Engine administrator credential. The following code fragment shows how to create a new authorization and associate the group to a rule set.

Important Security Notes for the Business Rule Engine

This topic summarizes known security issues in Microsoft BizTalk Server 2006 and the steps you must take to mitigate the security risks.

Malicious schema input causing Denial of Service attack

While asserting facts, each rule is verified against every object that matches the supported types within a policy. Suppose there is a rule in a policy that uses one of the elements in a schema passed by using a selector. Now if the instance if this element/attribute that matches the selector is repeated thousands of times, every such instance is asserted, causing performance degradation and subsequent possible Denial of Service (DoS).

To mitigate this potential issue, you need to validate all ambiguous input that is passed while executing a policy.

RuleSet not validating objects before asserting the facts

Any schema instance passed as a fact to the **RuleSet** is not validated against the schema before asserting any rules using selectors. You should validate all input that is passed while executing a policy.

Expected behaviors of the Business Rule Composer when RuleStore security is on

You can enable the role-based security feature for the rule store by calling the **EnableAuthorization** method of the **RuleStore** class. When this security feature is enabled, the expected behaviors in the Business Rule Composer are as follows:

1. The object model filters out rule sets and vocabularies to which the user does not have read access. Therefore, they do not appear in the Business Rule Composer.

2. If the user does not have write access to a policy or a vocabulary, any save attempt causes an exception to be thrown.

User types for rule store administrator

The rule store administrator has the privilege to define an authorization group for artifacts saved in the rule store. However, note the following differences between two types of user to which the rule store administrator can belong:

- When the rule store administrator is a Windows user, which means using Windows authentication to connect the rule store, the rule store administrator can define an authorization group whose user is a Windows group or a Windows user.

- When the rule store administrator is a SQL user, which means using SQL authentication to connect the rule store, the rule store administrator cannot define an authorization group whose user is a Windows group, but can define an authorization group whose user is a Windows user.

User cannot associate an authorization group with an artifact without sufficient rights

An artifact creator that is neither a SQL dbo user nor a member of RE_ADMIN_USERS, and does not have MODIFY_DELETE permission to an artifact, cannot associate a new authorization group with the artifact. The following scenario is an example of this behavior:

- Rule set creator is not dbo, is not in the RE_ADMIN_USERS group, and does not have MODIFY_DELETE permission after the rule set is created.

- Creator creates a rule set.

- Member of the RE_ADMIN_USERS group creates authorization AG1 with MODIFY_DELETE permission to User2.

- Creator associates AG1 with the rule set.

- Enable the rule store authorization.

- Member of the RE_ADMIN_USERS group creates authorization AG2 with READ_EXECUTE permission to User2.

- Creator associates AG2 with the rule set. Although the creator does not have sufficient right to do so, no error message appears.

- Attempt to read the rule set by User2 fails.

Business Rule Engine Security Recommendations

The Business Rule Engine is the runtime component of the Business Rule Framework. With the Business Rule Framework, you can connect highly readable, declarative, semantically rich rules to any business objects (.NET components), XML documents, or database tables. For more information about the Business Rule Framework, see Creating Business Rules Using Business Rules Editor . For more information about the Business Rule Engine, see Rule Engine.

There are no Windows user groups for the Business Rule Engine, only individual accounts. BizTalk Server restricts access to the Business Rule Engine resources by using two SQL Server roles:

The RE_Admin_Users SQL Server role is for users that need to perform administrative tasks in the Business Rule Engine, such as deploying rules. Members of the RE_Admin_Users SQL Server role include BizTalk administrators.

The RE_Host_Users group is for all other users of the Business Rule Engine that do not require administrative user rights, and perform tasks such as reading and executing rules. Members of the RE_Host_Users role include the BizTalk_Host_Users role. You can use the SQL Server roles to implement permissions to the Business Rule Engine independent from the BizTalk Server permissions. For more information about the minimum permission needed to use the Business Rule Engine, see **Minimum Security User Rights**. It is recommended you follow these guidelines for securing and deploying the Business Rule Engine in your environment.

- BizTalk Server gives the account you used to install the update service logon as service rights and adds it to the RE_Host_Users SQL Server role on the Business Rule Engine database. If the account you use for installation is not the same you are going to use for running the update service, you must remove the installation account from the RE_Host_Users SQL Server role.

- If you use the Update service component, you must install it on all BizTalk runtime computers. In order to retrieve a rule from the Rule Engine database, the update service impersonates the caller of the service.

- By default, all BizTalk Hosts have the same level of access to rules engine artifacts. There is not per-host segregation of security. You can configure per-policy security by using the rule engine APIs. For more information about how to configure per-policy security, see.

Programming with Business Rules Framework

This section covers Microsoft® BizTalk® Server programming related tasks and examples of business rules.

In This Section

How to Execute Policies How to Create a Fact Retriever How to Create a Fact Creator Transaction Support for DataConnection How to Iterate ArrayList in Business Rules How to Edit XPath Selector to Process Multiple Records How to Log Tracking Information to File How to Script Import/Export and Deployment of Business Policies How to Use the Update Command to Re-retrieve Values How to Analyze Multiple Objects of the Same Type in a Business Rule

How to Execute Policies

You can execute your policies within the context of a business application such as a BizTalk orchestration. You simply create a new instance of a policy, and then execute the policy on a set of facts.

How to Create a Fact Retriever

A fact retriever is a component that is used to assert instances of long-term facts into a policy during its execution. You can implement the **IFactRetriever** interface and configure a policy version to use this implementation at run time to bring in the long-term fact instances. The policy version invokes the **UpdateFacts** method of the fact retriever implementation on every execution cycle, if a fact retriever is configured for that particular version.

To specify a fact retriever for a policy

You could use the following code to configure the rule set to use a class named "Retriever" in the assembly named "MyAssembly" as the fact retriever.

You can design the fact retriever with the required application-specific logic to connect to the required data sources, assert the data as long-term facts into the engine, and specify the logic for refreshing or asserting new instances of the long-term facts into the engine. Until updated, the values that are initially asserted into the engine and consequently cached will be used on subsequent execution cycles. The fact retriever implementation returns an object that is analogous to a token and can be used along with the **factsHandleIn** object to determine whether to update existing facts or assert new facts. When a policy version calls its fact retriever for the first time, the **factsHandleIn** object is always null and then takes the value of the return object after the fact retriever's execution.

Note that for the same rule engine instance, a long-term fact only needs to be asserted once. For example, when you use the **Call Rules** shape in an orchestration, the policy instance is moved into an internal cache. At this time, all short-term facts are retracted and long-term facts are kept. If the same policy is called again, either by the same orchestration instance or by a different orchestration instance in the same host, this policy instance is fetched from the cache and reused. In some batch processing scenarios, several policy instances of the same policy could be created. If a new policy instance is created, you must ensure that the correct long-term facts are asserted.

Additionally, you would need to write custom code to implement the following strategies:

- Know when to update the long-term facts
- Keep track of which rule engine instance uses which long-term facts

The following sample code shows different fact retriever implementations, which are associated with MyPolicy to assert MyTableInstance as a long-term fact, using different binding types.

How to Create a Fact Creator

You can write a fact creator to create instances of your facts. Your fact creator must implement **IFactCreator** and its **CreateFacts** method and **GetFactTypes** method. After you have created your fact creator dll, you can browse to it from within the policy tester. The following is an example of a fact creator implementation.

Transaction Support for DataConnection

If you update the database rows inside rule actions, this update in memory needs to be committed to the database table in a transactional manner. To do this, you need to explicitly provide a transaction to the **DataConnection** object during initialization. **DataConnection** has constructors that take a transaction as an additional parameter. The transaction is not automatically committed with an **Update** method call, because users can decide to commit or roll back based on other considerations**.**

How to Iterate ArrayList in Business Rules

This section provides an example of iterating through members of an **ArrayList** in business rules.

Assume you have an **ArrayList** with a collection of **MyClass** objects. Your business rules would look like the following.

Rule A

IF 1==1

THEN Assert (ArrayList.GetEnumerator)

An **IEnumerator** type is asserted into the working memory, because the rule condition (1==1) always evaluates to true.

Rule B

IF IEnumerator.MoveNext

THEN Assert (IEnumerator.get_Current)

Update (IEnumerator)

As the rule iterates through the **ArrayList**, each **MyClass** object in the collection is asserted into the working memory.

Rule C

IF MyClass.MyProperty==2

THEN <Do Something...>

This rule executes action(s) when the property value of the object is matched in the condition

How to Edit XPath Selector to Process Multiple Records

As explained in the Assert section of "Engine Control Functions," separate child TypedXmlDocuments are created when a TypedXmlDocument is asserted into the engine. The engine determines which child TypedXmlDocuments to create based on the XPath selectors defined in the rules. When you build rules in the Composer, the XPath Selector value defaults to the node above the node selected in the XML Schemas tab in the Facts Explorer. The XPath Field value defaults to the selected node itself, relative to its parent node.

In some situations you may want to customize the default XPath that the Composer creates when building rules. Assume the following sample XML document.

Assume that you want to build a rule that calculates the Total value for each Orderline. Your rule would look like the following.

IF 1==1

THEN **/Order/Orderline/**Total = (**/Order/Orderline/Hat/**Cost + **/Order/Orderline/Shirt/**Cost)

The bold portion of the XPaths indicate the Selector portion and the remainder represents the Field XPath. These are the defaults built by the Composer. Running this policy, however, would result in the creation of 6 objects—2 Orderline objects, 2 Hat objects, and 2 Shirt objects. The Orderline totals would be calculated for each combination of Hat and Shirt objects and the totals would always be set to the same value, which resulted from the last execution of the rule. The rule would fire 8 times. This is not what is intended in this scenario.

One solution would be to edit the XPath values to be as follows.

IF 1==1

THEN **/Order/Orderline/**Total = (**/Order/Orderline/**Hat/Cost + **/Order/Orderline/**Shirt/Cost)

The Selector XPath values for all three fields have been set to the same /Order/Orderline value and the Field XPath values have been edited accordingly. This is done by changing the Selector and Field XPath values in the Properties window when a node is selected in the XML Schemas Fact tab. This should be done prior to dragging the field into a rule argument.

Executing the policy with this change would result in the Total value being correctly calculated for each Orderline based on the Cost values of the Shirt and Hat nodes within that Orderline.

How to Log Tracking Information to File

When the Business Rule Engine executes, it passes execution tracking information to an interceptor. The engine is configured to use the BizTalk interceptor which writes to Health and Activity Tracking (HAT). If you are calling the engine outside of BizTalk orchestration, you can pass the interceptor you want to use when you execute the policy. In addition to the BizTalk interceptor, you can also use the **DebugTrackingInterceptor** to output the tracking information to file. The following code example demonstrates how to use **DebugTrackingInterceptor**.

How to Script Import/Export and Deployment of Business Policies

This section describes how to programmatically import/export and deploy policies using **RuleSetDeploymentDriver**.

How to Use the Update Command to Re-retrieve Values

The default value of **SideEffects** property is **false** for the members of typed facts (XML and database). For more information, see Rule Action Side Effects. Therefore, you have to call Update on the object if its value needs to be used elsewhere.

Assume you have an XML node MyXmlNode, which contains MyXmlField. The following rule will fire for each instance of MyClass that meets the condition.

IF MyClass.PropertyA== MyClass.PropertyB

THEN MyXmlField = MyXmlField + 1

Update (MyXmlNode)

Assume that multiple instances of MyClass are asserted into the engine and meet the conditional check. Because the value of MyXmlField changes with each firing of the rule, the Update command should be used. This would cause the engine to re-retrieve the value each time it is referenced, giving the result expected in this scenario.

How to Analyze Multiple Objects of the Same Type in a Business Rule

In many scenarios, you will write a business rule against a type and expect each instance of the type that is asserted into the engine to be separately analyzed and acted upon by the rule. In some scenarios, however, you will want to analyze multiple instances of a given type simultaneously in a rule.

Take for example a rule that uses instances of the **FamilyMember** class.

The rule identifies a **FamilyMember** instance that is a **Father** and another instance that is a **Son**. If the instances are related by surname, the **Son** instance is added to a collection of children on the Father instance. If each **FamilyMember** instance were analyzed separately in the rule, the rule would never fire, because in this scenario, the **FamilyMember** only has one role—**Father** or **Son**.

Therefore, you must indicate to the engine that multiple instances should be analyzed together in the rule, and you need a way to differentiate the identity of each instance in the rule. The **Instance ID** property is used to provide this functionality. This field is available in the Properties window when a fact is selected in Facts Explorer. You should change the value of the field prior to dragging a fact or member into a rule.

Using the **Instance ID** property, the rule would be rebuilt. For those rule arguments that use the **Son** instance of **FamilyMember**, the **Instance ID** field is changed from the default of 0 to 1. When the Instance ID is changed from 0 and the fact or member is dragged into the rule editor, the value of Instance ID will show up in the rule after the class.

Now, assume that a **Father** instance and a **Son** instance are asserted into the engine. The engine will evaluate the rule against the various combinations of the instances. Assuming that the **Father** and **Son** instance have the same surname, the **Son** instance will be added to the **Father** instance as intended.

Rule Engine Configuration and Tuning Parameters

The following table contains a list of registry keys that may be useful for configuration validation and troubleshooting. These registry keys are stored under **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\BusinessRules\3.0**.

With the exception of the first three keys listed, these keys are intended to allow products—and not users—to customize the rule engine. All of them are created upon installation; however, no interface is provided to set any of these values.

The definitions for the table columns are as follows:

- **Name**. Name of the registry key.
- **Use**. A brief description about the location or use of the key.
- **Accessor**. Name of function in configuration.cs used to retrieve the value.
- **Config default**. Value returned if the key does not exist.
- **Install default**. Value set by BizTalk Server when installing the rule engine.

Name	**Description**	**Accessor**
InstallPath	Location of the schema file.	Configuration.SchemaLocation
DatabaseServer	Database server used.	Configuration.DatabaseServer
DatabaseName	Name of database to be used.	Configuration.DatabaseName
PubSubAdapterAssembly	Assembly name of pub/sub adapter.	Configuration.PubSubAdapterDll
DeploymentDriverAssembly	Assembly name of deployment driver.	Configuration.DeploymentDriverDll
DeploymentDriverClass	Class name of deployment driver.	Configuration.DeploymentDriverClass
TrackingInterceptorAssembly	Assembly name of tracking interceptor.	Configuration.TrackingInterceptorDll
TrackingInterceptorClass	Class name of tracking interceptor.	Configuration.TrackingInterceptorClass
TranslationTimeout	Maximum time in milliseconds that can	Configuration.TranslationTimeout

	be used to translate a ruleset. **Note:** This can be overridden on a per-ruleset basis by using the RuleSetConfiguration).	
UpdateServiceName	Name of the Update service, used by .NET remoting to locate the service.	Configuration.RemoteUpdateServiceName
UpdateServiceHost	Computer hosting the Update service, used by .NET remoting to locate the service. **Note:** The service currently restricts incoming messages to same machine only.	Configuration.RemoteUpdateServiceMachine
UpdateServicePort	TCP port number used by the Update service, used by .NET remoting to locate the service.	Configuration.RemoteUpdateServicePort
CacheEntries	Maximum number of rulesets cached by the Update service.	Configuration.RemoteUpdateCacheCount
CacheTimeout	Time in seconds for entries to age out of the Update service cache.	Configuration.RemoteUpdateCacheTime
PollingInterval	Time in seconds for the Update service to check SqlRuleStore for updates.	Configuration.RemoteUpdatePollingInterval
UpdateServiceServerSinkAssembly	Assembly name of any security sink to be added into server channel for communication with	Configuration.UpdateServiceMessageServerSi

	the Update service. **Note:** If empty string, no security sink added.	
UpdateServiceServerSinkClass	Class name of any security sink to be added into server channel for communication with the Update service.	Configuration.UpdateServiceMessageServerSi
UpdateServiceClientSinkAssembly	Assembly name of any security sink to be added into client channel for communication with the Update service. **Note:** If empty string, no security sink added.	Configuration.UpdateServiceMessageClientSin
UpdateServiceClientSinkClass	Class name of any security sink to be added into client channel for communication with the Update service.	Configuration.UpdateServiceMessageClientSin
UpdateServiceSecurityPackage	Security package to be used by security sinks for communication with the Update service. **Note:** Possible values supported by RuleEngineExtensions are NTLM, Kerberos, or negotiate.	Configuration.UpdateServiceMessageSecurityI
UpdateServiceAuthenticationLevel	Authentication level to be used by security sinks for communication with the Update service.	Configuration.UpdateServiceMessageAuthenti

	Note: Possible values supported by RuleEngineExtensions are call, packetintegrity, or packetprivacy.	
UpdateServiceImpersonationLevel	Impersonation level to be used by security sinks for communication with the Update service. **Note:** Possible values supported by RuleEngineExtensions are identify, impersonate, or delegate.	Configuration.UpdateServiceMessageImperso

Using Web Services

Microsoft® BizTalk® Server 2006 provides built-in support for Web services. BizTalk Server enables the reuse and aggregation of all your existing Web services within your orchestrations. You can also publish (expose) your orchestrations as Web services to separate the Web service logic from the business process logic.

BizTalk Server implements support for native adapters in Web services. Native adapter support provides scalability, fault tolerance, and tracking capabilities for Web services without writing a single line of code. For information about the SOAP adapter, see SOAP Adapter.

The Web services support in BizTalk Server falls into two categories: consuming or calling Web services and publishing or creating Web services.

Before you consume or publish a Web service, you should have an understanding of XML Web services in ASP.NET. For information about the basics of XML Web services, see the article "XML Web Services Basics" at http://go.microsoft.com/fwlink/?LinkId=25489.

Consuming Web services

You can consume (call) Web services from within an orchestration. You can aggregate several Web services into single orchestration to complete an entire business process.

Publishing Web services

You can publish Web services using the BizTalk Web Services Publishing Wizard. Orchestrations and send adapters can use these published Web services.

Using SOAP headers

BizTalk Server provides support for defined and unknown SOAP headers. BizTalk Server creates a context property for each defined SOAP header in the Web service.

In This Section

- Consuming Web Services
- Publishing Web Services
- Using SOAP Headers

Consuming Web Services

Consuming Web services enables you to add existing Web services to your business process. You can aggregate several Web services into a single orchestration.

You can consume (call) a Web service from your orchestration by using Web ports. To consume a Web service from an orchestration, you create a Web port and construct Web messages.

You can use SOAP headers with the consumed Web service, change the URI of a consumed Web service, and dynamically set the URI for a consumed Web service.

For information about configuring a SOAP receive handler, see How to Configure a SOAP Receive Handler . For information about configuring a SOAP receive location, see **Configuring a SOAP Receive Location**.

In This Section

- Adding Web References
- Constructing Web Messages
- Creating Web Ports
- Considerations When Consuming Web Services

Adding Web References

Before you can add a Web port, you need to add a Web reference to your BizTalk project. A Web reference is a description of a Web service that is available to your project. When you add a Web reference to your project, BizTalk creates an orchestration Web port type, Web message types, Reference.map (map file), Reference.odx (orchestration file), <*WebService*>.disco (discovery file), and <*WebService*>.wsdl (Web Service Description Language file) to your project. If your Web Service Description Language (WSDL) file contains schema Web message types, BizTalk adds Reference.xsd to your project.

A Web reference includes:

1. A Universal Resource Locator (URL) for the Web service.

2. A WSDL file that offers information about the service such as available methods, ports, and message types.

3. A reference map (Reference.map).

When you add a Web reference, all the Web methods for that Web service must be compatible with BizTalk Server. You cannot specify conditional attributes for specific Web methods in a Web service.

You add a Web reference to your project by using the **Add Web Reference** dialog box. For more information about adding Web references, see Using the Project Menu.

If you are planning to export your orchestration using the Business Process Execution Language (BPEL) export process, you cannot reference a Web reference from an existing BizTalk project. When you reference a Web reference from an existing BizTalk project, the BPEL export process will auto-generate a second WSDL file and you will lose your binding information.

Constructing Web Messages

You construct a Web message from a Web message type. When you add a Web reference, BizTalk automatically creates Web message types, which BizTalk creates based on the Web methods from the added Web service. You add a Web message to your orchestration, setting the message type to one of the Web message types. You create individual message parts based on primitive .NET or schema types. You can construct a Web message that contains no message parts.

Web messages types

Web message types are identical to a normal message type, except you cannot modify, rename or delete them. To delete a Web message type, you must remove the Web reference from your BizTalk project.

BizTalk creates one request and one response Web message type for each Web method in the added Web service. If the Web method is a one-way operation, BizTalk only creates a request Web message type. A request Web message type contains one message part for each input parameter of the Web method. A response

Web message type contains one message part for the return value and one message part for each output parameter of the Web method.

Depending on the Web method parameter (input or output), BizTalk creates a Web message type from a primitive .NET type or a schema type. If the Web method parameter is a primitive .NET type, the message part uses a primitive .NET type. If the Web method parameter is a schema type, BizTalk adds the schema type to the BizTalk project as a schema in Reference.xsd. The schema is the basis for the message part. You can find Reference.xsd in the Web references folder.

Alternatively, you can create both primitive and schema .NET types by calling a .NET class. For more information on creating message types by using a .NET class, see Constructing Messages in User Code.

Web messages

Web messages are the messages you use when you consume (call) a Web service. You add a Web message to an orchestration the same way that you add a regular message, except that you set the message type to one of the Web message types that BizTalk created when you added a Web reference.

Message parts

After you create the Web message, you construct the individual message parts. If your message part uses a primitive .NET type, you use a **Message Assignment** shape. If your message part uses a schema type, you use a **Transform** shape.

In This Section

1. How to Add Web Messages
2. How to Determine a Web Message Part Type
3. How to Construct a Web Message Part from a Primitive .NET Type
4. How to Construct a Web Message Part from a Schema Type
5. How to Construct a Web Message with No Message Parts

How to Add Web Messages

You create Web message variables in the Orchestration View window. You must set the message type for the Web message to a Web message type.

To add a Web message

1. With an orchestration open, on the **View** menu, click **Other Windows**,and then click **Orchestration View**.

2. Right-click the **Message** node and then click **New Message**.

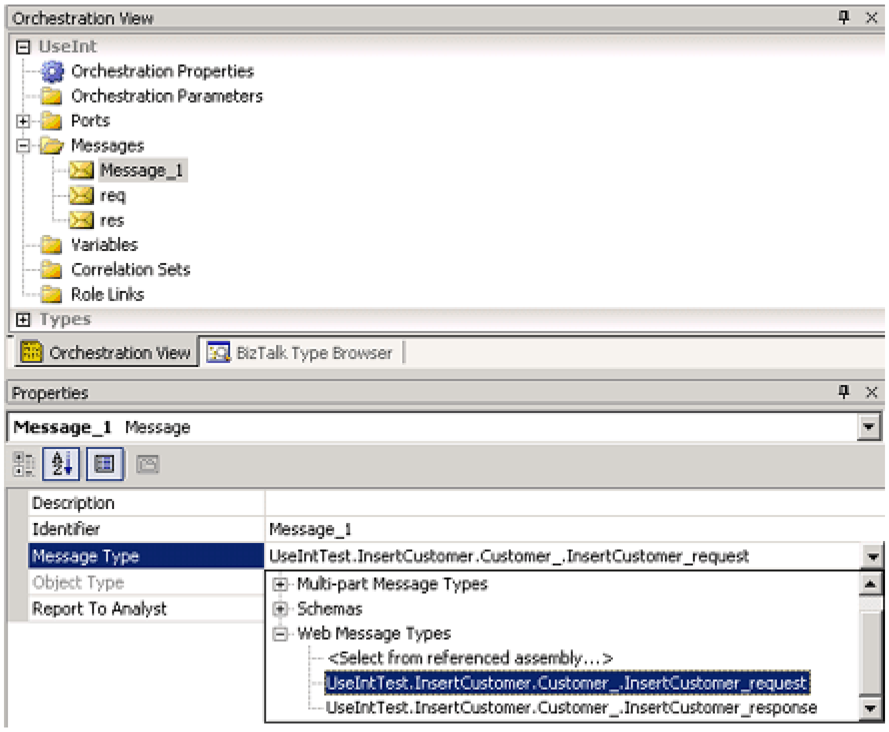
3. Message_*x* appears under the **Message** node.

4. In the Properties window, from the **Message Type** drop-down box, select a Web message type.

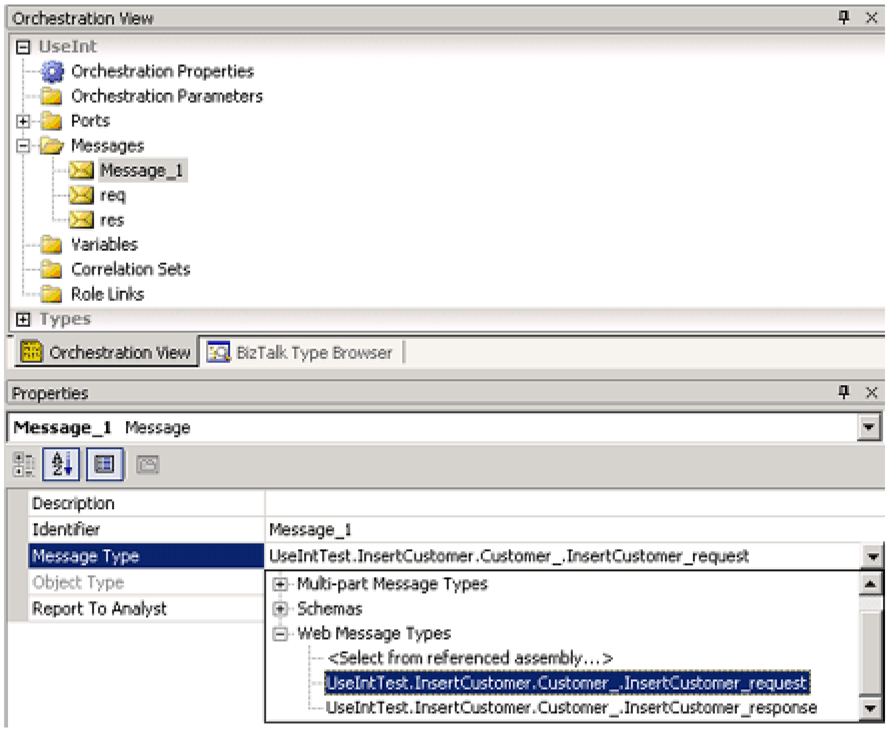
5. Scroll through the drop-down box to the Web message type you want to use, and then click the Web messsage type to select it.

The following figure shows the Orchestration View window with Message_1 selected and the Properties window, selecting a Web message type for Message_1.

Figure showing the Orchestration View window with Message_1 selected and the Properties window, selecting a Web message type for Message_1.



Description	
Identifier	Message_1
Message Type	UseIntTest.InsertCustomer.Customer_.InsertCustomer_request
Object Type	
Report To Analyst	



How to Determine a Web Message Part Type

You can determine if a Web message part type is a primitive .NET type or a schema type by using the Properties window for a given Web message type.

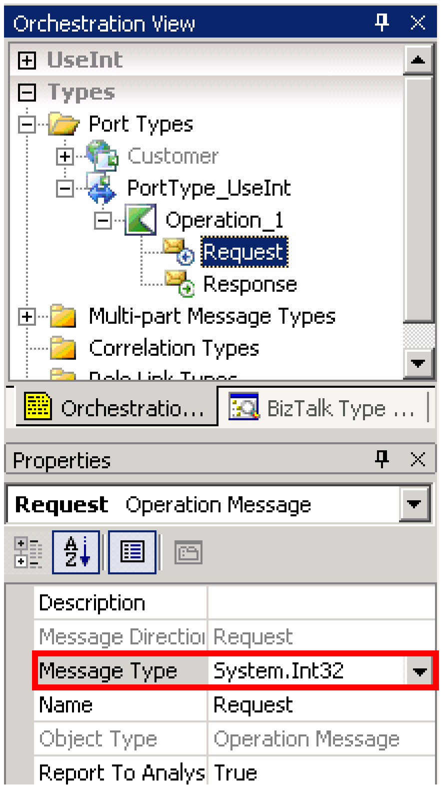
To determine a Web message part type

1. With an orchestration open, on the **View** menu, click **Other Windows**,and then click **Orchestration View**.
2. Expand the **Multi-part Message Types** node, and then expand a Web message type.
3. Select a message part.

A Web message part signature that begins as **<project default namespace>.<Web reference name>.Reference.<schema root>** is a schema type. The **<schema root>** part of the type is the root element of the Web reference schema that constructs the Web message part. Otherwise, the message part signature is a primitive .NET type such as **System.String** or **System.Int32**.

The following figure shows a primitive .NET type for the Web message type.

Figure showing a primitive .NET type for the Web message type



Request Operation Message

Description	
Message Directioi	Request
Message Type	System.Int32
Name	Request
Object Type	Operation Message
Report To Analys	True

How to Construct a Web Message Part from a Primitive .NET Type

You create a Web message part from a primitive .NET type by using a **Message Assignment** shape. Alternatively, you can create a Web message part from a primitive .NET type by using a .NET helper class to set the parts. For more information on creating message types by using a .NET class, see Constructing Messages in User Code .

To construct a Web message part from a primitive .NET type

1. With an orchestration open, open the **Toolbox** and click the **BizTalk Orchestrations** tab.

2. Drag a **Construct Message** shape to the orchestration.

3. Edit the **Message Constructed** property to include the message instance that you created for the Web message type.

4. Drag a **Message Assignment** shape onto the **Construct Message** shape.

5. Double-click the **Message Assignment** shape to open the BizTalk Expression Editor.

6. Set the parts of the Web message type to their required values in the BizTalk Expression Editor box.

   For example, the following code sets the expression to a string:

How to Construct a Web Message Part from a Schema Type

You create a Web message part from a schema type by using a **Transform** shape. Alternatively, you can create a Web message part from a schema type by using a .NET helper class to set the parts. For more information on creating message types by using a .NET class, see Constructing Messages in User Code .

To construct a Web message part from a schema type

- Add a new map. For information about creating maps, see How to Create New Maps .

- In BizTalk Mapper, click **Open Destination Schema** in the **Destination Schema** pane of the map and in the **BizTalk Type Picker** dialog box, expand the **Schemas** node, select the schema for the added Web reference, and then click **OK**.

  Note The format of the Web reference schema is **<project default namespace>.<Web reference name>.Reference**.

- In the **Root Node for Target Schema** dialog box, select a root node for the destination schema, and then click **OK**. For more information about how to determine a root node for a Web message part type, see How to Determine a Web Message Part Type .

- Click **Open Source Schema** in the **Source Schema** pane of the map and in the **BizTalk Type Picker** dialog box, expand the **Schemas** node, select the source schema to map data from, and then click **OK.**

- Open an existing orchestration (or create a new orchestration), open the **Toolbox**, and click the **BizTalk Orchestrations** tab.

- Drag a **Construct Message** shape to the orchestration.

- Edit the **Message Constructed** property to include the message instance that yo created for the Web message type.

- Drag a **Transform** shape onto the **Construct Message** shape and double-click to open the **Transform Configuration** dialog box.

- Click the **Existing Map** button and selec the map that you created in step one in the **Fully Qualified Map Name** list box.
- In the **Transform** pane, select **Source**. In the **Source Transform** pane, select a valid message instance from the list box.
- In the **Transform** pane, select **Destination**. In the **Destination Transform** pane, select the Web message instance from the list box, and then click **OK**.

For more information about using the **Transform Configuration** dialog box, see How to Use the Transform Configuration Dialog Box .

You can also use this procedure to map the Web method response message instance to another Web message instance.

How to Construct a Web Message with No Message Parts

A Web message that does not have message parts is a method that does not have input parameters.

To construct a Web message that does not have message parts

1. With an orchestration open, open the **Toolbox** and click the **BizTalk Orchestrations** tab.
2. Add a **Construct Message** shape.
3. Select the message for the **Construct Message** shape.

Creating Web Ports

Web ports are specially configured ports that you use to consume (call) Web services. A Web port can contain multiple operations that represent a mix of one-way (request only) and two-way (request-response) Web methods. Each operation in a Web port represents one method of a Web service.

You must add a Web reference before you can create a Web port.

In This Section

1. How to Add a Web Port
2. How to Change the URI of a Consumed Web Service
3. How to Dynamically Set the URI of a Consumed Web Service

How to Add a Web Port

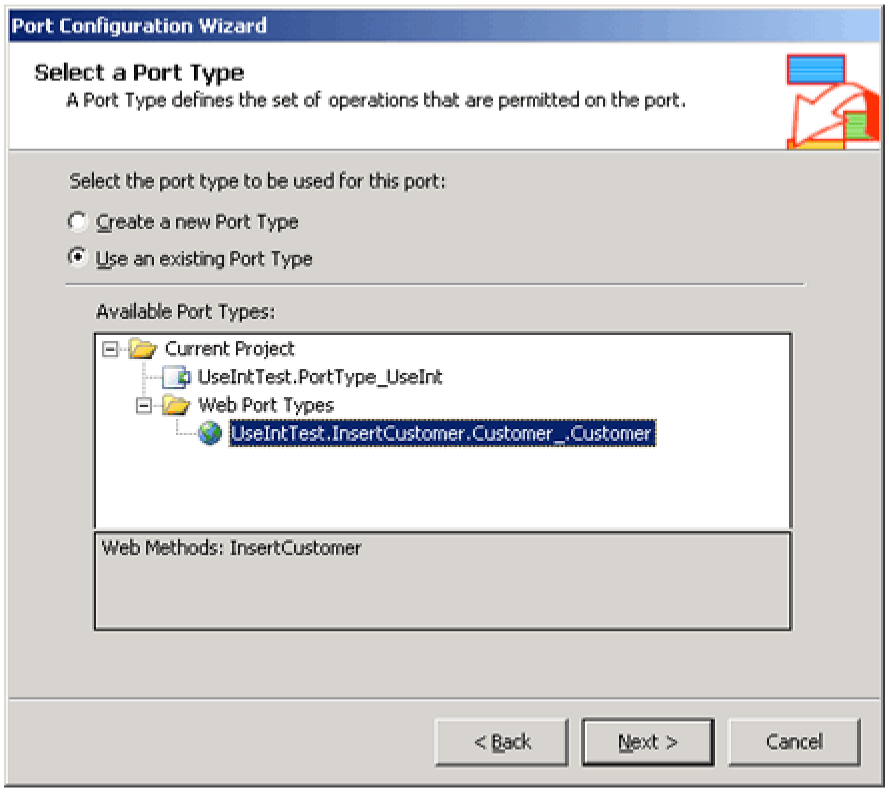
You add a Web port on the port surface in Orchestration Designer. Unlike other configured ports, Web ports support a mixture of request (one-way) and request/response (two-way) operations. Each operation in the Web port represents a Web method. If the Web method contains *input* and *output* parameters, BizTalk creates a request/response operation. If the Web service contains only an *input* parameter, BizTalk only creates a one-way operation.

To add a Web port

1. In Orchestration Designer, with an orchestration open, right-click the port surface, and then select **New Configured Port**.
2. In the **Name** text box, type a name for the port, and then click **Next**.
3. In the **Select a Port Type** page, select **Use an existing Port Type**.
4. In the **Available Port Types** box, expand the **Web Port Types** node and select the Web port type that corresponds to the added Web service, and then click **Next**.

The following figure shows the Select a Port Type dialog box.

Figure showing the Select a Port Type page



Select a Port Type

A Port Type defines the set of operations that are permitted on the port.

Select the port type to be used for this port:

( ) Create a new Port Type
[✓ Use an existing Port Type]

Available Port Types:

- Current Project
  - UseIntTest.PortType_UseInt
  - Web Port Types
    - UseIntTest.InsertCustomer.Customer_.Customer

Web Methods: InsertCustomer

1. In the **Port Binding** page, in the **Port binding** drop down box, select **Specify now**. BizTalk automatically places the values from the referenced Web service in the URI, transport, and receive and send pipelines. BizTalk uses these values to create the send port when you deploy your BizTalk project.

2. Click **Next**, and then **Finish** to complete the wizard

How to Change the URI of a Consumed Web Service

After you deploy your orchestration, BizTalk Server configures a send port for each Web service that the orchestration references. By default, BizTalk uses the URL of the Web service at run time for the same URL for the imported Web service. You can change this URL using BizTalk Explorer.

To change the URI of a consumed Web service

1. In Visual Studio .NET 2003, on the **View** menu, click **BizTalk Explorer**.

2. In BizTalk Explorer, expand the **Send Ports** node.

3. Right-click the send port that BizTalk created when you deployed your orchestration, and then click **Edit**.

If you do not have physical ports, you can create send ports and receive locations by using BizTalk Explorer. For information, see How to Add a Static Send Port Using BizTalk Explorer and How to Add a Receive Port Using BizTalk Explorer .

4. Select **Address** (URI) from the property grid and then click the ellipsis () button to open the **SOAP Transport Properties** dialog box.

5. In the **SOAP Transport Properties** dialog box, in the **Web Service URL** box, type the full URL for the location of the Web service, and then click **OK**.

How to Dynamically Set the URI of a Consumed Web Service

When you create a Web port for a consumed Web service, you can select a dynamic port binding. When you select a dynamic port binding, you must set the URI of the consumed Web service at run time. The selected URI must call a Web service that has the same Web proxy as the Web service that you used to create the Web port type.

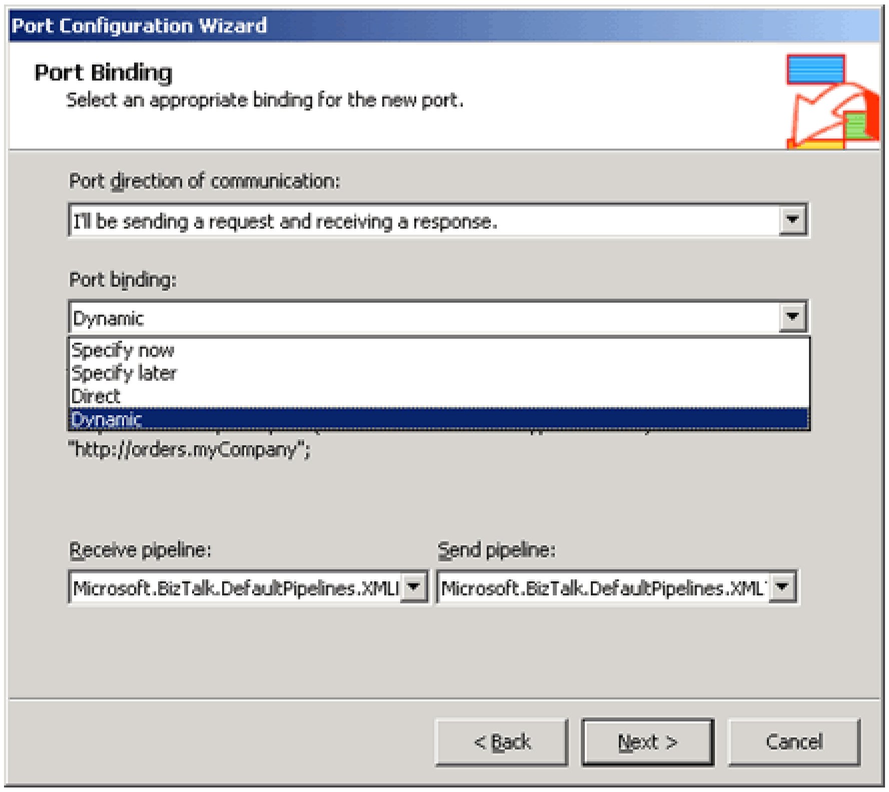
Dynamic port bindings for Web ports have different behavior than dynamic port binding for non-Web ports. When selecting dynamic bindings for a non-Web port, you cannot use the SOAP adapter. A dynamic binding of a Web port only enables you to dynamically change the URI of the Web service. You cannot dynamically change the transport type or the proxy of the Web service.

When using dynamic Web ports to consume a Web service, the send port properties are set to the default values. Some of these values are set internally and other values default to the values that are set in the **SOAP Adapter Handler** property pages. You can overwrite these values when you use dynamic send ports.

To dynamically change the URI of a consumed Web service

- Add a Web port as outlined in Adding a Web Port. However, instead of selecting the **Specify now** port binding, select **Dynamic** port binding, as shown in the following figure.

Figure showing the Port Binding page.


Port Configuration Wizard
Port Binding
Select an appropriate binding for the new port.
Port direction of communication:
I'll be sending a request and receiving a response.
Port binding:
Dynamic
Specify now
Specify later
Direct
Dynamic
"http://orders.myCompany";
Receive pipeline:
Microsoft.BizTalk.DefaultPipelines.XMLI
Send pipeline:
Microsoft.BizTalk.DefaultPipelines.XML
< Back
Next >
Cancel

- In the orchestration that calls the consumed Web service, add an **Expression** shape at some point prior to the **Send** shape that you have connected to the Web port.
- In the **Expression** shape, add an expression similar to:

To dynamically modify send port properties

1. In the **Construct Message** shape that you use to construct the Web message, add a **Message Assignment** shape if one is not already present.
2. In the **Message Assignment** shape, add an expression similar to:

All of the properties for the SOAP send port use the SOAP namespace.

The following table contains a list of the SOAP send port properties that you can set when using dynamic Web ports.

Property Name	Type	Description
AuthenticationScheme	String	Authentication method to use for calling the Web service Default value: Anonymous Other allowed values: Basic, Digest, NTLM
Username	String	User name to specify for accessing the target Web service. Default value: Blank
Password	String	User password to use for authentication with the server. Default value: Blank
ClientCertificate	String	Thumbprint of client Secure Sockets Layer (SSL) certificate. Default value: Blank
UseSSO	Boolean	Indicates whether this Web port will use Single Sign-On (SSO). Default value: False
AffiliateApplicationName	String	The name of the SSO application that this Web port will use to redeem the ticket for client credentials. Default value: Blank
UseHandlerSetting	Boolean	Indicates whether this Web port will use SOAP send handler HTTP proxy settings. Default value: False
UseProxy	Boolean	Indicates whether this Web port will use a proxy server to access the target Web service. Default value: False
ProxyAddress	String	Address of the HTTP proxy to use for the Web service call. Default value: Retrieved from SOAP send

		handler properties.
ProxyPort	Integer	Port of the HTTP proxy to use for the Web service call. Default value: Retrieved from SOAP send handler properties.
ProxyUsername	String	User name to use for the HTTP proxy. Default value: Retrieved from SOAP send handler properties.
ProxyPassword	String	Password to use for the HTTP proxy. Default value: Retrieved from SOAP send handler properties.
ClientConnectionTimeout	Int32	Time-out value for HTTP client connection. Default value: Same as default ASP.NET HTTP connection time-out.

Considerations When Consuming Web Services

This section provides information that you should taken into consideration when consuming Web services.

Using two underscore characters in a parameter name

A parameter name for a Web method cannot begin with "__" (two underscore characters). Names that begin with two underscore characters may create Web message parts that are unsupported (unusable) by XLANG/s.

Using the any element and the anyAttribute attribute

You cannot use the **any** element or **anyAttribute** attribute in the schema for a Web method.

Using XLANG/s keywords

A Web service name or a Web method name cannot be a keyword in an XLANG/s. If you use an XLANG/s keyword in the Web service name or Web method name, you will get a compiler error when you add the Web service. For a list of reserved words for the XLANG/s language, see **XLANG/s Reserved Words**.

Required XLANG/s support for parameter types

You will get a compilation error if you use a non-XLANG/s supported Web method parameter type. For example, BizTalk does not support a parameter that consists of a single dimensional array of schema types. In addition, BizTalk does not support any multi-dimensional arrays. For a list of words that XLANG/s language reserves in BizTalk, see **XLANG/s Reserved Words**.

Avoiding compilation errors caused by adding Web references containing C# keywords or identifiers

When you use the **Add Web Reference** to add Web references to BizTalk projects, BizTalk converts the schema types that are required to call each Web method to schemas. BizTalk adds these schemas to Reference.xsd. If your schemas contain element names that are C# keywords or the element name is not valid as a C# identifier, you may get a run-time error:

To avoid run-time errors, ensure that the Web service you consume does not contain element names that are C# keywords or not valid C# identifiers.

Multiple service/port type definitions are unsupported

BizTalk Server supports adding a Web service file with a single service and port type definition. If you add a WSDL file with multiple service or port type definitions, you may get the following error.

⊟Support of consuming arrays exposed by Web services

BizTalk Server can consume arrays exposed by web services that are not BizTalk web services. The syntax for doing this is shown below:

Web method parameters must be Xml Serializable

All parameters in a consumed Web service must be Xml Serializable. If you add a Web method that contains a parameter that is not Xml Serializable, you may receive the following error message:

Messaging-only consumed Web services

Messaging-only consumption will only work in the following case:

Setting send port properties should only be done for dynamic send ports

The send port properties **AssemblyName**, **TypeName** and **MethodName** specify the web service proxy to invoke for web service consumption and can be overwritten by a custom component in the send pipeline. However, this should only be done when using dynamic send ports. For more information on custom pipeline components, see Creating Custom Pipeline Components .

Adding a web reference to a consumed web service that contains a multi-rooted schema will cause a compilation error

If you add a web reference to your project for a web service that was derived from a published BizTalk orchestration and the orchestration contains a schema with multiple roots then an error will occur when the project is compiled. Ensure that if you add a web reference to your project that was derived from a published BizTalk orchestration that the orchestration does not contain any multi-rooted schemas.

Publishing Web Services

Publishing Web services enables you to create a Web service that can submit messages to Microsoft® BizTalk® Server for use by orchestrations and other send adapters. You create published Web services using the BizTalk Web Services Publishing Wizard. For information about configuring a SOAP send handler, see Configuring a SOAP Send Handler. For information about configuring a SOAP send port, see **Configuring a SOAP Send Port**.

The BizTalk Web Services Publishing Wizard provides you with two methods to publish Web services: publishing an orchestration as a Web service and publishing schemas as a Web service.

You must install and enable Microsoft® Internet Information Services (IIS) and ASP.NET before you can publish Microsoft BizTalk® Server orchestrations and schemas as Web services. For information about installing IIS and ASP.NET, see Installing Internet Information Services (IIS) in the BizTalk Server Installation Guide located at http://go.microsoft.com/fwlink/?linkid=22120.

Important Before running the BizTalk Web Services Publishing Wizard, you must enable Web services. For more information about enabling Web services for your system, see Enabling Web Services.

In This Section

- Planning for Publishing Web Services
- Enabling Web Services
- Publishing an Orchestration as a Web Service
- Publishing Schemas as a Web Service
- Considerations When Publishing Web Services
- Debugging Published Web Services
- Testing Published Web Services

- Deploying Published Web Services on a Non-Visual Studio .NET Computer

Planning for Publishing Web Services

You can access Web services from your orchestrations. You can also use the BizTalk Web Services Publishing Wizard to publish your Web service.

The following table lists some of the questions that you need to answer in planning for Web services.

Planning question	**Recommendation**
Have you built your BizTalk project?	You must build BizTalk projects prior to running the BizTalk Web Services Publishing Wizard.
Have you enabled your system to run Web services?	You must enable your system before running the BizTalk Web Services Publishing Wizard. For more information about enabling your system for Web services, see **Enabling Web Services2**.
Have you verified that your schema contains only valid XML characters and elements?	Web services do not support Chinese/Japanese/Korean (CJK) Unified Ideograph Extension A characters. There are also restrictions on certain XML Schema (XSD) elements. For more information about valid XML characters and supported elements and considerations for elements, see Considerations When Publishing Web Services .
Do any of the message types in your BizTalk project use user-defined .NET classes?	You must install assemblies containing user-defined .NET classes that message types reference in the global assembly cache (GAC).
Do your Web clients use the supplied credentials for the **WindowsUser** context property?	The credentials supplied by the Web clients consuming (calling) a published Web services use the **WindowsUser** context property. The Party Resolution uses this property. If you set up a Party using the **WindowsUser** context property and the Web client consumes (calls) the Web service with credentials that match the Party, BizTalk identifies the message as coming from the corresponding predefined party. For more information about party resolution with pipeline components, see Party Resolution Pipeline Component .

The BizTalk Server Web Services Publishing Wizard relies on functionality provided with ASP.NET 2.0 from the .NET Framework 2.0. If you have upgraded your BizTalk Server installation from BizTalk 2004, you will need to update any IIS virtual directories that are used in conjunction with the BizTalk Server 2006 Web Services Publishing Wizard to use ASP.NET 2.0 or later. To update a virtual directory to use ASP.NET 2.0, complete the following steps:

1. Launch the Internet Information Services (IIS) Manager. Click **Start**, click **Programs**, click **Administrative Tools**, and click **Internet Information Services (IIS) Manager**.

2. If you need to connect to a remote IIS Server, right click the Internet Information Services node and click the **Connect...** option.

3. Type in the computer name for the remote IIS Server and credentials if necessary.

4. Click to expand the server name that houses the web site or virtual directory to be updated.

5. Click to expand **Web Sites**.

6. Click to expand the Web Site to view the virtual directories under the web site.

7. Right-click the virtual directory that you want to update to use ASP.NET 2.0 and select **Properties** to display the virtual directory properties dialog.

8. Click the **ASP.NET** tab of the virtual directory properties dialog.

9. Click the dropdown option next to **ASP.NET version:** and change to 2.0 or later and click the **OK** button to apply changes.

Enabling Web Services

To publish Web services, you must configure Internet Information Services (IIS), BizTalk Isolated Hosts, and Windows user and group accounts. This section discusses how to enable Web services on Windows 2000 Server, Windows XP, and Windows Server 2003. For more information about enabling Web services, see the IIS documentation.

Internet Information Services 5.0, 5.1, and 6.0

You can publish Web services to Windows systems that have IIS 5.0, 5.1, or 6.0 installed. IIS 5.0 (for Windows 2000 Server) and IIS 5.1 (for Windows XP) use the ASP.NET worker process for processing Web service requests. For each server, all Web services run within the ASP.NET worker process.

The ASP.NET worker process uses the local ASPNET account by default. IIS 6.0 (for Windows Server 2003) uses the IIS application pools for processing Web service requests. IIS 6.0 supports multiple application pools. Each application pool process can run under a different user context. The ASPNET user account only applies to Windows 2000 Server and Windows XP.

BizTalk Isolated Hosts

To enable Web services, you must create at least one Isolated Host in BizTalk Server. Isolated Hosts represent external processes, such as ISAPI extensions and ASP.NET processes that BizTalk Server does not create or control. These types of external processes must host certain adapters, such as HTTP/S and SOAP.

The Microsoft BizTalk Server 2006 Configuration Wizard created the BizTalkServerIsolatedHost that BizTalk uses as the default Isolated Host. The BizTalk Isolated Host Users group is the default Windows group. For more information about Hosts and Host Instances, see Managing BizTalk Hosts and Host Instances.

Database access for single server installations

If BizTalk Server 2006 and the BizTalk Management database reside on the same server, you should set the user context of the ASP.NET worker process or the IIS Application Pool to the local ASPNET user account, a local, or domain user account that has minimal privileges.

Database access for multiple server installations

If BizTalk Server 2006 and the BizTalk Management database reside on different servers, you should change the user context of the ASP.NET worker process or the IIS Application Pool to a domain user account. Using the ASPNET or another local user account on different servers requires the servers synchronize the local accounts. Yu should not set up your system to maintain synchronized local accounts since this is a manual process and is difficult to maintain.

When implementing a multi-server deployment, the same Isolated Host Windows groups must exist on any server that runs the Isolated Host and the server containing BizTalk MessageBox database. You can achieve this by using domain groups. If you use local group, you must use identical Windows group names that contain the same user accounts.

Minimizing account privileges and user rights

You use Isolated Hosts to give adapters that run in external processes access to the minimal amount of required resources to interact with BizTalk Server. For a secure deployment, you should give the user context for the external processes minimal privileges.

Security recommendations for BizTalk Web Services Publishing Wizard

The virtual directory created by the BizTalk Web Services Publishing Wizard will inherit the access control lists (ACL) and authentication requirements from the parent virtual directory or Web site. If the parent virtual directory or Web site allows anonymous access, the BizTalk Web Services Publishing Wizard will remove that capability when creating the virtual directory.

The following contains security notes and recommendations for Windows 2000 Server, Windows XP, and Windows Server 2003.

In This Section

1. How to Enable Web Services for Windows 2000 Server and Windows XP

2. How to Enable Web Services for Windows Server 2003

How to Enable Web Services for Windows 2000 Server and Windows XP

In Windows 2000 Server and Windows XP, Web services run in the ASP.NET worker process (aspnet_wp.exe). This process runs under the ASPNET user context by default. You can use this default account or you can change the process to run under the local or domain user account. You must add the account to the Windows group that you have configured in the Isolated Host that represents this adapter.

By default, the local ASPNET user account is included in the local users group. You should remove the ASPNET account from the local users group. If you are using a different local or domain account, you should only add the user to the BizTalk Isolated Host User group. Do not add the user to any other groups (local or domain).

You should create the user context that the ASP.NET worker process runs under with minimal privileges. For more information about minimal settings required for this user, see the MSDN article "How To: Create a Custom Account to Run ASP.NET" at http://go.microsoft.com/FWLink/?LinkID=16725.

Changing the user context of ASP.NET

To change the user context of ASP.NET, you must manually edit the machine.config file. By default, the machine.config file is located at systemroot\Microsoft.NET\Framework\version number\CONFIG. By default, the **processModel** element has **username** set to **machine** and **password** set to **AutoGenerate**. You can update the **username** and **password** attributes to contain a new **username** and **password**. This access method requires saving the **username** and **password** as clear text in the machine.config file.

The following example shows the default settings for the machine.config file:

The following example shows a possible setting for the modified machine.config file:

For a more secure access method, you should use the ASP.NET Set Registry console application to encrypt and store these credentials in the registry.

For more information about configuring the ASP.NET process model settings, see "<processModel> Element" in the .NET Framework SDK documentation at http://go.microsoft.com/fwlink/?LinkId=25657.

For more information about running aspnet_wp.exe, see "ASP.NET Debugging: System Requirements" in the .NET Framework SDK documentation at http://go.microsoft.com/fwlink/?LinkId=25658.

Storing the ASP.NET worker process username and password in the registry

The **username** and **password** attributes are stored in clear text in the configuration file. Although Internet Information Services (IIS) does not transmit configuration files in response to a user agent request, IIS can read configuration files in other ways. For example, an authenticated user with proper credentials on the domain that contains the server is able to read the configuration file. For increased security, the **processModel** element supports storage of encrypted **username** and **password** attributes in the registry. The credentials must be in REG_BINARY format encrypted by the Windows 2000 Server and Windows XP Data Protection API (DPAPI) encryption functions.

For more information about storing a user name and password, see "<processModel> Element" in the .NET Framework SDK documentation at http://go.microsoft.com/fwlink/?LinkId=25657.

To verify the BizTalk Isolated Host installation

1. In the BizTalk Administration Console, expand **Hosts**.
2. Verify that your computer has installed an isolated. The BizTalkServerIsolatedHost is the default isolated host name.

If an isolated host does not exist (not installed), you must install an isolated host.

To verify the BizTalk Isolated Host Windows group

1. In the BizTalk Administration Console, right-click the isolated host.

The Windows group appears in the list in the Windows group box.

To change the user context of ASP.NET to a different local or domain account

1. Open the machine.config file.
2. Modify the attributes of the **processModel** element to reflect a user name (domain accounts must include the domain name and "\") and password.
3. Save and close the machine.config file.

How to Enable Web Services for Windows Server 2003

Internet Information Services (IIS) 6.0 in Windows Server 2003 allows multiple application pools to run simultaneously. Each application pool can run under a different user context. For more information regarding IIS Manager in Windows Server 2003, see the IIS documentation in the Windows Server 2003 documentation set.

The BizTalk Web Services Publishing Wizard automatically adds the application pool set for the parent Web site for the Web services that you publish. The default setting is DefaultAppPool. By default, this application pool runs under the Network Service security account. For a complete description of the Network Service security account and alternative accounts that you can use as the identity for the application pool, see the IIS documentation.

You should create a new application pool to contain all your published Web services. Each time you create a Web service using the BizTalk Web Services Publishing Wizard, you must change the application pool to the previously added application pool.

To add a new application pool in IIS Manager

1. In IIS Manager, right-click **Application Pool**, click **New**, and then click **Application Pool**.
2. In the **Add New Application Pool** dialog box, in **Application pool ID** text box, type a name for your application pool ID.
3. In the **Application pool settings** group, select **Use default settings for new application pool**, and then click **OK**.

To change the new application pool to run under a new identity

- In IIS Manager, right-click the application pool, and then click **Properties**.
- Click the **Identity** tab.
- In the **Application pool identity** group, select **Configurable** and in the **User name** text box, type a low-privilege local or domain account.
- In the **Password** text box, type a password and click **OK**.
- In the **Confirm Password** dialog box, reenter password and click **OK**.
- Add the user to the BizTalk Isolated Host Users group.

To move a Web service to a new application pool

1. In IIS Manager, edit the properties for the virtual directory that the BizTalk Web Services Publishing Wizard created.

2. Select the added application pool from the **Application Pool** drop-down list, and then click **OK**.

Publishing an Orchestration as a Web Service

You use the BizTalk Web Services Publishing Wizard to publish an orchestration as a Web service. The wizard creates a Web service based on an orchestration in a BizTalk assembly. Using the wizard, you select orchestrations and receive ports to publish Web services. You can define target namespaces, SOAP header requirements, and locations for the Web service project the wizard generates.

Before you run the wizard, you must compile your BizTalk assemblies that contain the orchestrations and schemas that you intend to publish as a Web service.

Prior to running the BizTalk Web Services Publishing Wizard, you must enable Web services. For more information about enabling Web services for your system, see Enabling Web Services.

In This Section

1. How to Map Orchestrations to Web Services

2. How to Use the BizTalk Web Services Publishing Wizard to Publish an Orchestration as a Web Service

3. How to Throw SOAP Exceptions from Orchestrations Published as a Web Service

How to Map Orchestrations to Web Services

An orchestration can have multiple receive ports. Using the BizTalk Web Services Publishing Wizard, you select receive ports to publish as Web services. For each receive port, the wizard creates one Web service (.asmx file). Operations become function calls. Each operation in the receive port becomes a Web method. Request operations become input parameters. Response operations become return types.

If the request and response operations are the same Web message type, the input parameter becomes a **byres** and the return type is **void**. The operation message types define the Web method signatures. Each message type part is a parameter in the Web method.

Note ASP.NET Web clients may change the Web method signature by combining the in and out parameters of the same type. For example, an ASP.NET Web client may change a BizTalk Web method from **string myService(string part)** to **void myService(ref string part)**.

Message type part names and target namespaces

Message type parts that have defined target namespaces include document schemas and user defined classes with **XmlRootAttribute** specified. Message type parts without defined target namespaces include EDI schemas, user-defined classes without **XmlRootAttribute** specified, and built-in types, such as **System.String**.

The following table shows the parameter used for a given target namespace.

If the message type part name has a	**Parameter Name Used**
Defined target namespace	Root element name
No defined target namespace	Message type part name

Orchestrations with multiple operations

If your orchestration has multiple operations, you should design your orchestrations to have one receive port instead of several receive ports. This design prevents the BizTalk Web Services Publishing Wizard from creating multiple Web service (.asmx) files and only works when all the operations have the same calling pattern—all one-way operations or all request-response operations. A single receive port cannot contain both one-way and request/response operations.

Web services naming conventions for published orchestrations

The BizTalk Web Services Publishing Wizard generates Web service (.asmx) file names based on the orchestrations namespace, followed by an underscore (_), followed by the type name, followed by an underscore (_), and followed by the name of the receive port. An underscore (_) replaces any of the parts that contain periods. The name of the Web service always has the port name appended.

The following table shows how the BizTalk Web Services Publishing Wizard generates Web service names.

Orchestration(s) with Web port(s)	**Generated Web service name**
One orchestration with one Web port	orchestration1_port1.asmx
One orchestration with two Web ports	orchestration1_port1.asmx and orchestration1_port2.asmx
Two orchestrations with one Web	orchestration1_port1.asmx and

port each	orchestration2_port2.asmx

How to Use the BizTalk Web Services Publishing Wizard to Publish an Orchestration as a Web Service

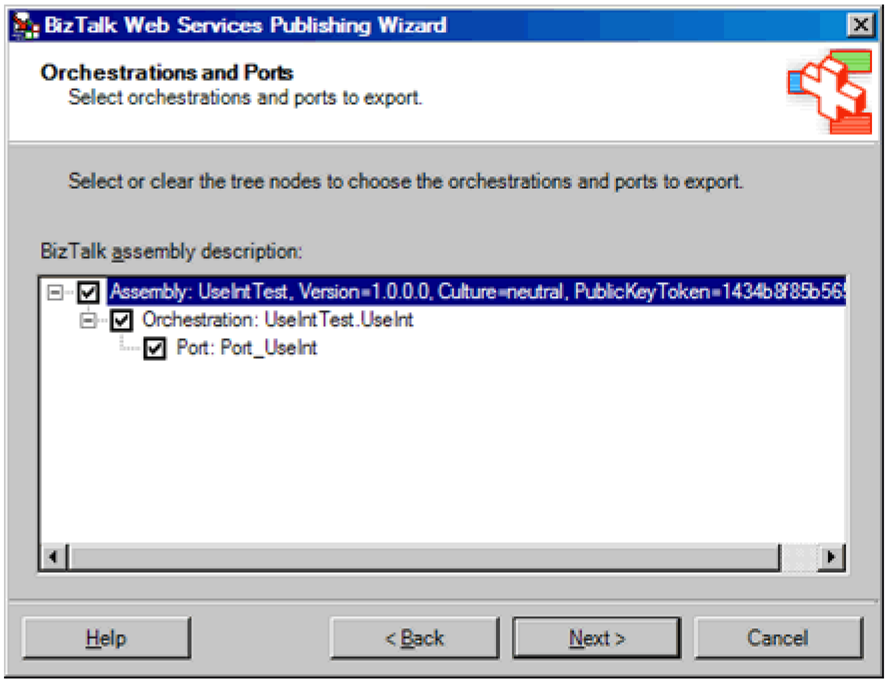
You use the BizTalk Web Services Publishing Wizard to publish an orchestration as a Web service.

To publish an orchestration as a Web service

1. Click **Start**, point to **All Programs**, point to **Microsoft BizTalk 2006**, and then click **BizTalk Web Services Publishing Wizard**.
2. On the **Welcome to the BizTalk Web Services Publishing Wizard** page, click **Next**.
3. On the **Create Web Service** page, select **Publish BizTalk orchestrations as web services**, and then click **Next**.
4. On the **BizTalk Assembly** page, in the BizTalk assembly file (*.dll) text box, type the name of the BizTalk assembly file or click **Browse** to browse to the assembly containing the orchestration(s) to publish, and then click **Next**.
5. On the **Orchestrations and Ports** page, expand the tree nodes for each assembly and orchestration by clicking the plus sign. Select orchestrations and ports to publish by checking the corresponding tree node check boxes, and then click **Next**.

The following figure shows the BizTalk Web Services Publishing Wizard Orchestrations and Ports page with the tree nodes for the assembly expanded.

Figure showing the BizTalk Web Services Publishing Wizard Orchestrations and Ports page.



Orchestrations and Ports
Select orchestrations and ports to export.

Select or clear the tree nodes to choose the orchestrations and ports to export.

BizTalk assembly description:

Assembly: UseIntTest, Version=1.0.0.0, Culture=neutral, PublicKeyToken=1434b8f85b56
Orchestration: UseIntTest.UseInt
Port: Port_UseInt

- On the **Web Service Properties** page, in the **Target namespace of the web service** box, type a target namespace for the Web service, select the appropriate boxes to specify how the wizard should handle SOAP headers and SharePoint Portal Server 2003 Single Sign-On (SSO) support for the Web service, and then click **Next**.

- If you selected **Add additional SOAP headers** option, the **Request SOAP Headers** and **Response SOAP Headers** pages appear. You can add and remove request and response SOAP headers using the **Add** and **Remove** buttons in the following dialog boxes:

  - To add a SOAP header, click **Add**. In the **BizTalk assembly file (*.dll)** text box, type or browse for the assembly containing the SOAP header schema. The **Available schema types** list view displays each root element of the schema. Select a root node to add as a request or response SOAP header. To select multiple items, hold the CTRL key and click **OK**.

  - To remove a SOAP header from the list, select it from the list of added SOAP headers, and then click **Remove**.

  - Click **Next** on each **SOAP Header** page to continue the wizard.

- On the **Web Service Project** page, in the **Project name** text box, type the name for the project. You can accept the default location (http://localhost/<*project_name*>), type a location for the project in the **Project location** text box, or click **Browse** and select a Web directory. Select any of the following options:

- **Overwrite existing project.** This option is only available if the project location already exists. You will only be able to publish to the same location if you select this option. Otherwise, you must enter a different project location.
- **Allow anonymous access to web service.** This option adds anonymous access to the created virtual directory. By default, the virtual directory inherits the access privileges from its parent virtual directory or the Web site (if it is a top-level virtual directory).
- **Create BizTalk receive locations.** This option automatically creates the SOAP adapter receive ports and locations that correspond to each generated .asmx file. If a receive location already exists, it is not replaced. Receive locations for the SOAP adapter are resolved using the format /*<virtual directory name>*/*<orchestration namespace_typename_portname>*.asmx.

- Click **Next** to review your settings for the ASP.NET Web service project.
- Click **Create** to create the ASP.NET Web service.
- Click **Finish** to complete the BizTalk Web Services Publishing Wizard.

How to Throw SOAP Exceptions from Orchestrations Published as a Web Service

You can return a SOAP exception from an orchestration that you have published as a Web service. You add a fault message to your SOAP port and send the fault message instead of the response.

To throw a SOAP exception from an orchestration published as a Web service

1. Add a fault message to the SOAP port type. The message type for the fault message can be any XML Schema (XSD) compliant schema or simple type.

   Note To return a string as a **SoapException** with error information, you can use the simple type string as the fault message type.

2. In your orchestration, create the fault message.
3. Use the **Send** shape to link to the fault operation in the SOAP port that corresponds to the fault message. A SOAP exception wraps the returned fault message.

If your orchestration does not return an error, use a different **Send** shape to send the standard SOAP response message using the usual response operation.

Publishing Schemas as a Web Service

You use the BizTalk Web Services Publishing Wizard to create Web services that use existing schemas. You declare the Web services, Web methods, and request and response schemas that you want to publish. Using the wizard, you define the target namespace, SOAP header requirements, and the location of the generated Web service project.

In This Section

- Creating Web Services and Web Methods
- Using the BizTalk Web Services Publishing Wizard to Publish Schemas as a Web Service

Creating Web Services and Web Methods

When you publish schemas as a Web service, you control the creation of Web services and Web methods in the BizTalk Web Services Publishing Wizard. You can rename the Web service description, Web service and Web method inside the tree available on the Web Services page. You can add and remove Web services and Web methods. The wizard uses the root element names of the selected request schemas as the input parameter name. The Web method return value returns the response schema.

Web services naming conventions for published orchestrations

The BizTalk Web Services Publishing Wizard generates Web service (.asmx) file names based on the Web services description that you define in the Web Service page. The following table shows the default values for the Web service file names.

Generated Web service	File name
BizTalkWebService	Visual Studio .NET Web Service project name
WebService1	Web service (.asmx) file name
WebMethod1	Web method name

The generated Web service does not reflect the names of the remaining nodes.

How to Use the BizTalk Web Services Publishing Wizard to Publish Schemas as a Web Service

You use the BizTalk Web Services Publishing Wizard to publish schemas as a Web service.

To publish schemas as a Web service

1. Click **Start**, point to **All Programs**, point to **Microsoft BizTalk 2006**, and then click **BizTalk Web Services Publishing Wizard**.

2. On the **Welcome** page, click **Next**.

3. On the **Create Web Service** page, select **Publish schemas as web services** and then click **Next**.

4. On the **Web Service** page, define the Web service(s) to publish. You use the tree in the **Web service description** dialog box to add, remove, rename, and edit the Web service description nodes. The **Information** dialog box provides information about the selected node and displays any errors in the current node or any sub nodes:

    1. The root node to the tree (Web service description) describes the Web service project name. The virtual directory name uses the root node as the default name. You can modify the Web service description by selecting **Rename web service description**.

    2. To add a new Web service, right-click the **Web service description** node, and then click **Add web service**. This creates a new Web service without any Web methods. To modify the name of the Web service, right-click the Web service node, and select **Rename web service**,and then press Enter to accept the new name.

    3. To add a new Web method, right-click the Web service node, point to **Add Web Method**, and then click **One-way** (for a Request Web method) or **Request-response** (for a Request-Response Web method) from the shortcut menu.

    4. To set the request and response schema types, right-click the **Request** or **Response** node, and then click **Select schema type**. In the **Request Message Type** dialog box, type the name of the assembly containing the SOAP Header schema in the **BizTalk assembly file** dialog box or click **Browse** to search for the assembly. The **Available schema types** list view displays each root element of the schema. Select a root node to add as the request or response schema type.

    5. You can rename the **Request** and **Response** nodes without affecting the generated code. After defining your schemas, you can rename the part elements, which modifies the Web method parameter name. You can see the changes by viewing the generated Web service code.

5. Click **Next** to continue the wizard.

6. On the **Web Service Properties** page, in the **Target namespace of Web service** dialog box, type a target namespace for the Web service, and select the

appropriate boxes to specify how the wizard should handle SOAP headers and Single Sign-On support for the Web service, and then click **Next**.

7. If you selected **Add additional SOAP headers**, the **Request SOAP Headers** and **Response SOAP Headers** pages appear. You can add and remove request and response SOAP headers using the **Add** and **Remove** buttons in the following dialog boxes:

    1. To add a SOAP header, click **Add**. In the **BizTalk assembly name (*.dll)** text box, type the assembly name or browse for the assembly containing the SOAP Header schema in the **BizTalk assembly file** text box. The **Available schema types** list view displays each root element of the schema. Select a root node to add as a request or response SOAP header. To select multiple items, hold the CTRL key and click **OK**.

    2. To remove a SOAP header from the list, select it from the list of added SOAP headers, and then click **Remove**.

    3. Click **Next** on each SOAP header page to continue the wizard.

9. On the **Web Service Project** page, in the **Project location** text box, type the project location. You can accept the default location (http://localhost/<*project_name*>), type a location for the project, or click **Browse** and select a Web directory. Select any of the following options:

    1. **Overwrite existing project.** This option is only available if the project location already exists. You will only be able to publish to the same location if you select this option. Otherwisse, you must enter a different project location.

    2. **Allow anonymous access to web service.** This option adds anonymous access to the created virtual directory. By default, the virtual directory inherits the access privileges from its parent virtual directory or the Web site (if it is a top-level virtual directory).

    3. **Create BizTalk receive locations.** This option automatically creates the Soap adapter receive ports and locations that correspond to each generated .asmx file. If another receive location already exists, the receive location is not be replaced. Receive locations for the SOAP adapter are resolved using the format "/<*virtual directory name*>/<*orchestration namespace_typename_portname*>.asmx".

10. Click **Next** to review your settings for the ASP.NET Web service project.

11. Click **Create** to create the ASP.NET Web service.

12. Click **Finish** to complete the BizTalk Web Services Publishing Wizard.

Considerations When Publishing Web Services

This section provides information that you should consider before you publish your Web services.

Publishing schemas and the include element

There are a few scenarios where schemas that contain the **include** element cannot be published as a Web service. An error will occur when you complete the BizTalk Web Services Publishing Wizard. These restrictions include the following:

- Circular includes (the included schema has an **include** element to the including schema)
- An unresolved **schemaLocation** attribute will cause an error

Publishing schemas and the redefine element

Published Web services do not support schemas that contain the **redefine** element.

Publishing schemas that specify values for minOccurs or maxOccurs attributes

If you publish a schema that contains minOccurs or maxOccurs attributes with specific values, these values may be different in schema exposed by the published web service. As a general rule of thumb, all minOccurs attributes are converted to 0 (minOccurs=0) and maxOccurs attributes are converted to either 1 or unbounded (maxOccurs=1 or maxOccurs=unbounded).

Publishing envelope schemas

If you have an envelop schema that you are publishing as a Web service, you need to manually modify the generated Web project.

To modify the generated Web project for envelope schemas

- Open the Web project at the <*myWebService*>.asmx.cs file.
- Edit the file and change `bodyTypeAssemblyQualifiedName = <dll.name.version.>` to `bodyTypeAssemblyQualifiedName = null`.
- Rebuild the Web project.

Web service and Web method attributes

The BizTalk Web Services Publishing Wizard does not allow you to customize the Web service or Web method attributes you use in ASP.NET. Some attributes are automatically set based upon information that the wizard provides. The wizard does not use the other attributes.

Modifying the existing attributes or adding new attributes to Web services that the BizTalk Web Devices Publishing Wizard generates may cause the Web service to function incorrectly.

For more information about Web services and Web method attributes, see the **WebServiceAttribute** and **WebMethodAttribute** classes in the .NET Framework SDK documentation.

Web method required

A Web service must have at least one Web method. Without at least one Web method, port types will not have their operations created. XLANG/s does not support port types that do not have operations.

DBCS character support

Web services do not support Chinese/Japanese/Korean (CJK) Unified Ideograph Extension A characters.

Republishing Web services using the BizTalk Web Services Publishing Wizard

You can use the BizTalk Web Services Publishing Wizard to republish a published Web service. On the **Web Service Project** page, you can select the **Overwrite Web Service** option.

The wizard does not store previously used settings. If you make changes in the settings when you rerun the wizard, any Web clients that consume (call) the published Web service may fail. You should update the Web references of any clients that consume (call) a republished Web service.

Clients of published web services may not receive Server script timeout errors

Web services generated with the Web Services Publishing Wizard in BizTalk Server 2006 are configured by default with a script timeout value of **110** seconds. This is the default value for the .NET Framework 2.0 **HttpServerUtility.ScriptTimeout** property. Web clients that use .NET Framework 2.0 are configured by default with a request timeout value of **100** seconds. This is the default value for the .NET Framework 2.0 **HttpWebRequest.Timeout** property.

If web clients that use .NET framework 2.0 are calling a Web Service generated with the BizTalk Server 2006 Web Services Publishing Wizard, it is possible that the client cannot receive server script timeout errors because the client request timeout occurs first by default. To resolve this problem you can do one of the following:

1. Increase the client request timeout to a value greater than the server script timeout by increasing the value for the **HttpWebRequest.Timeout** property on the client.

2. Reduce the server script timeout to a value less than the client request timeout by reducing the value for the **HttpServerUtility.ScriptTimeout** property on the server.

Debugging Published Web Services

This section provides information about debugging your published Web services.

In This Section

1. Enabling SOAP Message Tracing

2. Uncommenting BTSWebSvcWiz.exe.config

3. Using the ThrowDetailedError Switch

How to Enable SOAP Message Tracing

You can enable SOAP message tracing to help you debug the Web services publishing application. You set a breakpoint in a published Web service to return detailed exceptions to the Web client.

After completing the BizTalk Web Services Publishing Wizard, the published Web service project folder contains TraceExtension.cs that implements a SOAP extension. The SOAP extension traces the request and response SOAP message by saving it to a file during run time. By default, the file is not included in the Web service project. You must include and enable the SOAP extension.

To enable the SOAP extension

- Open the published Web service project.
- Select the project node in Solution Explorer.
- In Solution Explorer, click **Show All Files**.
- In the project node, select TraceExtension.cs.
- Right-click the file node, and then click **Include In Project** on the shortcut menu.
- Double-click the file node to open TraceExtension.cs.

- Copy the SOAP extension configuration node from the comments, located at the top of the file:
- In the project node, double-click the **Web.config** file.
- Paste the SOAP extension configuration node under the **<soapExtensionTypes>** node in the Web.config file.
- Rebuild the Web service project.
- Reset Internet Information Services, by typing **iisreset** from a command prompt.
- Run the Web service to log the SOAP message to a file.

To view the SOAP traced messages

- In Windows Explorer, browse to the BizTalkWebServices folder in the ASPNET temporary folder (%SystemDrive%\Documents and Settings\%COMPUTERNAME%\ASPNET\Local Settings\Temp\BizTalkWebServices).

1. Use the Visual Studio .NET debugger to set a breakpoint inside of the trace extension. For more information about setting a breakpoint in Visual Studio .NET, see "Using the BreakPoints Window" in the Visual Studio .NET Help Collection at http://go.microsoft.com/fwlink/?LinkId=26062.

Uncommenting BTSWebSvcWiz.exe.config

You can enable tracing to debug your published Web services by uncommenting the <add> node in the BTSWebSvcWiz.exe.config file. If the trace listener node is uncommented and the *initializeData* parameter is unchanged, BizTalk Server writes the trace file output to the current directory. Alternatively, you can set the trace level of **ApplicationTraceSwitch** and set the path name of the trace file.

Using the ThrowDetailedError Switch

If an error occurs, the Web client receives a generic **SoapException**.

To debug your published Web service, you can add a switch to the Web.config file to control the level of the exception details returned from the published Web service.

The Web.config file contains an application setting switch, **ThrowDetailedError**. **False** is the default setting for **ThrowDetailedError**. If you change the setting to **True**, the server proxy returns inner exception information to the Web client enabling you to debug the published Web service.

The following XML code shows the **ThrowDetailedError** switch that appears in the Web.config file under the <appSettings> node:

Testing Published Web Services

As you develop your Web service, you may want to test it by republishing your Web service. Each time you republish your Web service, you must reset Internet Information Services (IIS).

You can test your published Web service by creating a Web service client application or by using a browser such as Internet Explorer.

In This Section

1. Creating a .NET Application to Test a Published Web Service

2. Using Internet Explorer to Test a Published Web Service

Creating a .NET Application to Test a Published Web Service

To test your published Web service, you can create an ASP.NET Web client application that consumes your published Web service. The Visual Studio .NET Help Collection contains a valuable walkthrough for creating an ASP.NET Web client application. You can use the walkthrough to test your published Web service.

The walkthrough creates an ASP.NET Web client application, adds a Web reference, and provides sample code to show you how to access your published Web service. The walkthrough takes you through running the application in debug mode.

For information and procedures about creating an XML Web service client project, see "Walkthrough: Accessing an XML Web Service Using Visual Basic or Visual C#" in the Visual Studio .NET Help Collection at http://go.microsoft.com/fwlink/?LinkId=25564. You can replace the Web service used in this example, TempConvert1, with a BizTalk published Web service.

How to Use Internet Explorer to Test a Published Web Service

You can test your published Web service without writing a Web client application. You can use a Web browser, such as Internet Explorer, to test your published Web service. Although you can access any published Web service using a Web browser, you can only test Web services with Web methods that contain simple type parameters. To test your Web method in a Web browser, your message parts for the request and response messages that are used in the receive port can only be a simple type, such as **System.String** or **System.Int32**. If any message part uses a schema as a message type, you cannot test the Web method with a browser.

If you want to test your published Web services using HTTP-GET or HTTP-POST, you must configure your BizTalk receive location for the SOAP adapter and modify the Web.config file for your published Web service.

Modifying the Receive Locations

When the SOAP adapter configures receive locations, the SOAP adapter normally sets the URI of the receive location by giving the virtual directory and the Web service .asmx file name:

This allows the SOAP adapter to receive Web service requests using the HTTP-SOAP protocol. To configure the receive location to use the HTTP-GET or HTTP-POST protocol, you must append the method name to the URI:

The method name is the same as the port operation name in the orchestration.

Modifying the Web.config file

By default, the wizard configures the Web services to use the HTTP-SOAP protocol. HTTP-GET and HTTP-POST are explicitly disabled. To test a Web service with a Web browser, you must enable HTTP-GET.

To modify the Web.config file for the published Web service

1. Open the Web.config file for the published Web service.
2. Find the <protocols> section:
3. For testing HTTP-GET, HTTP-POST, or HTTP-POST from the local computer, remove the corresponding line from the <protocols> section.

For more information about the configuration options, see "Configuration Options for XML Web Services Created Using ASP.NET" in the .NET Framework SDK documentation at http://go.microsoft.com/fwlink/?LinkId=25765.

To access a Web service with Internet Explorer

1. In Internet Explorer, in the **Address** box, type the URL for the Web service using the format **http://*servername*/*apppath*/*webservicename*.asmx**.

Parameter	**Value**
servername	The name of the server that you have deployed your XML Web service.
Apppath	The name of your virtual directory and the Web application path.

webservicename.asmx	The name of the XML Web service .asmx file.

The description for the Web service shows you all the Web service methods that the particular Web service supports. The Web service description page contains links for each available Web method and the service description of the Web service.

To test a Web service with Internet Explorer using HTTP-GET

- After accessing the Web service description page, click one of the Web methods listed in the Web service description page.
- Type the necessary parameters for the Web method, and then click **Invoke**.
- The server returns an XML response in the browser. If the return data type for the Web service is a double-precision floating-point number, the result might look like the following:

To test a Web service with Internet Explorer using HTTP-GET (alternate method)

1. In Internet Explorer, in the **Address** box, type the URL for the Web service using the format **http://servername/vdir/webservicename.asmx/Methodname?parameter=value**.

Parameter	**Value**
servername	The name of the server that you have deployed your XML Web service.
Apppath	The name of your virtual directory and the Web application path.
webservicename.asmx	The name of the XML Web service .asmx file.
Methodname	The name of a public method that the XML Web service exposes. If left blank, the description page for the XML Web service appears, listing each public method available in the .asmx file. (Optional)
parameter	The appropriate parameter name and value for any parameters required by your method. If left blank, the description page for the XML Web service appears, listing each public method available in the .asmx file. (Optional)

3. Press Enter. The Web browser displays an XML response from the server.

Deploying Published Web Services on a Non-Visual Studio .NET Computer

To deploy your published Web service on a computer that does not have Visual Studio .NET 2003 installed, you need to create a Web setup project, distribute the Web setup package (.msi file), install the package on the non-Visual Studio .NET 2003 computer, and configure the installed Web service.

In This Section

1. Creating a Web Setup for Your Published Web Service
2. Distributing the Web Setup Package
3. Installing the Web Setup Package
4. Configuring the Installed Web Service

How to Create a Web Setup for Your Published Web Service

You can create an installation package to facilitate setting up your Web service in a production environment. This produces an .MSI file.

To create an installation package to produce an .MSI file

1. Open your published ASP.NET Web service project.
2. In Solution Explorer, right-click the solution node, and then click **Add New Project**.
3. In the **Add New Project** dialog box, in the **Project Types** area, select **Setup and Deployment Projects** and in the **Templates** area, select **Web Setup Project**.
4. In the **Name** text box, type a name for the Web Setup Project. You can use the same name as the ASP.NET Web Service project and add "_Setup" as a suffix and then click **OK**.
5. In Solution Explorer, right-click the new **Web Setup Project** node and point to **View**, and then click **File System**.
6. In the **File System** window, right-click the **Web Application Folder** node and point to **Add**, then click **Project Output**.
7. In the **Add Project Output Group** dialog box, select **Primary Output** and **Content Files** from the ASP.NET Web Service project and then click **OK**.

8. In Solution Explorer, expand the **Detected Dependencies** node under the Web Setup project.

9. Select all dependencies except **Microsoft.BizTalk.WebServices.ServerProxy.dll**.

10. In the **Properties** window, set the **Exclude** property to **True**.

11. Build your Web Setup project.

For more information about creating Web setup projects, see "Creating or Adding a Web Setup Project" in the Visual Studio .NET Help Collection at http://go.microsoft.com/fwlink/?LinkId=25571.

How to Distribute the Web Setup Package

After you create the installation package, you need to create a MSI file and a BindingInfo.xml file in your distribution folder to set up the Web service.

To distribute the Web setup package

1. Create a distribution folder to contain your setup files.

2. Copy the .MSI file from the Web Setup project output folder to the distribution folder.

3. Copy the BindingInfo.xml file from the ASP.NET Web Service project folder to the distribution folder.

How to Install the Web Setup Package

Use the contents of the distribution folder to setup the Web service on a destination computer.

To install the Web setup package

- Copy the distribution folder onto the destination computer.
- Run the .MSI file to install the Web service and follow the setup wizard prompts.
- Verify that the security settings for the virtual directory are correct for authorization.

How to Configure the Installed Web Service

After you install the Web service files, you must configure BizTalk Server to receive messages from the Web service.

To configure the Web service

- At the command prompt, type **BTSDeploy.exe Import Binding="BindingInfo.xml"**, and then press Enter.

  The binding parameter is the path name to the binding information file.

- In BizTalk Explorer, bind the orchestration ports to the correct receive ports.

- Start your orchestrations and enable receive locations. Administer as usual.

Using SOAP Headers

Microsoft® BizTalk® Server provides support for defined and unknown SOAP headers. Defined SOAP headers are headers in the Web Service Description Language (WSDL) that are explicity stated in the Web service. Unknown SOAP headers are headers that in the WSDL that are not explicity stated in the Web service. For more information about using SOAP headers, see "Using SOAP Headers" in the Microsoft .NET Framework documentation at http://go.microsoft.com/fwlink/?LinkId=25363.

BizTalk Server creates a context property for each defined SOAP header in the Web service.

In This Section

- SOAP Headers with Consumed Web Services

- SOAP Headers with Published Web Services

SOAP Headers with Consumed Web Services

After you add Web services to your orchestration using the **Add Web Reference** dialog box, you can use the SOAP headers that the Web Services Description Language (WSDL) defines in the Web service.

The WSDL for the consumed Web service lists the defined SOAP headers in the binding element. The following example shows a binding element in the WSDL file for the consumed Web service:

For more information about using SOAP headers, see "Using SOAP Headers" in .NET Framework documentation at http://go.microsoft.com/fwlink/?LinkId=25363.

In This Section

- Consuming Web Services with SOAP Headers

- Using SOAP Headers in Orchestrations

- Using SOAP Headers in Pipeline Components

Consuming Web Services with SOAP Headers

After you consume a Web service with defined SOAP headers, these headers become available to your orchestrations and pipeline components as context properties. These context properties contain string representations of the SOAP headers. For each defined SOAP header in the Web service, you can create a context property by using the name that corresponds to the root element of the SOAP header. All defined SOAP header context properties are in the **http://schemas.microsoft.com/BizTalk/2003/SOAPHeader** namespace.

The following example shows how to create a SOAP header context property **OrigDest** using the SOAP header example in SOAP Headers with Consumed Web Services:

Response SOAP headers also contain string representations of the defined SOAP header. The values are similar to creating a request SOAP header.

Using SOAP Headers in Orchestrations

Orchestrations use property schemas to define SOAP header context properties. You use the BizTalk Editor to set SOAP header context properties.

Defining SOAP header context properties with property schemas

You need a property schema to use defined SOAP header context properties in orchestrations. The property schema must have the target namespace of http://schemas.microsoft.com/BizTalk/2003/SOAPHeader. Each root element name in the property schema must match the root element name in the defined SOAP header. You can then set values for the context properties using the namespace of the property schema and the property name.

The following code shows assigning a SOAP header context property where the property schema namespace is SOAPHeader with a property name of OrigDest:

For more information about property schemas and context properties, see Property Schemas.

Using BizTalk Editor to set SOAP header context properties

For orchestrations, the SOAP header context properties are set to strings that contain XML data. You set these strings using the BizTalk Expression Editor in a Message Assignment or Expression shape.

The following example shows the string setting the context property:

Using BizTalk Editor to create an instance of a SOAP header root element

Setting the SOAP header to the correct string can be difficult. When adding a Web reference to a BizTalk project, all complex Web message parts are added to Reference.xsd as root elements. Reference.xsd also contains root elements for each defined SOAP header. To ensure that you set the SOAP header with the correct string, you should use BizTalk Editor to create an instance of the SOAP header root element for Reference.xsd. You can use the generated instance data directly or the instance data to contain your real data.

For more information about using the BizTalk Editor to generate instance data, see Generating Instance Messages.

Creating an XmlDocument to set context properties

You can set context properties by creating an XmlDocument and writing the string value of the XmlDocument to the context property. You declare a variable of type XMLDocument and assign the XML data.

The following example shows setting a declaring a variable of type XMLDocument and assign the XML data:

The following example shows setting the context property:

For more information about using BizTalk Expression Editor, see BizTalk Expression Editor. For more information about calling .NET classes, see Constructing Messages in User Code.

Creating SOAP headers for a SOAP request

When you create SOAP headers for the SOAP request, you must ensure that you have correctly created the SOAP headers. The SOAP adapter does not verify the contents of the SOAP header context properties.

The SOAP response that BizTalk returns to the Web service may also contain SOAP headers. You can only access these SOAP headers if they are defined SOAP headers.

For more information about defined SOAP headers, see Using SOAP Headers. The response SOAP headers are set to context properties using the same syntax as the request SOAP headers.

The following code shows how to access the response SOAP headers:

The values contained in the context properties are strings containing XML data. You set these strings using BizTalk Expression Editor in a Message Assignment or Expression shape. You load the string in an XmlDocument and use XPath queries to access specific fields.

For more information about creating XML documents in BizTalk Expression Editor, see Orchestration Developer Reference.

Using SOAP Headers in Pipeline Components

To access the SOAP header context properties in pipeline components, you use a combination of the context property name and target namespace as discussed in Using SOAP Headers in Orchestrations.

SOAP Headers with Published Web Services

You can add SOAP headers to your published Web services. In the BizTalk Web Services Publishing Wizard on the **Web Services Properties** page, you can check the **Add SOAP headers** box to add SOAP headers to your published Web services.

In This Section

- Publishing Web Services with SOAP Headers
- Accessing SOAP Headers in Orchestrations
- Accessing SOAP Headers in Pipeline Components

Publishing Web Services with SOAP Headers

You add SOAP headers to your Web services when you run the BizTalk Web Services Publishing Wizard. When you publish a Web service that supports SOAP headers, the headers become available to orchestrations and pipeline components as context properties that contain string representations of the SOAP headers.

Defined SOAP headers

When you add a defined SOAP header using the wizard, the wizard creates a context property with a name that corresponds to the root element of the SOAP header. All defined SOAP header context properties have the namespace **http://schemas.microsoft.com/BizTalk/2003/SOAPHeader**. When the SOAP adapter converts the SOAP request to a BizTalk message, it creates one SOAP header context property.

The following example shows a simple SOAP request:

For the simple SOAP request, the SOAP adapter created a BizTalk message with one SOAP header context property **OrigDest** and the string.

The following example shows the string created by the SOAP adapter:

Unknown SOAP headers

If you choose to support unknown SOAP headers in the wizard, the wizard creates a context property with the name **UnknownHeaders** and the namespace **http://schemas.microsoft.com/BizTalk/2003/soap-properties**. The **UnknownHeaders** context property contains all of the received unknown SOAP headers.

For example, if you receive an unknown SOAP header with the root element name, **CustomerGroup**, the **UnknownHeaders** context property contains the string:

Accessing SOAP Headers in Orchestrations

You can access the SOAP header context properties in orchestrations for defined and unknown SOAP headers. For more information about property schemas and context properties, see Property Schemas.

Defined SOAP header context properties

The defined SOAP header context properties in orchestrations require a property schema. The property schema must have the target namespace **http://schemas.microsoft.com/BizTalk/2003/SOAPHeader**. Each root element name in the property schema must match the root element name of the defined SOAP header. You can then access the values of the context properties using the namespace of the property schema and the property name. The namespace of the property schema is different from the target namespace listed above. Although the namespace of the property schema can be any string, it usually defaults to the name of the project.

The following example shows accessing the SOAP header context property for a property schema namespace, **SOAPHeader**, and the property name **OrigDest**:

Copying SOAP header context property of incoming message

You can copy the SOAP header context property of an incoming message to the same SOAP header context property of the response message.

The following example shows copying the SOAP header context property:

When you create SOAP headers for the SOAP response, you ensure that you create your SOAP headers correctly. The SOAP adapter does not verify the contents of the SOAP header context properties. If the values of the response SOAP headers are incorrect, the SOAP adapter cannot send the response message to the consumer of your Web service.

Unknown SOAP header context property

The unknown SOAP header context property does not require a property schema. You can access this global context property **SOAP.UnknownHeaders**.

The following example shows accessing the unknown SOAP header context property **SOAP.UnknownHeaders**:

The values contained in the context properties are strings containing XML data. The simplest way to access this data is to use the BizTalk Expression Editor in a **Message Assignment** or **Expression** shape and load the string in an **XmlDocument** and use XPATH queries to access specific fields. For more information creating XML documents in the BizTalk Expression Editor, see **Orchestration Developer Reference**.

Context properties are associated with a particular message. The Messaging Engine does not automatically map the values of known SOAP headers from the request message to the response message. When creating the response message for a Web service, you must specifically set the SOAP header values. The following command is the simplest method of setting a SOAP header context property:

You can also achieve this by creating an **XmlDocument** and writing the string value of the **XmlDocument** to the context property.

Accessing SOAP Headers in Pipeline Components

You can access SOAP header context properties in pipeline components. You use a combination of the context property name and the target namespace **http://schemas.microsoft.com/BizTalk/2003/SOAPHeader**.

International Considerations for Designing BizTalk Applications

It is strongly recommended that you review the following known issues when you develop your international BizTalk Server applications.

Character Restrictions for Machine Names

Installing BizTalk Server 2006 on machines with names that contain letters outside of the following set is not supported: letters (A-Z, a-z), digits (0-9), hyphens (-), and underscores (_). Only machine names that contain the letters in this set are supported.

Characters do not appear correctly or do not appear at all due to incorrect font and font fallback settings

You may experience issues displaying characters (such as Czech characters) in the BizTalk Server tools hosted in Microsoft® Visual Studio® .NET. To address these issues, you may need to modify the font settings available in the Visual Studio Options tab and select another font that you know supports the characters. You can select Tahoma or Microsoft Sans Serif as the default font for which font fallback capabilities are provided.

Surrogate pair characters displayed as squares in BizTalk Server Administration Console and other BizTalk Server tools

You may not be able to display surrogate pair characters in BizTalk Administration Console and other BizTalk Server tools. Surrogate pairs are a coded character representation for a single abstract character that consists of a sequence of two code units. Make sure you have the appropriate font installed on your system (it is included in the Chinese versions of Office XP and 2003). It may also be necessary to change the font options in the tools that have such capability (Such as Visual Studio .NET).

In other tools without a font setting option, surrogate pair characters will be displayed as squares, such as the Administration Console. If you see squares, the characters are not corrupted; they just can not be displayed properly because of a lack of font support.

Non-ASCII characters and TargetNamespace schema property

The TargetNamespace property of a schema in a BizTalk project defaults to "http://<Project name>.<Schema name>." However, it will be set to http://tempURI.org if non-ASCII characters are used and the length of the namespace is longer than 15 characters.

This is a restriction from the uniform resource identifier (URI) class in the .NET Framework. The URI validation fails if any string has invalid domain name characters and is longer than 15 characters before the first slash, which is the limit for universal naming convention (UNC) names.

When only ASCII characters are used, the host is checked as an Internet host name. The character limit for Internet host names is fewer than 63 characters before the dot. No restriction is imposed on the Internet hostname length when only ASCII characters are used.

Web Services character limitations

If you plan on publishing an orchestration as a Web Service, you may encounter issues with characters used in the orchestration names and port names as those names are used in the Web Services file names (.asmx files) and the virtual directory in the Web Services Publishing Wizard. Only Microsoft Internet Information Services (IIS) 6.0 (included in Microsoft Windows Server™ 2003) fully supports Unicode characters. Therefore, if you use an earlier version of IIS or Windows, the names of the orchestrations, ports, Web services and virtual directories names must include only ANSI characters supported by the language version of Windows (for example, Japanese characters are not allowed on an English version of Windows).

Note also that project names for Web Services in Visual Studio .NET are restricted to ASCII characters.

Working with different document encodings

BizTalk Server 2006 supports many different encodings for XML and flat file documents, for example UTF-16, UTF-8, Simplified Chinese GBK, Simplified Chinese GB18030, and so on.

For inbound documents, BizTalk Server can recognize the encoding declaration in XML documents, such as "<?xml version="1.0" encoding="GB2312" ?>". The flat file schema has a **Code Page** property to indicate the encoding of the inbound flat file documents.

For outbound documents, XML and flat file assemblers use the **Target charset** property. If this property is specified, BizTalk Server converts the outbound documents to the specified character set regardless of the original one. If no **Target charset** property is set, XML uses the UTF-8 protocol and flat files use the code page specified in the flat file schema.

Code Conversion from an unsupported code page to a Windows code page

To implement code conversions from unsupported code pages into a Windows code page, you must create a custom pipeline component.

Byte-order mark impact on document encoding

BizTalk Server determines character encoding and produces documents with a particular character encoding differently for flat file and XML messages.

Schema Editor may contain properties in more than one language

XML Schema Definition language (XSD) property names shown in the Schema Editor Properties window and in the XML source code are not localized, and appear in English in all localized versions. Other properties are shown in the local language. For example, in the simplified Chinese version of BizTalk Server 2006, the schema properties are in English, but additional properties are displayed in Chinese.

Locale-dependent data in flat files

Many locales represent data such as date, time, number, and currency using different formats than those formats defined in the XML standard. For instance, several locales use a decimal separator other than a period (.), so the number five and three quarters may be represented as 5,75.

In BizTalk Server 2006, all fields from flat files except date and time are treated as strings, so that parsing can succeed. However, when using XML validation, the resulting XML message fails during validation against the schema.

For date and time fields, the parser attempts to parse the field value to the DateTime instance using a custom date or time format if it is defined, and writes it in XML format, or uses the original value as a string if date or time format is not defined. Again, if XML validation is used, the resulting date or time may fail validation if a custom date or time format is not used, and the field value used in flat file message was not in the correct XML date or time format.

Note that you can also create custom pipeline components or maps to update field values to produce valid XML.

Using the Find Message View in HAT using a non-English locale

If you run a query in the Find Message View of HAT, and you query on non-English characters, you may get incorrect results if you use the EQUAL operator. Use the LIKE operator instead.

BAM definition language support

Before you can deploy a BAM definition XML file, you must ensure that the language used to create this file matches the locale settings of the computer on which it is being deployed. If the file and the computer settings dont match, you must first reboot the computer used to run the BM.exe.

To change the locale settings on your computer, update the following settings based on your operating system:

Operating System	Control Panel	Setting
Windows 2000	Regional Options	Set the Default button (General tab). Select the value in the drop-down list.
Windows XP	Regional and Language Options	Language for non-Unicode programs (Advanced tab)
Windows Server 2003	Regional and Language Options	Language for non-Unicode programs (Advanced tab)

Deploying BizTalk Assemblies from Visual Studio into a BizTalk Application

This section describes how to deploy and redeploy BizTalk assemblies from Visual Studio 2005 into a BizTalk application. You may want to do this to test the functionality of the assemblies that you have been developing and package them for handoff. New features included with BizTalk Server 2006 make this process quicker and easier. For more information, see What's New in BizTalk Server 2006 . This section also provides background information about the way Visual Studio deploys applications into a BizTalk application.

BizTalk applications are new with BizTalk Server 2006. They provide a way to view and manage the items, called "artifacts," that comprise a BizTalk business solution. Artifacts include BizTalk assemblies and their orchestrations, schemas, maps and pipelines, which you can deploy into BizTalk Server from Visual Studio. Artifacts also include items such as .NET assemblies, certificates, scripts, Readme files, and policies, which you add to your BizTalk application by using the BizTalk Server Administration console or the BTSTask command-line tool. The solution developer or

IT administrator can then view, package, deploy, and manage the application and its artifacts as a single entity. For more information about creating, deploying, and managing BizTalk applications, see Managing BizTalk Applications.

Before you can build and deploy a BizTalk assembly, you must create a project in Visual Studio and either create or add the items that you want to include in the assembly. You can optionally create a solution in Visual Studio to contain multiple projects and then build and deploy their assemblies all at once into the same application or different applications. For instructions on performing these tasks, see Creating BizTalk Projects .

After completing these tasks, you can build and deploy the BizTalk assemblies by taking the following steps, as described in the topics in this section:

- Configure a strong name assembly key file for each Visual Studio project.
- Set deployment properties for the project, including configuring the Redeploy option to easily redeploy the assembly.
- Use the Deploy command in Visual Studio to build the BizTalk assembly in the project and deploy it into a BizTalk application. Alternatively, you can use the Deploy command at the solution level to build and deploy all of the assemblies in the solution.
- After testing the application and making necessary changes, use the Deploy command in Visual Studio to rebuild and redeploy the assembly.
- Install the assembly in the global assembly cache (GAC), if necessary, or remove the assembly from the GAC.

After deploying one or more assemblies into a BizTalk application, you can complete the configuration of the application and deploy it into a test and then production environment. For more information, see Development Tasks for BizTalk Application Deployment.

In This Section

- What Happens When You Deploy an Assembly from Visual Studio
- How to Configure a Strong Name Assembly Key File
- How to Set Deployment Properties in Visual Studio
- How to Deploy a BizTalk Assembly from Visual Studio
- How to Redeploy a BizTalk Assembly from Visual Studio
- How to Install an Assembly in the GAC

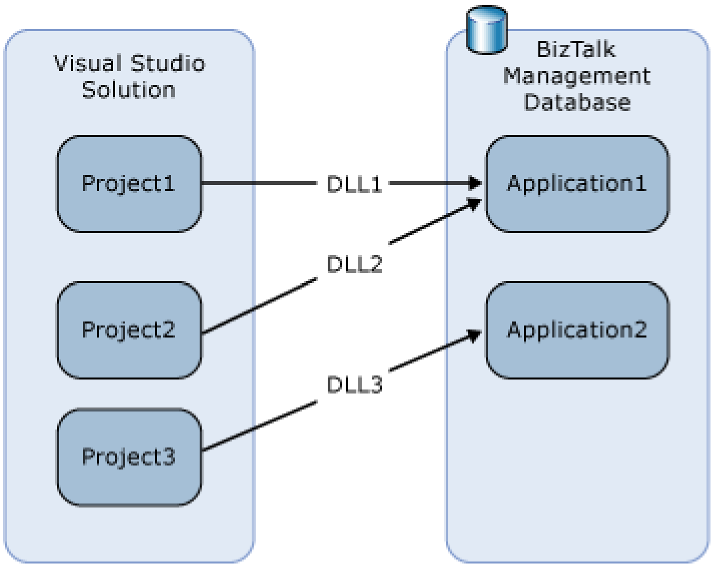
- How to Uninstall an Assembly from the GAC

What Happens When You Deploy an Assembly from Visual Studio

This topic describes what happens when you deploy assemblies from Visual Studio 2005 into a BizTalk application on BizTalk Server 2006.

You can deploy a project individually, or you can deploy all of the projects in a solution at the same time. Before deploying a project, either separately or as part of a solution, you specify the application into which to deploy its assembly in project properties, as described in How to Set Deployment Properties in Visual Studio. When you deploy a project or solution in Visual Studio, the assemblies are automatically built and deployed into the specified application. If an existing application in the local BizTalk group has the same name as the application specified in project properties, the assembly is deployed into the existing application; otherwise, a new application having the specified name is created and assembly is deployed into it. As part of this process, the assembly along with the orchestrations, pipelines, schemas, and maps that it contains (called "artifacts") are imported into the local BizTalk Management database and associated in the database with the specified application.

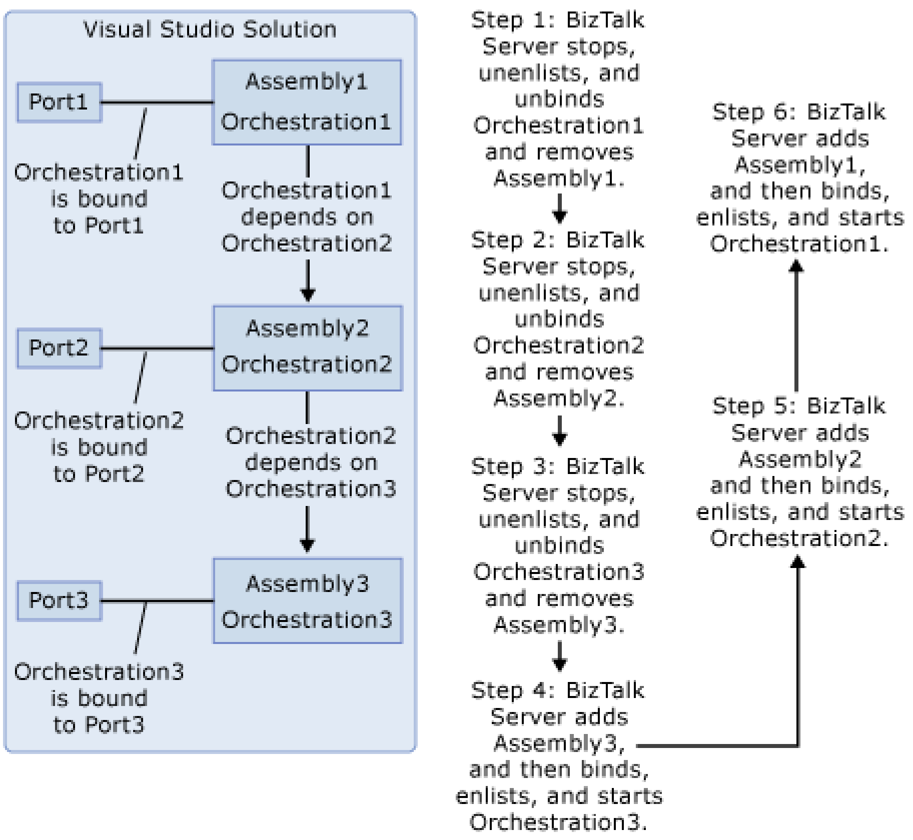
You can deploy the projects in a solution into the same BizTalk application or different BizTalk applications, even when you deploy the projects in a solution at the same time. The following diagram illustrates deploying three assemblies contained in a BizTalk solution in Visual Studio into two different BizTalk applications.



After you deploy a project or solution, you can view and manage the assemblies and their artifacts from within the BizTalk Server Administration console or by using the BTSTask command-line tool.

If there are dependencies between the assemblies in your solution, we recommend that you deploy solutions rather than projects. Otherwise, you must take a number of manual steps to complete the deployment. When you deploy a solution, BizTalk Server automatically take all of the steps to manage dependencies between assemblies.

The following diagram illustrates the steps that BizTalk Server 2006 takes to redeploy assemblies that have dependencies when you deploy a solution.


Visual Studio Solution
Port1
Assembly1
Orchestration1
Orchestration1 is bound to Port1
Orchestration1 depends on Orchestration2
Port2
Assembly2
Orchestration2
Orchestration2 is bound to Port2
Orchestration2 depends on Orchestration3
Port3
Assembly3
Orchestration3
Orchestration3 is bound to Port3
Step 1: BizTalk Server stops, unenlists, and unbinds Orchestration1 and removes Assembly1.
Step 2: BizTalk Server stops, unenlists, and unbinds Orchestration2 and removes Assembly2.
Step 3: BizTalk Server stops, unenlists, and unbinds Orchestration3 and removes Assembly3.
Step 4: BizTalk Server adds Assembly3, and then binds, enlists, and starts Orchestration3.
Step 5: BizTalk Server adds Assembly2 and then binds, enlists, and starts Orchestration2.
Step 6: BizTalk Server adds Assembly1, and then binds, enlists, and starts Orchestration1.

How to Configure a Strong Name Assembly Key File

In the process of deploying a BizTalk solution, Visual Studio 2005 first builds the assemblies. The deployment process requires that each project in the solution is associated with a strong name assembly key file. If you haven't already done so, before deploying a solution from Visual Studio, use the following procedure to generate a strong name assembly key file and assign it to your projects.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the BizTalk Server Administrator's group. In addition, your account must have Write permissions on the global assembly cache (GAC). The Administrators account on the local computer has this permission.

To configure a strong name assembly key file

1. On the **Start** menu, point to **Programs**, point to **Microsoft Visual Studio 2005**, point to **Visual Studio Tools**, and then click **Visual Studio 2005 Command Prompt**.
2. At the command prompt, from the folder where you want to store the key file, type the following command, and then press ENTER:

   sn -k *file_name***.snk**

   Example: **sn -k ErrorHandling.snk**

   A confirmation message, **Key pair written to <resource_file>.snk**, displays on the command line.
3. In Visual Studio Solution Explorer, right-click the project and then click **Properties**.
4. In the left pane, expand Common Properties and click **Assembly**.
5. In the right pane, scroll to the **Strong name** box.
6. In the **Strong name** box, click the field next to **Assembly Key File**, click **...,** and then browse to the key file.
7. Click the key file, click **Open**, and then click **OK**.
8. Repeat steps 3 through 7 for each project in the solution that you want to deploy using this strong name assembly key file.

How to Set Deployment Properties in Visual Studio

Before you can deploy a solution from Visual Studio 2005 into a BizTalk application, you must first set project properties. If a solution in Visual Studio contains multiple projects, you must separately configure properties for each project.

Prerequisites

To perform the procedures in this topic, you must be logged on with an account that has Read/Write permission on the local file system. The Administrators account on the local computer has this permission.

To configure project properties

1. In Visual Studio Solution Explorer right-click a project for which you want to configure properties, and then click **Properties**.

2. In the left pane, expand **Configuration Properties**, and then click **Deployment**.

3. Configure project properties as described in the following table, and then click **OK**.

4. Repeat steps 1, 2, and 3 for each project in the solution.

Property	**Value**	**Explanation**
Server	<Server name>	Name of the SQL Server instance that hosts the BizTalk Management database on the local computer. In a single-computer installation, this is usually the name of the local computer. This property is stored in the *.btproj.user file as the Config property Server.
BizTalk Management database	<BizTalk Management database name>	Name of the BizTalk Management database for the group, for example BizTalkMgmtDb. This property is stored in the *.btproj.user file as the Config property ConfigurationDatabase.
Application Name	<Name>	Name of the BizTalk application to which to deploy the assemblies in this project. If the application already exists, the assemblies will be added to it when you deploy the project. If the application does not exist, the application will be created. If this field is blank, the assemblies will be deployed to the default BizTalk application in the current group, "BizTalk Application 1" by default. Names that include spaces must be enclosed by double quotation marks ("). This property is stored in the *.btproj file as the Config property ApplicationName.
Redeploy	True or False	Setting this to True (the default) enables you to redeploy the BizTalk assemblies without changing the version number. This property is stored in the *.btproj file as the Config property Redeploy.
Install to Global Assembly Cache	True or False	Setting this to True (the default) installs the assemblies to the Global Assembly Cache (GAC) on the local computer when you install the application. Set this to False only if you plan to use other tools for this installation, such as gacutil. This property is stored in the *.btproj.user file as the Config property Register.

Restart all host instances	True or False	Setting this to True automatically restarts all host instances running on the local computer when the assembly is redeployed. If set to False (the default), you must manually restart the host instances when you redeploy an assembly.

How to Deploy a BizTalk Assembly from Visual Studio

This topic provides instructions on using Visual Studio 2005 Solution Explorer or the Visual Studio 2005 command prompt to deploy the BizTalk assemblies in a solution from Visual Studio 2005 into a BizTalk application. You can deploy a single assembly from the project level (such as by right-clicking the project and clicking Deploy) or deploy all of the assemblies in the solution at once from the solution level (such as by right-clicking the solution and clicking Deploy). When redeploying your solution, we strongly recommend using the latter method. For more information, see How to Redeploy a BizTalk Assembly from Visual Studio.

With previous versions of BizTalk Server, if you wanted to deploy a multiple assemblies in a solution, and any of the assemblies had a dependency on any of the other assemblies, you had to individually deploy the assemblies in the reverse order of their dependencies. For example, if Assembly1 had a dependency on Assembly2, you had to deploy Assembly2 first, and then you could deploy Assembly1. This is still the case when you deploy assemblies from the project level. With BizTalk Server 2006, however, when you deploy assemblies from the solution level, BizTalk Server automatically takes care of all of the deployment steps, including deploying assemblies in the correct order. Therefore, to simplify deployment, when another assembly has a dependency on the assembly that you are redeploying, you should redeploy your assemblies at the solution level.

When you select the option to deploy a project or solution from within Visual Studio, the assembly or assemblies are automatically built and deployed into the specified BizTalk application in the local BizTalk group. If the application does not already exist in the group, deployment also creates the application. The assemblies and the artifacts they contain are registered and their data stored in the BizTalk Management (configuration) database for the BizTalk group. In addition, if you specify this option in deployment properties for the project, the assemblies are added to the global assembly cache (GAC).

An "artifact" is any item included in a BizTalk application, including resources you work with in Visual Studio, such as assemblies and orchestrations as well as other items you create or add later after deploying the application, such as send and receive ports, certificates, and scripts. After the assembly has been deployed, you can view and manage its artifacts in the Applications node of BizTalk Server Administration console. Each application is stored in its own folder, with subfolders displaying the artifacts in the application. For more information, see Using the BizTalk Server Administration Console. For more information about creating and managing applications, see Managing BizTalk Applications.

Before you deploy an assembly, you must take the following steps:

- Create a strong name assembly key file and assigned it to each project, as described in How to Configure a Strong Name Assembly Key File.
- Set deployment properties for the project, as described in How to Set Deployment Properties in Visual Studio.
- If you have previously deployed the assembly, enable the redeployment option for the project. For instructions and other important information about redeployment, see How to Redeploy a BizTalk Assembly from Visual Studio.

Prerequisites

To perform the procedures in this topic, you must be logged with an account that is a member of the BizTalk Server Administrators group. If, in Deployment properties, you have enabled the option to install an assembly to the global assembly cache (GAC), then you also need Read/Write permissions on the GAC. The Administrators account on the local computer has this permission. For more detailed information on permissions, see Permissions Required for Deploying and Managing a BizTalk Application.

To deploy a BizTalk assembly or assemblies

Using Visual Studio 2005 Solution Explorer

- In Visual Studio 2005 Solution Explorer, right-click a BizTalk project or solution, and then click **Deploy**.

  The assembly in the project or assemblies in the solution are deployed into the specified BizTalk application. The status of the build and deployment process displays in the lower left corner of the page.

Using the Visual Studio 2005 command prompt

1. Open a Visual Studio 2005 command prompt as follows: Click **Start**, point to **Programs**, point to **Microsoft Visual Studio 2005**, point to **Visual Studio Tools**, and then **click Visual Studio 2005 Command Prompt**.
2. Type the following command, substituting the appropriate values, as described in the following table:

   devenv /deploy *SolnConfigName SolutionName* [**/project** *ProjName*] [**/projectconfig** *ProjConfigName*]

   Example:

devenv /deploy Release "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln" /project "MyBizTalkApp\MyBizTalkApp.csproj" projectconfig Release

Parameter	**Value**
/deploy	Deploys a solution after a build or rebuild.
SolnConfigName	Name of the solution configuration that will be used to build the solution named in SolutionName.
SolutionName	Full path and name of the solution file.
/project *ProjName*	Path and name of the project file within the solution. You can enter a relative path from the SolutionName folder to the project file, or the project's display name, or the full path and name of the project file.
/projectconfig *ProjConfigName*	Name of a project build configuration to be used when building the project.

The first time you deploy an assembly containing an orchestration, you may receive a warning message that the orchestration is not contained in the binding file. This is because orchestrations are not automatically bound to the host upon deployment. You must perform this step manually. For instructions, see How to Bind an Orchestration Using BizTalk Explorer .

How to Redeploy a BizTalk Assembly from Visual Studio

In the process of developing an assembly, you often need to deploy, test, modify, and redeploy it repeatedly. In previous versions of BizTalk Server, if you wanted to redeploy an assembly without changing the version number, you first needed to manually stop, unenlist, and unbind artifacts contained in the assembly in BizTalk Server and then remove the assembly from the BizTalk Management (configuration) database. In addition, after redeploying the assembly, you needed to bind, enlist, and start its artifacts in BizTalk Server.

With BizTalk Server 2006, however, when you enable the Redeploy option in Visual Studio 2005, BizTalk Server automatically takes all of the steps to redeploy the assembly for you. You can redeploy assemblies from the project level (such as by right-clicking the project in Solution Explorer and clicking Deploy) to redeploy an individual assembly or the solution level (such as by right-clicking the solution and clicking Deploy) to redeploy all of the assemblies in the solution at once.

When redeploying an assembly, bear in mind the following important points:

- **You must install the new assembly in the GAC.** When you redeploy an assembly, you must always install the new version of the assembly in the GAC, as described in How to Install an Assembly in the GAC. You can do this after you redeploy it.

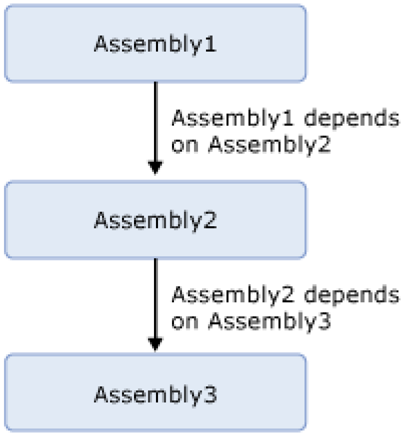
- **You should redeploy at the solution level when there are dependencies.** If you have multiple assemblies in a solution, and one or more assemblies in the solution has a dependency on the assembly that you want to redeploy, you should redeploy your assemblies at the solution level. This is because when you redeploy an assembly at the project level, BizTalk Server will stop, unenlist, unbind, and remove the artifacts in a dependent assembly, but will not take the additional steps to deploy, bind, enlist, and start the artifacts. When you redeploy the entire solution, however, BizTalk Server automatically takes the steps required to undeploy and redeploy all of the artifacts in the solution based on their dependencies.

- **You may need to manually redeploy dependent assemblies.** BizTalk Server always undeploys dependent assemblies, but in the following cases, you must take the additional steps to deploy, bind, and enlist the artifacts in each dependent assembly after redeploying the assembly on which the assembly depends:

  - If you redeploy an assembly at the project level and another assembly in the same solution depends on it.

  - If you redeploy an assembly at the solution level, but a dependent assembly exists in a different solution.

  For example, if you were to redeploy only Assembly 3 shown in the following diagram, following redeployment you would need to deploy, bind, and enlist the artifacts in Assembly 2, and then deploy, bind, and enlist the artifacts in Assembly 1.


Assembly1
Assembly1 depends on Assembly2
Assembly2
Assembly2 depends on Assembly3
Assembly3

- **You must restart host instances.** When you redeploy an assembly that contains an orchestration without changing the assembly version number, the existing assembly is overwritten in the BizTalk Management database. Before the change will take effect, however, you must restart each host instance of the host to which the orchestration is bound. You can specify the option that all host instances on the local computer restart automatically when you redeploy an assembly. For instructions, see How to Set Deployment Properties in Visual Studio. You can also manually stop and start each host instance, as described in How to Stop a Host Instance and How to Start a Host Instance.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that has Read/Write permissions on the local file system and the global assembly cache (GAC). The Administrators account on the local computer has these permissions.

To redeploy a BizTalk assembly or assemblies

1. Ensure that the Redeploy option is enabled in Deployment properties for each project that you want to redeploy, as described in How to Set Deployment Properties in Visual Studio. This option is enabled by default.

2. Redeploy the assembly or assemblies by using the procedure in How to Deploy a BizTalk Assembly from Visual Studio.

How to Install an Assembly in the GAC

This topic describes how to manually install either a BizTalk assembly or a non-BizTalk .NET assembly in the GAC by using the Gacutil tool included with Visual Studio 2005.

In Visual Studio 2005, you can have BizTalk assemblies automatically installed in the global assembly cache (GAC) when they are deployed from Visual Studio by specifying this option in Deployment Properties for the BizTalk project, as described in How to Set Deployment Properties in Visual Studio. You cannot, however, use this method to install non-BizTalk .NET assemblies in the GAC; you must install them manually, as described in this topic.

Prerequisites

To perform this procedure, you must be logged on with an account that has Write permission on the GAC. The Administrators account on the local computer has this permission.

To install an assembly in the GAC

1. Copy the BizTalk assembly to your local computer into a folder.

2. Open the Visual Studio 2005 command prompt, as follows: Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2005**, point to **Visual Studio Tools**, and then click **Visual Studio 2005 Command Prompt**.

3. Type the following command:

gacutil.exe /if "<*path to the assembly .dll file***>"**

This installs the assembly to the GAC, overwriting any existing assembly that has the same assembly name.

How to Uninstall an Assembly from the GAC

This topic describes how to use the Windows interface or the Gacutil command-line tool to uninstall an assembly from the global assembly cache (GAC). This is necessary to completely undeploy an application. To automate this process, before deploying the application into a production environment, we recommend that you write a post-processing script that will uninstall the assembly from the GAC automatically when the application is uninstalled and add it to the application. For more information, see Using Pre- and Post-processing Scripts to Customize Application Deployment.

You can also use such a script to remove additional files and settings, as described in How to Remove Other Files and Settings for a BizTalk Application.

Prerequisites

To perform the procedures in this topic, you must be logged on an account that has Write permissions on the GAC, which is located at %systemdrive%\Windows\Assembly. The Administrators account on the local computer has this permission. For more detailed information on permissions, see Permissions Required for Deploying and Managing a BizTalk Application.

To uninstall an assembly from the GAC

Using the Windows interface

1. Navigate to the GAC, which is located at %systemdrive%\Windows\Assembly.
2. Right-click each assembly file that is included in your application, click **Uninstall**, and then click **Yes** to confirm.

Using the command line

1. Open a Visual Studio 2005 command prompt as follows: Click **Start**, point to **All Programs**, point to **Microsoft Visual Studio 2005**, point to **Tools**, and then click **Microsoft Visual Studio 2005 Command Prompt**.
2. At the command prompt, type the following command:

   gacutil –u <*fully qualified assembly name*>

   In this command, assembly name is the name of the assembly to uninstall from the GAC.

The following example removes an assembly named hello.dll from the GAC.

```
gacutil /u "hello,Version=1.0.0.0, Culture=neutral,
PublicKeyToken=0123456789ABCDEF"
```

Creating a Single Sign-On Application

This section provides information about Enterprise Single Sign-On (SSO) technology.

In This Section

- Programming Single Sign-On
- Programming with Enterprise Single Sign-On

Programming Single Sign-On

A business process that relies on several different applications is likely to face the challenge of dealing with several different security domains. Accessing an application on a Microsoft® Windows® operating system may require one set of security credentials, while accessing an application on a remote system may require different credentials. Dealing with this profusion of credentials is hard for users, and it can pose an even greater challenge for automating processes. To address this problem, Microsoft provides Enterprise Single Sign-On.

Enterprise Single Sign-On provides a way to map a Windows user ID to non-Windows user credentials. Enterprise Single Sign-On also allows you to synchronize passwords across network applications, and also provides additional levels of database security. These services can simplify business processes that use applications on diverse systems.

To use Enterprise Single Sign-On, an administrator defines affiliated applications, each of which represents a non-Windows system or application. An affiliated application might be a Customer Information Control System (CICS) application running on an IBM mainframe, a SAP ERP system running on Unix, or any other kind of software. Each of these applications has its own mechanism for authentication, and so each requires its own unique credentials.

Enterprise Single Sign-On stores an encrypted mapping between the Windows user ID of a user and the associated credentials for one or more affiliated applications. These linked pairs are stored in a Credentials database. Enterprise Single Sign-On uses the Credentials database in two ways. The first way, called Windows-initiated Single Sign-On, uses the user ID to determine to which affiliated applications the user has access. For example, a Windows user account may be linked with credentials that grant access to a DB2 database running on a remote AS/400 server. The second way, called host-initiated Single Sign-On, acts in reverse: determining what remote applications have access to a specified user ID, and the privileges that go with that account. For example, a remote application may be linked with credentials that grant access to a user account that has administration privileges on a Windows network.

Note that Enterprise Single Sign-On also includes administration tools to perform various operations. All operations performed on the Credential database are audited; for example, tools are provided that enable an administrator to monitor these operations and set various audit levels. Other tools enable an administrator to disable a particular affiliated application, turn on and off an individual mapping for a user, and perform other functions. There is also a client program that enables end users to configure their own credentials and mappings.

One of the administrative requirements for Enterprise Single Sign-On is that your local system must be aware of the credentials necessary to log onto a remote system. Similarly, the remote system must be aware of the credentials on your local system. Thus, when you update your credentials, such as when you update your password on your local machine, you must also inform the remote systems that you have done so. The component you design that synchronizes passwords across an enterprise is called a password sync adapter.

In This Section

- The Single Sign-On Interface
- Single Sign-On Applications
- What you Need To Know Before Programming Single Sign-On
- Supported Platforms for Single Sign-On

The Single Sign-On Interface

The following table describes the .COM and .NET interfaces that make up the Single Sign-On interface.

.NET	COM	Description
ISSOAdmin	**ISSOAdmin Interface (COM)**	Sets up and maintains the SSO server database. This database contains the names and descriptions of the various affiliated applications currently exposed on the enterprise.
ISSOAdmin2	**ISSOAdmin2 Interface (COM)**	As above, but also enables you to create and update applications using the IPropertyBag object.
ISSOConfigDB	**ISSOConfigDB Interface (COM)**	Allows you to create, update, and retrieve database information.

ISSOConfigOM	**ISSOConfigOM Interface (COM)**	Discovers servers and retrieves server status.
ISSOConfigSS	**ISSOConfigSS Interface (COM)**	Configures the secret server.
ISSOConfigStore	**ISSOConfigStore Interface (COM)**	Gets and sets information in the SSO configuration store.
ISSOLookup1	**ISSOLookup1**	Enables you to look up the external credentials on a specified application for the current user.
ISSOLookup2	**ISSOLookup2**	As above, but also enables you to look up the Windows® credentials for a specified external user.
ISSOMapper	**ISSOMapper**	Mainly acts as a set of **Get** methods to expose the properties of **Islamic** and **ISSOMapping**, but also enables you to set the external credentials for the current user for a specified application.
ISSOMapper2	**ISSOMapper2 Interface (COM)**	As above, but returns additional user and administration account information.
ISSOMapping	**ISSOMapping**	Creates and maintains the map between users and affiliated applications.
ISSOPSAdmin	**ISSOPSAdmin Interface (COM)**	Administers the password synchronization features.
ISSOTicket	**ISSOTicket**	Creates the ticket that contains the appropriate security information. This ticket is then sent on with the appropriate message from your application.

In addition, Microsoft Host Integration Server supports the Password Sync Helper (PS Helper) component. You can use PS Helper when designing password synchronization components. The following table describes the COM interface exposed by PS Helper.

COM Object	Description
ISSONotification Interface (COM)	Handles password changes to and from non-Windows operating systems.
SExternalAccount Structure (COM)	Describes an external account.

SPasswordChange Structure (COM)	Describes a password change.
SPasswordChangeComplete Structure (COM)	Describes the completion of a password change.
SStatus Structure (COM)	Describes an error or event.
SAdapterInGroup Structure (COM)	Describes the adapters in a given group.
SAdapter Structure (COM)	Describes a specific adapter.

Single Sign-On Applications

From a programming perspective, you will write two different kinds of applications using Single Sign-On: a traditional Single Sign-On application that uses the Single Sign-On interface to interact with remote applications and to administer SSO databases, and a password sync adapter that uses the Password Sync (PS) Helper interface to synchronize passwords across your enterprise.

In This Section

- Traditional Single Sign-On Applications
- Password Sync Adapters

Traditional Single Sign-On Applications

The Single Sign-On programming architecture contains a mapping component to map between applications and users, a lookup component to look up credentials for a specified use, and an administration component to perform administrative tasks. In addition, SSO also contains a ticketing interface so that your application can issue and redeem tickets.

Mapping

Mapping is the process of linking a specified user with a specified application. You can map between affiliate applications and users using ISSOMapping, ISSOMapper, and ISSOMapper2. With ISSOMapping, you can create, delete, enable, and disable mappings. With ISSOMapper and ISSOMapper2, you can get and set mapping data for the current user.

Lookup

The core of the Single Sign-On programming interface is the ability to look up credentials for specified users. You can look up credentials using ISSOLookup1 and ISSOLookup2. ISSOLookup1 enables you to look up the credentials of a local user for a remote affiliate application. In contrast, ISSOLookup2 also enables you to look up the credentials of a remote user for a local affiliate application. The former is necessary for a traditional Single Sign-On application, while the latter is useful when writing a host-initiated Single Sign-On application.

Administration

You can perform many of the administrative capabilities programmatically through ISSOAdmin, ISSOAdmin2, and ISSOConfigStore. Such tasks include configuring single sign-on and creating and describing an application to single sign-on. For SSOv3, you also have the ability to create and modify SSO databases using ISSOConfigDB, ISSOConfigOM, and ISSOConfigSS. Finally, you can administer the password sync features using ISSOPSAdmin.

Communication and ticketing

Your application will most likely issue messages so that your user can communicate with a remote application. To communicate with a remote application securely, you will need to issue and redeem a Single Sign-On ticket. You can also issue and redeem tickets programmatically.

Password Sync Adapters

A password sync adapter is a component that propagates password changes to and from a non-Windows system. A password sync adapter is similar to a traditional Single Sign-On application, with several differences: they are administered using a specialized interface, they are described using a specialized XML format in the configuration store, and Microsoft Host Integration Server uses a specialized feature to organize adapters in the configuration store.

In This Section

- Password Sync Programming Architecture
- Adapter Programming Administration
- Adapter Programming Configuration
- Adapter Groups and Group Adapters

Password Sync Programming Architecture

A password sync adapter uses a pull model for interacting with the rest of the Enterprise Single Sign-On system: that is, the adapter actively receives password changes from the Enterprise Single Sign-On (ENTSSO) service as well as the non-Windows system. Similarly, the adapter pushes password changes received from one system to the other. With this model, your adapter interacts with three architectural components: the ENTSSO architecture, the Password Sync (PS) Helper component, and a specified non-Windows system.

Enterprise Single Sign-On architecture

ENTSSO is the service that implements the Enterprise Single Sign-On technology, and runs as a local service on the same system as your adapter. Therefore, communication between the adapter and ENTSSO is always local. However, a password sync adapter runs in a separate process from the ENTSSO service.

ENTSSO uses the configuration store to store configuration information for an adapter. The Application Users account of a configuration store application corresponds to the access account. When an adapter calls into ENTSSO, ENTSSO checks that the adapter is within the configured access account for that adapter. The access account must be a (local or domain) group account.

Because ENTSSO stores information about an adapter, the adapter can identify itself to ENTSSO by its adapter name. The adapter name corresponds to the configuration store application name and the configuration store identifier used to store the adapter properties. Therefore, adapters must know only their adapter name to access configuration information and to correctly identify themselves to the ENTSSO system at run time.

Password Sync Helper

Password Sync (PS) Helper is a COM component provided by ENTSSO. PS Helper runs in-process with the password sync adapter, and exposes ISSONotification. Your adapter can call ISSONotification to communicate with the ENTSSO service. The PS Helper passes communications to and from ENTSSO using encrypted (packet privacy) lightweight remote procedure call (LRPC). While the communications are mainly for password updates, you can also use ISSONotification to pass other types of notifications between the adapter and ENTSSO. Because PS Helper runs as a singleton value per process, it is possible for multiple adapters to call the same PS Helper object from within the same adapter process. For more information on using PS Helper programmatically, see Synchronizing a Password.

Non-Windows system

The non-Windows system is the remote computer your adapter interacts with. How you interact with the non-Windows system is up to you.

Adapter Programming Administration

An adapter is a special type of configuration store application: that is, an adapter is a component that shares a namespace with other Single Sign-On and configuration store applications. Therefore, you can access information about an adapter using ISSOConfigStore. Unlike a configuration store application however, you do not perform administrative functions on an adapter with the ISSOAdmin interface. Instead, you administer an adapter through ISSONotification. The reason for a specialized adapter administration interface is so that the system can coordinate other activities with the configuration store.

Adapter Programming Configuration

Every type of password sync adapter has different configuration requirements, depending on what non-Windows system you design the adapter for. Like affiliate applications, you can use the Enterprise Single Sign-On configuration store to store configuration properties centrally and securely. As with other configuration store applications, an administrator can use the Enterprise Single Sign-On management user interface to locate and read an XML file that describes the configuration properties for your adapter. The management tools use the XML file to render a property page to gather the required property values, for the specified adapter. You can also use ISSOConfigStore to load and read a XML name/value combinations to and from the configuration store, or you can use the SSOPS tool. You can also use the Enterprise Single Sign-On administration tools to enable and disable an adapter. On initial creation, an adapter is disabled.

Adapter Groups and Group Adapters

An adapter group is an administration mechanism you can use to collect and organize a set of adapters. In contrast, a group adapter is a component that services all adapters in an adapter group. For example, you may write a set of adapters that all use the same COM component to transmit password synchronizations over TCP/IP. Your set of adapters is called the adapter group, while the component that services them all is called a group adapter. Adapter groups are described in the configuration store. You can retrieve information and updates on an adapter group using ISSONotification.ReceiveNotification.

A group adapter has the same name as the adapter group. Other than the naming restriction, a group adapter is otherwise identical to a normal adapter. For example, a group adapter can have can have independent access groups and configuration properties, as described by its configuration file. A group adapter will most likely be on the same computer as any adapters in the adapter group. However, this is not currently enforced. Likewise all adapters in the same adapter group may be expected to be on the same computer.

Using InitializeAdapter, you can access and initialize a group adapter during startup. When you initialize a group adapter, the system informs the group adapter of all adapters in the adapter group on the current system. In addition, the system will send notifications to the group adapter any time an adapter is added, deleted, enabled, or disabled in the adapter group. However, the group adapter does not receive any password change notifications.

Adapter groups and group adapters are optional using the Administration tool.

What you Need To Know Before Programming Single Sign-On

In order to use this documentation effectively, you should be familiar with the following:

- Microsoft Windows Server ™ 2003, Microsoft Windows® XP, Microsoft Windows 2000 Server operating systems. You should be particularly familiar with Windows Security features.
- Administrative features of Enterprise Single Sign-On, especially how to perform administrative actions using the user interface.

Depending on the API and development used, you should also be familiar with the following interfaces:

- COM
- .NET Framework and the common language runtime
- Windows Networking, and in particular how to send and receive tickets

Supported Platforms for Single Sign-On

Enterprise Single Sign-On ships with Microsoft® BizTalk Server 2006, and as such is supported by all platforms that support it as well.

Programming with Enterprise Single Sign-On

Enterprise Single Sign-On is a technology implemented in the Enterprise Single Sign-On service (ENTSSO) that enables users to access multiple remote services without having to work their way through additional sign-on screens.

In This Section

- How to Determine Current Single Sign-On Access
- How to Configure Single Sign-On
- How to Create and Describe an Application to Single Sign-On
- How to Map Single Sign-On Credentials
- Logging on to a Remote or Local Application
- How to Issue and Redeem a Single Sign-On Ticket

- Synchronizing a Password

How to Determine Current Single Sign-On Access

One of the first tasks you may need to perform for a user is to determine what affiliated applications have already been set up for the current user. You can perform this query with a call to ISSOMapper.GetApplications.

⊟To query the Single Sign-On database for the applications available to the current user

1. Create a new instance of ISSOMapper.

   In general, ISSOMapper is an interface designed to retrieve information from SSO. You will most likely be using ISSOMapper in many similar queries.

2. Retrieve all of the applications affiliated with the current user with a call to ISSOMapper.GetApplications.

   ISSOMapper.GetApplications automatically returns only the affiliated applications of the current user.

How to Configure Single Sign-On

Before accessing Enterprise Single Sign-On, you should make sure that Enterprise Single Sign-On is set correctly for the current user. For most configurations, you are using one of two interfaces. ISSOAdmin is the general administration interface that enables you to create new affiliation applications. However, by using ISSOAdmin.GetGlobalInfo and UpdateGlobalInfo, you can set a variety of flags and administration values. One possible task, as described below, is ensuring that SSO ticketing has been enabled.

To enable ticketing

1. Create a new instance of ISSOAdmin.
2. Retrieve the current settings through ISSOAdmin.GetGlobalInfo.

   If necessary, you may wish to confirm that the flags are set to the correct values at this point.

3. Change any relevant flags using ISSOAdmin.UpdateGlobalInfo.

   In this particular case, all the flags are being set to validate and enable tickets. Note that Microsoft Host Integration Server currently supports only the SSO_FLAG_ENABLED. For a complete listing of all the available flags for SSO, see Enterprise Single Sign-On Flags.

How to Create and Describe an Application to Single Sign-On

Another common administrative task you may need to perform is adding an affiliate application into the Enterprise Single Sign-On database. Adding an affiliate application to the Enterprise Single Sign-On database enables you to associate users and credentials with the affiliated application. Note that creating an affiliated application requires administrative privileges, and can be done through the user interface as well.

To create and describe an application in the SSO database

1. Create a new ISSOAdmin object.
2. Create a new application with a call to ISSOAdmin.CreateApplication.
3. Add the relevant fields describing the application with a call to ISSOAdmin.CreateFieldInfo.

   It is during this step that you tell the database that an application has users and associated passwords.
4. Push the newly created description out to the server with a call to ISSOAdmin.UpdateApplication or ISSOAdmin2.UpdateApplication2.

   The difference between the two methods is that UpdateApplication2 uses an IPropertyBag as the way to describe the application updates, while UpdateApplication has multiple parameters.
5. Purge the local cache for the changes you made using a call to ISSOAdmin.PurgeCacheForApplication.

   Purging the local cache is a security measure that prevents having the names and passwords that you describe in step three to exist in an unsecured location.

How to Map Single Sign-On Credentials

Once you know you have affiliated applications in your Enterprise Single Sign-On database, you can map the credentials for a user to that application. Mapping the credentials of the current user to an affiliated application requires you use a combination of the ISSOMapper and ISSOMapping interfaces.

To map between an affiliated application and user credentials

1. Create new instances of ISSOMapper and ISSOMapping.
2. Set the ISSOMapping properties to the relevant values.

The relevant properties for ISSOMapping are the Microsoft® Windows® domain name of the user, the Windows user name, the name of the affiliated application, and the logon name the user goes by for the affiliated application.

3. Make sure the mapping exists using a call to ISSOMapping..

Calling ISSOMapping.Create propagates the local copy of mapping out to the Enterprise Single Sign-On server.

4. Set the credentials on the mapping with a call to ISSOMapper.SetExternalCredentials.

It is necessary to set the credentials with a separate call due to security reasons.

5. Enable the mapping with a call to ISSOMapping.Enable.

Calling ISSOMapping.Enable exposes your created map to the enterprise. Without calling this step, your map would remain private on the server, and thus unusable.

Logging on to a Remote or Local Application

After you have finished setting affiliate credentials for your user, you can use Enterprise Single Sign-On service (ENTSSO) to provide access to applications. If the user is local, you can use ENTSSO to retrieve credentials for a non-Windows application. In contrast, if the user is remote, you can use ENTSSO to retrieve credentials for a local application.

In This Section

- How to Log a Local User on to a non-Windows Application
- How to Log a Remote User on to a Local Application

How to Log a Local User on to a non-Windows Application

Once you have set up your user with an affiliate application, you can use Single Sign-on to access the external user name and credentials of the current user. Using these credentials, you can then log your user on to the affiliate application running on a host server.

To log a local user on to a host application

1. Receive the request to log the current user on to an application running on the host server.

It is your responsibility to determine how the current user requests to be logged on to a host application.

2. Retrieve the credentials for the current user using ISSOLookup1.GetCredentials or ISSOLookup2.GetCredentials.

   You must supply the name of the host application along with any relevant flags. GetCredentials returns the associated username and credentials for the host application.

   Note that you may use either ISSOLookup1 or ISSOLookup2. The only difference is that ISSOLookup2 also has a method for logging a remote user on to a local windows application.

3. Use the external user name and credentials to log on to the host application.

   It is your responsibility to determine how to use the credentials to log on to the host application.

How to Log a Remote User on to a Local Application

The other main feature of Enterprise Single Sign-On service (ENTSSO) is supporting a host-initiated process (HIP). ENTSSO interacts with HIP when a remote user attempts to access a local Windows resource. Using ENTSSO, you can receive the request from the host user and request access to the local Windows application.

To log a remote user on to a local Windows application

1. Receive the request from the remote user.

   It is your responsibility to determine how to retrieve a request from the remote user.

2. Request that the remote user be given access to the specified affiliate application, using IssoLookup2..

   LogonExternalUser passes in the name of the application the external user wishes to access, the name of the user, the associated credentials for the user, and any relevant flags. If successful, LogonExternalUser will return a Windows handle.

   Note that the remote user must already be identified in the Single Sign-On database, be associated with an affiliate application, and have his credentials on file. Otherwise, LogonExternalUser will return an error.

   You can keep the user names and credentials up to date using Password Sync.

3. Use the Windows handle returned from LogonExternalUser to grant access to your remote user.

To continue keeping your application secure, you can use SSO to issue and redeem tickets.

As with the original request to log on, it is your responsibility to determine how the remote host will use the Windows handle.

How to Issue and Redeem a Single Sign-On Ticket

Once you have linked together a user and an affiliate application, you can issue tickets to ensure security while maintaining communications. Single Sign-On ticketing works as other ticketing technologies: prior to sending the message off, you append the Single Sign-On ticket to the message as a string. The server receives your message, decodes the ticket, and uses the information as appropriate. The server then acknowledges that it received the message by sending a return ticket, which you can decode and redeem.

To issue a Single Sign-On ticket

1. Create a new ticket object with a call to.
2. Issue the ticket with a call to SSOTicket..

IssueTicket does not currently accept any valid parameters. Therefore, you must assign the returned encoded string only to the relevant message. The affiliate application you send the message to uses the encoded Single Sign-On ticket to determine to what parts of the application you have access.

To redeem an Single Sign-On ticket

1. Create a new ticket object with a call to SSOTicket.
2. Redeem the ticket with a call to SSOTicket.RedeemTicket.

Synchronizing a Password

Synchronizing a password is performed through a password sync adapter. A password sync adapter is an application you create that is registered with the configuration store. This application ought to be capable of communicating with a specific, remote, non-Windows system, and should also be capable of instructing that system to update password information. Once you have created the adapter, you must register the adapter with the system, and then associate the adapter with the applications that the adapter interacts with. Finally, you can add an adapter to an adapter group which allows you to more effectively organize permissions.

In This Section

- How to Create a Password Sync Adapter
- How to Register and Configure a Password Sync Adapter

- How to Assign an Application to an Adapter
- How to Create and Modify an Adapter Group

How to Create a Password Sync Adapter

A password sync (PS) adapter is an application that uses the Password Sync Helper component to pass notifications to and from Enterprise Single Sign-On. Note that although the PS Helper component exposes a COM interface, your adapter does not necessarily have to be a COM component. You can design your adapter as a stand-alone process, a COM+ application, or a Windows service.

To create a password sync adapter

1. Inform Enterprise Single Sign-On service (ENTSSO) that your provider is active using ISSONotification.InitializeAdapter.

   InitializeAdapter informs ENTSSO that a provider, usually the same provider that is making the call, is currently turned on, and therefore will be communicating password updates to and from the system. You can also use InitializeAdapter to activate other resources such as group adapters.

2. Send password updates to ENTSSO using ISSONotification.SendNotification.

   It is up to you to determine how you receive password updates from your non-Windows system. Once you have received the update, you can pass the information on to ENTSSO using SendNotification. Note that SendNotification is not limited to sending password updates: the architecture of SendNotification allows you to send other types of notifications as well.

3. Request password updates from ENTSSO using ISSONotification.ReceiveNotification.

   Because the password sync adapter is a pull technology, ENTSSO will never call your adapter. Instead, your adapter will periodically call ReceiveNotification to see if any password updates are available. You can choose to set the WAIT flag on ReceiveNotification. Setting WAIT will block the thread until a notification is available.

   Note that ENTSSO delivers a password change to your adapter in plain text. It is the responsibility of the adapter to protect that password information against incorrect disclosure. It is also the responsibility of the adapter to protect itself against spoofing or attacks from other invalid sources, including spoofing of the Password Sync Helper component.

   Once you receive a password update from ENTSSO through the pReceiveNotification parameter, you must pass this information on to your non-

Windows system. As with SendNotification, it is up to you to determine the best means of communicating with the remote server.

4. Turn off your adapter using ISSONotification.ShutdownAdapter.

ShutdownApplication should be the last method called by an adapter, and indicates that the adapter will no longer be sending or receiving password updates to ENTSSO.

Note that ENTSSO buffers any password changes a user makes while the adapter is shut down, up to a buffer size limit.

How to Register and Configure a Password Sync Adapter

Once you are finished creating your password sync adapter, you must load your adapter on to a system. Further, you must inform Enterprise Single Sign-On and the configuration store that your application is a password sync adapter. You must register with the configuration store for security purposes: your adapter will request updates to passwords and other credentials. Therefore, ENTSSO must know that a given adapter is allowed to ask for such permissions.

To configure a password sync adapter

1. Create your adapter with the configuration store using the ssops tool.

   Using ssops, you load an XML configuration file into the config database that describes your application to Enterprise Single Sign-On.

2. Once you coreate the adapter with the config database, you can modify the adapter information with ISSOConfig.SetConfigInfo or ISSOPSAdmin.SetAdapterProperties.

   While the two methods are similar, SetAdapterProperties sends a message to the adapter when the configuration information changes.

3. To delete an adapter, use ISSOConfig.DeleteConfigInfo.

How to Assign an Application to an Adapter

In order to process information between a local application and a remote server, an adapter must have one or more applications assigned to it.

To assign an application to an adapter

1. Add the application to the adapter with ISSOPSAdmin.AssignApplicationToAdapter.

2. You may retrieve the current list of applications assigned to an adapter using ISSOAsmin.GetApplicationsForAdapter.

3. Once you are finished, you may remove an application from an adapter with ISSOPSAdmin.RemoveApplicationFromAdapter

How to Create and Modify an Adapter Group

One of the new features of SSOv3 is the ability to create and modify adapter groups. As the name implies, an adapter group is a collection of adapters. You can use adapter groups to organize security settings and other properties for your adapters.

To create and modify an adapter group

1. Create the adapter group with a call to the ssops tool.

   In the associated xml file, you must set the group flag as an adapter group.

2. Add one or more adapters to the adapter group with ISSOPSAdmin.AddAdapterToAdapterGroup.

   At this point, your adapter group is ready to work as intended. If necessary, you may view the full list of associated adapters with ISSOPSAdmin.GetAdaptersForAdapterGroup.

3. You may modify the settings of your adapter group using ISSOPSAdmin.SetAdapterProperties.

4. Once you are finished, you may choose to remove the adapter from the adapter group using ISSOPSAdmin.RemoveAdapterFromAdapterGroup.

5. Finally, you may delete the adapter group with ISSOAdmin.DeleteApplication.

   You may choose instead to delete the adapter group using the _delete command of the ssops tool.

Viewing Assemblies with the BizTalk Assembly Viewer

When you are developing applications or deploying assemblies, you need to see which assemblies are deployed and available to you. Microsoft BizTalk Server 2006 provides a convenient tool for this purpose - BizTalk Assembly Viewer.

The BizTalk Assembly Viewer is a Windows shell extension for examining deployed BizTalk assemblies and assembly types. BizTalk Assembly Viewer enables you to:

- View a list of all BizTalk Server assemblies installed in the global assembly cache (GAC) on your local computer.

- View the types within a BizTalk Server assembly.
- View attributes of a BizTalk Server assembly, such as name, version, culture, code base, and public key token.
- View referenced assemblies for a BizTalk Server assembly.
- Add assemblies in the GAC.
- Remove assemblies from the GAC.
- View the XML code for various types including schemas, maps, orchestrations, and pipelines.
- Search for particular types across all BizTalk Server assemblies deployed on a given server.
- Search type dependencies across BizTalk Server assemblies.

In This Section

- How to Register and Remove the BizTalk Assembly Viewer
- How to View Assemblies and Types on the Local Server

How to Register and Remove the BizTalk Assembly Viewer

The BizTalk Assembly Viewer is not registered automatically during BizTalk Server setup. To register or remove the BizTalk Assembly Viewer, follow these steps.

To add Assembly Viewer to the Windows registry

1. From the **Start** menu, click **Run**.
2. In the Run dialog box, type **cmd**.
3. From the command line, navigate to *<BizTalk Server Installation Folder>*\Developer Tools\ where BTSAsmExt.dll resides.
4. At the command line, type:

   ```
   regsvr32 BTSAsmExt.dll
   ```

5. To begin using Assembly View, log off and then log back onto the computer.

To remove Assembly Viewer from the Windows registry

1. From the **Start** menu, click **Run**.
2. In the **Run** dialog box, type **cmd**.
3. From the command line, navigate to *<BizTalk Server Installation Folder>*\Developer Tools\ where BTSAsmExt.dll resides.
4. At the command line, type:

   regsvr32/u BTSAsmExt.dll

5. To complete the removal, log off and then log back onto the computer.

How to View Assemblies and Types on the Local Server

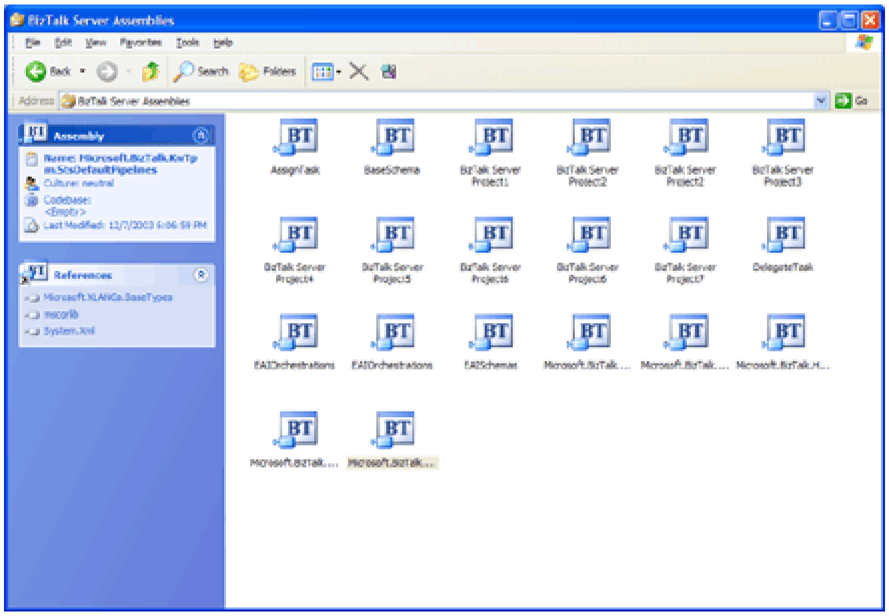
You can use the BizTalk Assembly Viewer to examine all BizTalk assemblies and types installed on the local server.

To view all BizTalk Server assemblies installed in the GAC

1. To open the BizTalk Assembly Viewer, double-click **My Computer**, and then double-click **BizTalk Server Assemblies**.

   All BizTalk assemblies installed in the global assembly cache (GAC) on the computer appear.

   Assembly Viewer open page



Name: Microsoft.BizTalk.KwTp m.ScsDefaultPipelines
Culture: neutral
Codebase: <Empty>
Last Modified: 11/7/2003 6:06:59 PM

Microsoft.XLANGs.BaseTypes
mscorlib
System.Xml

To view types, attributes, and references for a BizTalk assembly

1. To open the BizTalk Assembly Viewer, double-click **My Computer**, and then double-click **BizTalk Server Assemblies**.
2. Double click the appropriate assembly.

   The BizTalk assembly types appear in the right pane. The attributes and types appear in the left pane.

   You can also navigate to the installed location of the assembly by clicking the **Codebase** link in the upper left section of the task pane. (You cannot view this link when Windows Explorer is in Folder view because the task pane is replaced by the folder list.)

To view the contents of a type in a BizTalk assembly

1. To open the BizTalk Assembly Viewer, double-click **My Computer**, and then double-click **BizTalk Server Assemblies**.
2. Double click the appropriate assembly.
3. In the right pane, double-click the appropriate type.

   The **Type Content Viewer** window opens and the XML definition appears.

To add a BizTalk assembly in the GAC

1. To open the BizTalk Assembly Viewer, double-click **My Computer**, and then double-click **BizTalk Server Assemblies**.
2. Use Windows Explorer to navigate to the assembly that you want to add in the GAC.
3. Drag the assembly and drop it into the BizTalk Assembly Viewer.

   The BizTalk Assembly Viewer accepts only BizTalk Server assemblies. If you drop a non-BizTalk assembly in the viewer, it is ignored.

To remove a BizTalk assembly from the GAC

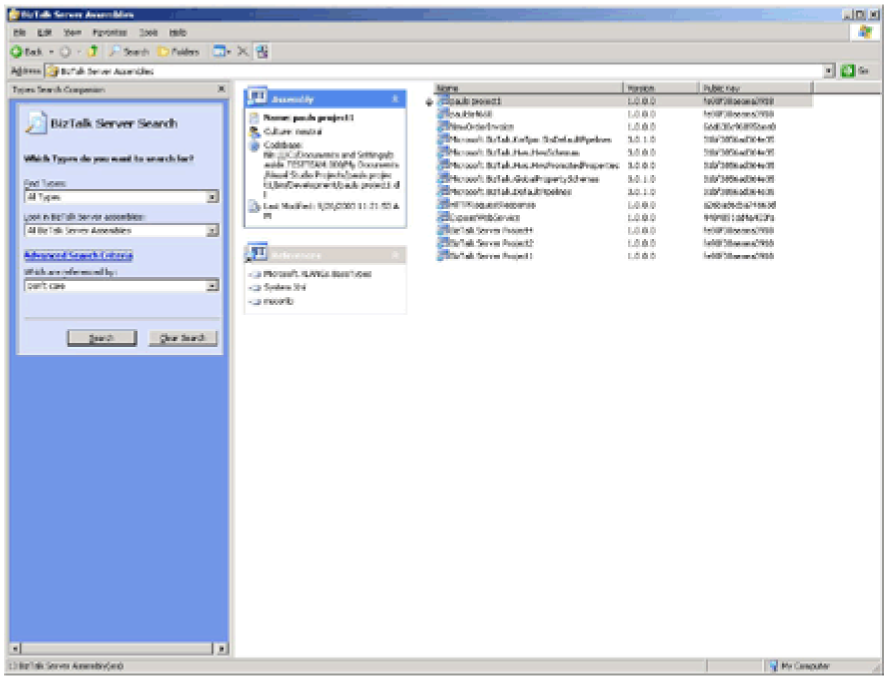
1. To open the BizTalk Assembly Viewer, double-click **My Computer**, and then double-click **BizTalk Server Assemblies**.
2. Right-click the assembly you want to remove from the GAC and click **Delete**.

To search for a type in all BizTalk assemblies

1. To open the BizTalk Assembly Viewer, double-click **My Computer**, and then double-click **BizTalk Server Assemblies**.
2. From the menu bar, click the **Biz Talk Server Search** icon.
3. In the Search window, specify the type in the **Find Types** drop-down list.
4. In the **Look in BizTalk Server assemblies** drop-down list, select the assemblies to be searched.
5. To narrow your search to include only types that are referenced by a specified type, do the following:
   a. Click the **Advanced Search Criteria** link.
   b. In the **Which are referenced by** drop-down list, select the type.
   c. Click **Search**.

   The specified types appear in the search results.

BizTalk Assembly Types



Which Types do you want to search for?

Find Types: All Types

Look in BizTalk Server assemblies: All BizTalk Server Assemblies

Advanced Search Criteria

Which are referenced by: Don't care

Name: pauls project1

Culture: neutral

Codebase: file:///C:/Documents and Settings/p aulale TESTTEAM 000/My Documents /Visual Studio Projects/pauls projec t1/bin/Development/pauls project1.d ll

Last Modified: 9/29/2005 11:21:53 A M

References: Microsoft.XLANGs.BaseTypes; System.Xml; mscorlib

Name	Version	Public Key
pauls project1	1.0.0.0	[illegible]
[illegible]	1.0.0.0	[illegible]
NewOrderInvoice	1.0.0.0	[illegible]
Microsoft.BizTalk.[illegible].DefaultPipelines	3.0.1.0	[illegible]
Microsoft.BizTalk.Hws.HwsSchemas	3.0.0.0	[illegible]
Microsoft.BizTalk.Hws.HwsPromotedProperties	3.0.0.0	[illegible]
Microsoft.BizTalk.GlobalPropertySchemas	3.0.1.0	[illegible]
Microsoft.BizTalk.DefaultPipelines	3.0.1.0	[illegible]
HTTPRequestResponse	1.0.0.0	[illegible]
ExposedWebService	1.0.0.0	[illegible]
BizTalk Server Project4	1.0.0.0	[illegible]
BizTalk Server Project2	1.0.0.0	[illegible]
BizTalk Server Project1	1.0.0.0	[illegible]

13 BizTalk Server Assembly(ies)

Creating Custom Components

Custom components in BizTalk Server are developed and used to modify or extend certain infrastructure elements of BizTalk Server. Primarily, this includes such things as: send or receive adapter components which are based on the Adapter Framework and used to handle specific transport protocols; pipeline components used in any of the send or receive pipeline stages; and functoids used in maps. You can extend BizTalk Server 2006 with custom code in several ways. The following topics describe how to do this.

In This Section

- Creating Custom Adapters Using the Adapter Framework
- Creating Custom Pipeline Components
- Creating Custom Functoids

Creating Custom Adapters Using the Adapter Framework

To exchange messages with external systems, applications, and entities BizTalk uses the concept of an adapter. Adapters are COM or .Net-based components that transfer messages to and from business end points (such as file systems, databases, and custom business applications) using various communication protocols. Biztalk contains eight native adapters out of the box that support various protocols. These include:

- A FILE adapter, which supports sending and receiving messages from a File location BizTalk Message Queuing adapter
- Adapters for EDI, FTP, HTTP, MSMQ, SMTP, or SOAP protocols
- An adapter for the SQL database which exchanges data with BizTalk using XML

In some cases BizTalk may need to transport messages to a specific custom application or use a protocol for which an existing native adapter does not exist. Third party companies have written adapters to support additional protocols and you certainly may want to exhaust that search before deciding to write a custom adapter to support your unique requirements. For a list of adapters and associated vendors go to http://www.microsoft.com/biztalk/partners. However if you are unable to locate an adapter to support your communication requirements, the option exists to develop your own custom adapter.

The duty of writing a custom adapter can be challenging. To simplify this process Microsoft has developed a foundation called the Adapter Framework. This framework can be used as a basis for your development along with sample adapter source code also found in the BizTalk product SDK.

The following topics present development tips and recommendations in the hope that these will help adapter developers avoid common problems.

This section contains:

- What is a Custom Adapter?
- The Adapter Hosting Model
- Adapter Message Exchange Patterns
- Planning for Custom Adapters
- Introducing the Adapter Framework
- Adapter Development with the Adapter Framework
- Deploying a Custom Adapter
- Using the Adapter Framework Tools
- Modifying the Design-Time Configuration of the Sample File Adapter

What is a Custom Adapter?

Developing a custom adapter can be a challenging undertaking. In order to make this process as streamlined and productive as possible, it is best to first obtain an understanding of the concepts and functionality requirements of an adapter. In particular:

- The types of adapters, messages and transfer operations, and the artifacts supporting this process
- How an adapter batches messages in groups and leverages transactions to ensure proper message reception and transmission via its protocol
- Using the Adapter Framework to provide a base from which to build your custom adapter component

Terminology

The following terms are used throughout this section to describe the functions of BizTalk Server adapters.

Operations, Messages, Batching, and Transactions

Adapters are used by Biztalk Server to exchange messages with external entities via send and receive operations

- Send (or send side) operations occur when information is being sent by BizTalk Server to an external entity using the protocols supported by the adapter.

- *Receive (or receive side)* operations occur when the adapter receives information from an external entity and passes it into BizTalk Server.

Adapters in BizTalk Server perform a number of operations when receiving and sending messages. A majority of an adapter's functions relating to message batching and transfer requires interaction with the BizTalk Messaging Engine. The IBTTransportBatch interface exposes that functionality provided by the BizTalk Server Messaging Engine. Most of these operations exposed by this interface are listed as follows:

Receive side operations:

- *One-Way Submit:* void SubmitMessage(IBaseMessage msg). After receiving a message from a receive port, the adapter submits it to BizTalk to be processed by a subscribing orchestration or send port.

- *Suspend:* void MoveToSuspendQ(IBaseMessage msg)**.** When it determines a parsing, transmission, serialization, or other applicable failure has occurred after submission, an adapter can move a message to the suspended queue.

- *Submit Request:* void SubmitRequestMessage(IBaseMessage requestMsg, string correlationToken, [MarshalAs(UnmanagedType.Bool)]bool firstResponseOnly, DateTime expirationTime, IBTTransmitter responseCallback). A receive adapter submits an incoming message to BizTalk Server in a request-response pair. After BizTalk successfully processes this request message it sends the response to the adapter to transmit it to the specific entity. The adapter waits on a response.

Send side operations:

- *Resubmit: void* Resubmit(IBaseMessage msg, DateTime timeStamp). After a transmission failure occurs on a message, an adapter can resubmit it when appropriate. This is called on a per message basis. If a batch of messages was submitted unsuccessfully, the adapter must determine the messages causing the failure, and resubmit the ones which did not cause the batch to fail in separate calls to Resubmit.

- *Move to Next Transport***:** void MoveToNextTransport(IBaseMessage msg). If a message fails during a send operation and its retry attempts have been exhausted, the adapter can send the message to the next configured transport for retransmission.

- *Suspend:* void MoveToSuspendQ(IBaseMessage msg)**.** The adapter moves a failed send message to the suspended queue if no additional backup transport is configured

- *Delete:* void DeleteMessage(IBaseMessage msg) The adapter deletes a message after being notified by BizTalk of its successful transmission. Deleting a message tells BizTalk Server that the adapter is done with the message. Generally the SubmitResponse operation is done in the same batch as its associated Delete operation.

- *Submit Response:* void SubmitResponseMessage(IBaseMessage solicitMsgSent, IBaseMessage responseMsgToSubmit). The adapter submits a response to the batch to be sent back to BizTalk. This operation includes the original message in the call along with the response so that BizTalk Server can correlate the two of them.

- *Cancel Response*: void CancelResponseMessages(string correlationToken). Should the need arise to cancel the sending of a response message before the batch is submitted, the CancelResponseMessages method is used passing in the correlation token for the associated response message to be deleted.

Messages

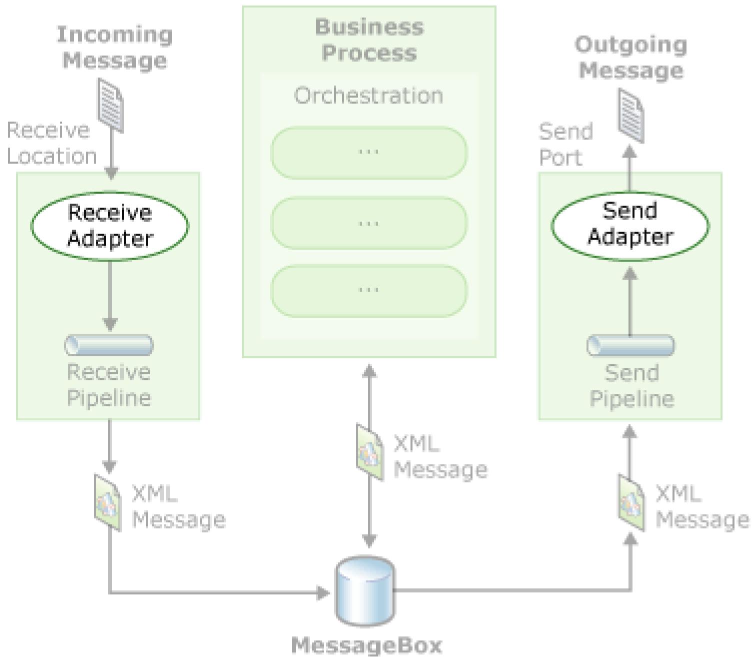
Any packet of data going in or out of BizTalk is a message. Examples include HTTP posts, FTP files, SMTP e-mail, MQSeries messages, and SQL result sets. The adapter is agnostic with respect to format of the data. It creates a BizTalk formatted message structure and blindly streams the raw and unmodified data into the message body. It is the job of the dissembler stage of the receive pipeline to convert that data into XML so BizTalk can work with it.

A message is made up of a message context section and a body section:

- The message context is composed of simple name-value pairs associated with the message that are not stored a part of the body. It is sometimes referred to as the message's "meta-data". The names are all drawn from various property schemas. The system provides some core schemas and BizTalk Server applications can add their own. The context is copied into memory when a message is processed.

- The body of a message is the actual data being sent. For instance, suppose we have a text file containing 10 rows of comma-delimited employee records. The 10 rows in that file will make up the body of a message, which itself can have one or more parts. In this case each employee record, or part, is stored as a stream of bytes. The body of the message is never copied into memory at one time.

The messaging process



This diagram illustrates how data is transformed into a message and sent through BizTalk. Here we are submitting one message at a time and not using the concept of batched messaging (discussed shortly):

- A **receive location** works in conjunction with a receive adapter to receive data from the outside world. When the adapter receives the data, it creates a new BizTalk message and connects the incoming stream of data to that message structure.

- The message is passed to the BizTalk messaging engine which finds a free thread in its thread pool to process that message.

- The BizTalk messaging engine passes it to the receive pipeline specified by the **receive location which originally accepted the input data**.

- The receive pipeline normalizes and validates the XML data and the message sender is authenticated. The message is then handed off to the **receive port**.

- The **receive port** manages any desired mapping, and publishes the message to the MessageBox. The MessageBox is a Microsoft SQL Server™ database containing messages to be sent to subscribers. Both orchestrations and send ports can subscribe to messages placed into the MessageBox database.

- If the message context properties match the specifications set in the filter on the subscriber (the orchestration or the send port), BizTalk sends the message to the subscriber. If the send port is the subscriber, the message goes through the reverse of its input processing on its way out of BizTalk. It is passed through the messaging engine, the send pipeline, and to a send port before being transmitted over the wire by a send adapter.

- If no subscriber is found the message is placed into the suspended queue.

Both native adapters and custom adapters built on the Adapter Framework use this message processing paradigm.

Batches

When your adapter has a group of messages that need to be processed at one time, you should *batch* these operations to optimize performance. Programmatically, *message batches* are collections of messages with an associated operation. By grouping messages in a batch rather than submitting each individually you optimize the use of resources and processing tasks. Conservation of BizTalk Server threads, database access, and transaction overhead all benefit from the use of batching messages.

Transactions

Batches are associated with *transactions*. For every batch, there is a single transaction. All batches inside of BizTalk Server are considered transactions. The adapter may create an explicit external transaction and associate it with the batch (in which case we call it a *transactional batch*). Or it can leave it to BizTalk Server to manage the transaction internally by providing a NULL transaction object with the batch (in which case we call it a *normal batch*). BizTalk Server itself creates an internal batch object in this case, but this batch object will not be accessible to the adapter. Internal transactions are visible only within BizTalk Server and cannot be referenced externally.

A transactional batch is a batch for which the adapter explicitly provides the transaction object for BizTalk Server to use.

For the purposes of this paper, an external transaction is a true Microsoft Distributed Transaction Coordinator (MSDTC) COM+ transaction.

Transactional adapter

A *transactional adapter* is an adapter that makes use of transactional batches by explicitly creating an external MSDTC transaction. A *non-transactional adapter* uses non-transactional batches allowing BizTalk Server to implicitly create its own internal transaction.

Receive Location

BizTalk uses the concept of a receive location to associate an adapter with a receive port and a receive pipeline component. This receive location also contains connection-specific information that the adapter needs to connect to a particular location.

The Adapter Framework

The following figure shows how an adapter and the Adapter Framework work together to connect your application to BizTalk Server.

1. Data is received from the wire through a receive location that is listening for messages of a certain protocol at a specified address. The receive location is associated with an adapter and a receive pipeline. You can configure both the adapter and the pipeline components to perform certain logic on messages of a predetermined protocol.

2. After the message is received by the receive location, the message is sent to the adapter, which creates a new BizTalk message, attaches the data stream to the message (typically in the body part of the message), adds any metadata pertaining to the endpoint over which the data was received, and then submits that message into the Messaging Engine.

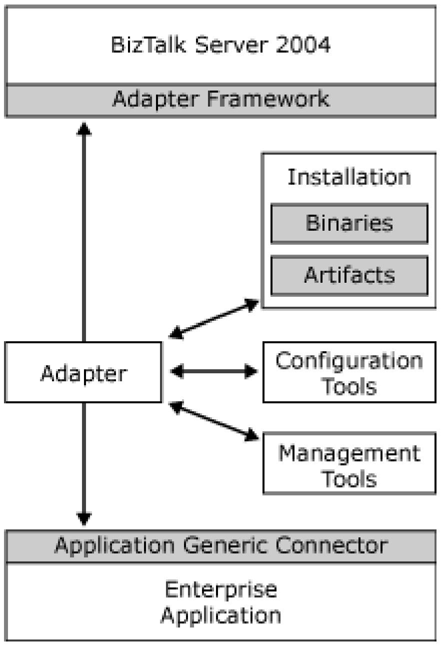
3. The Messaging Engine sends the message to the receive pipeline where the data is transformed into XML, the message sender is authenticated, the message is decrypted, and the XML is validated.

4. The Messaging Engine publishes the message to the MessageBox. The MessageBox is a Microsoft SQL Server™ table containing messages to be processed. Both orchestrations and send ports can subscribe to the MessageBox.

5. The Messaging Engine sends the message to either an orchestration or a send port subscriber based upon the message context properties matching the specifications set in the filter on the subscriber.

6. If an orchestration is the subscriber, it processes the message and sends it out via a send port. Once the send port has it, or is the only subscriber, the message passes through the send pipeline into a send adapter before being transmitted over the wire

The Adapter Framework



The Structure of an Adapter

An adapter shares the standardized configuration, management, and setup mechanisms used by the native adapters. With the standardization to the Adapter Framework, a custom adapter is managed via two applications regardless of how many custom adapters you create. Those applications are the BizTalk Server Administration Console for adapter managment and the BizTalk Explorer for adapter configuration. Previously each custom adapter had unique applications for configuration, management, and setup—three new applications to learn with each adapter. If you had three custom adapters you would need nine different supporting applictions.

The figure below illustrates the three main elements of a custom adapter: the Adapters Registry file, the Adapters design-time component and the Adapters run-time component.

Adapter Registry file

Adapters require certain information about them to be registered in the registry and the BizTalk Server Management database. Information such as an adapter's alias, receive hander, receive location, transport type, etc. is called 'meta data'. These meta data entries are created during manual adapter registration using the BizTalk Server Administration console. Alternatively, you can run the Adapter Registration Wizard (AdapterRegistryWizard.exe) SDK utility to generate a registry file for your custom adapter. Double-clicking on this registry file or selecting Import from the File meu using the registry editor (regedit32.exe) will write the adapters' meta data into the both the registry and the BizTalk Server Management database.

Design Time Component

The user interface for a custom adapter is implemented using the Adapter Framework. This provides a very productive approach for UI development since the UI is rendered from an XML schema provided as part of the adapter's assembly. A small amount of code is required to be written to transform the contents of the schema into a user interface to configure the adapter's properties.

Run Time Component

Typically an adapter will consist of two public run-time components; the component that implements the message receiver and the component that implements the message sender. This may be deployed all in one, or in two separate, assemblies.

Receive Adapter

Receive adapters are responsible for creating a new BizTalk message by attaching the network/data source stream to the message body. It also adds any meta data pertinent to the endpoint over which the data was received, then submits that message to the Messaging Engine. The adapter will delete the data from the receive endpoint or send the appropriate acknowledgement message to the client indicating that the data has been accepted into BizTalk.

Send Adapter

Send adapters are responsible for sending a BizTalk message to the specified endpoint using its specific transport protocol.

Transport and Application Adapters

Adapters can also be loosely categorized as Transport Adapters and Application Adapters.

Transport Adapters

Transport Adapters are typically concerned with moving the data to and from the wire using a specific transport protocol. Examples of these would be HTTP, FILE, FTP, and TCP adapters.

Application Adapters

Application Adapters typically perform the role of integrating a back end system with BizTalk server. These are developed with performance in mind but here the problem domain is subtly different. Meta data from the application may also flow to and from BizTalk Server. Application Adapters typically use a schema wizard at design time to dynamically ‘mine’ the schema from the application. Examples of Application Adapters are SQL, SAP, and PeopleSoft.

The Adapter Hosting Model

In general BizTalk adapters are hosted in the BizTalk service process, Btsntsvc.exe. This means that BizTalk manages the lifetime of the adapter. There are also situations, described below, where other processes manage the adapter.

Regular Adapters

Adapters that are managed by BizTalk Server are called Regular Adapters. BizTalk will:

- Instantiate the adapter when BizTalk Server is started
- Pass its Transport Proxy to it during initialization
- Service the adapter's requests
- Terminate the adapter on service shutdown.

BizTalk delivers any handler configuration and the endpoint configuration information to the adapter at runtime. Other aspects of configuration are specified, such as service windows which define specific time periods in which the adapter is enabled to actively handle requests.

The BizTalk service may be manually shut down using the BizTalk Admin MMC or using the service control manager. If connectivity to the BizTalk databases is lost the service will automatically re-cycle itself.

In the typical hosting model receive side adapters and send side adapters are hosted in the same process as the BizTalk Server Service, along with the Messaging Engine and the Orchestration Engine. The hosting model is flexible enough to allow for separation of receive, send and orchestration hosts and combinations of these, in the diagram below the host is executing all three in the same process.

Because of the rich hosting model, it is important when developing adapters to remember that the send and receive adapters may never be configured in the same host they may even be configured to run on different machines.

The Regular Adapter Hosting Model

Isolated Adapters

There are scenarios when hosting receive adapters in the BizTalk process is not possible. For example, the IIS process model is such that IIS manages the life time of ASP.NET applications and ISAPI Extensions. The BizTalk SOAP adapter must run within the same process space as IIS, thus making it impossible for BizTalk to control the lifetime of any instances of the SOAP adapter.

For this type of adapters there is another hosting model referred to as Isolated Receive Adapters, or simply Isolated Adapters. There is no concept of an Isolated Send Adapter.

Since Isolated Adapters cannot be created by BizTalk, it is the responsibility of the adapter to acquire its own Transport Proxy and register itself with that Transport Proxy.

The diagram below illustrate the BizTalk hosting architecture. For the sake of performance, the isolated host architecture makes a conscious effort to eliminate any unnecessary inter-process communication. Since the Isolated Adapter and the BizTalk Messaging Engine stack are in the same process there is no inter-process communication when the adapter is calling the Messaging Engine. In that scenario the only inter-process communication is between the Messaging Engine and the database, which is unavoidable.

Adapter Message Exchange Patterns

The Adapter Framework supports a rich set of message exchange patterns that can be utilized by adapters to enable many powerful messaging scenarios.These message exchange patterns may effectively be 'stitched' together using messaging and crchestrations to enable complex messaging scenarios.

One-way (Asynchronous)

In this message exchange pattern, messages flow one way into BizTalk through an adapter. The Messaging Engine publishes the message into the Messagebox database. If an orchestration has an active subscription to a message of that type, the message is routed to that orchestration. After processing the message, the orchestration subsequently publishes the message back into the MessageBox database before it is routed to an adapter to be sent out over the wire. The key concept here is that messages flow in one direction. When the message is submitted into the engine, no response is expected. On the outbound side when the message is transmitted over the wire no response is expected. This is typically referred to as asynchronous messaging and is in many ways the basic building block used by the engine for all messaging scenarios.

Request-Response Style Protocols (Sync-on-Async)

A request-response scenario consists of receiving a request message processing it, and sending a response message. It is also referred to as synchronous-on-asynchronous (sync-on-async) because the underlying BizTalk architecture is asynchronous for scalability reasons. However the architecture of the BizTalk engine enables exposing a synchronous message exchange pattern on top of these asynchronous exchanges. In doing this the engine handles the complex task of correlating the request and response messages together across a scaled out architecture by linking together a number of asynchronous message exchanges in order to expose a synchronous interface.

For example, a web page that checks inventory may make a SOAP call to a BizTalk SOAP receive adapter., BizTalk orchestrates a series of Web-Services aggregating the information and returning it in one SOAP response. To the client this appears to be a synchronous SOAP call but in actuality the engine knits together a number of asynchronous message exchanges.

Solicit-Response Style Protocols

This scenario is initiated by sending a solicit message and completed by receiving a response message. It is referred to as solicit-response since the initial message sent is soliciting an endpoint for a response message. A scenario using this message exchange pattern may involve an orchestration making an outbound HTTP call (a solicit for a response) and waiting for the response.

Request-Multi Response

This scenario is very similar to the request-response scenario. However in this scenario multiple responses may be returned for a given request. The number of responses is non-deterministic from the engines standpoint. The API's allow a timeout value be specified whereby all responses receive within the timeout will be returned to the receive adapter.

Loop-Back

This scenario is very similar to the request-response scenario. The request message is published as usual, but the engine ensures the response message is routed back to the adapter. Since the request message is publish to the message box, the tracking infrastructure will ensure both the request and response messages are tracked.

An example of this is a client requiringa receipt for for a message received that is atomically returned to the adapter within the same batch containing the message that was initially published to the MessageBox. The inbound message is copied, one instance being returned to the client with the other being processed in the normal manner. This specific scenario would also require a custom XML Dissassembler to be written.

Planning for Custom Adapters

The following table lists some of the questions that you need to answer in planning for custom adapters.

Planning question	Recommendation
Why create a custom adapter?	• To communicate with an external system, a system that may be proprietary, or a system unsupported by existing adapters. • To perform custom logic on incoming or outgoing

	messages. Custom logic can be implemented in the orchestration or a custom pipeline component as well as in the adapter. • To communicate with an external system with a nonstandard communication pattern.
What software or tools are needed to create an adapter?	All you need is Microsoft® Visual Studio® 2003, and a Microsoft BizTalk® Server 2006 installation with the Development and Runtime Components. Be sure to install the SDK, which contains many sample adapters that you can use as a template to create a custom adapter.
What decisions should I make before I start coding my custom adapter?	Read about each of the adapter types and variations to determine which are appropriate for your situation. For more information about the different types of adapters, see About Adapter Variables.
Are there any adapter samples that I can study before creating my own adapter?	Review the code samples in the Adapter Samples folder of the BizTalk Server SDK. For more information, see Adapter Samples - Development.

Introducing the Adapter Framework

This section documents how to use the BizTalk Adapter Framework run-time APIs used to develop custom adapters. More detailed information on these APIs can be found in the Developers Reference in the following Managed Reference topic:**Microsoft.BizTalk.Message.Interop**.

In This Section

- IBaseMessage
- Messaging Engine Interfaces
- Adapter Interfaces
- Transmitter Interfaces
- Receiver Interfaces
- About Adapter Variables

IBaseMessage

When a receive adapter accepts an incoming data packet via its protocol, it creates a message to pass to the Messaging Engine using the IBaseMessage interface. All messages sent to the MessageBox database must implement this interface.

The IBaseMessage Structure

A message has one or more message parts represented by the IBaseMessagePart interface. Each message part has a reference to it's data via an I Stream interface pointer. The context of a message is represented by its IBaseMessageContext interface. The following diagram illustrates the BizTalk message object model.

The message context is a dictionary which is keyed on a combination of the property name and the property namespace. This prevents collisions between similarly named properties from different sources, e.g. BizTalk system properties and custom adapter properties. The values for these properties are of the .NET type **object**, but under the covers these properties are VARIANT's.

Each part has a part context which is also a dictionary but without the notion of a namespace. The value of a part context is metadata referring to the data for that part. An example of this is the Charset property Which specifies the character set used for encoding that message.

Properties may be written to, and read from, the message context. They may also promoted to be used for message routing. Being promoted means they are written as a part of metadata that flows with the message. A promoted property allows its value to be used in the creation of filter expressions on send ports. Down stream components and user code in orchestrations may read promoted properties and also write new values to them.

Once a promoted property has been matched to an existing subscription and used to route a message, the property is demoted to prevent cyclic subscription matches. A demoted property remains on the message context as metadata but loses its promoted status.

Implementation Tip: Message context properties will be loaded into memory at run-time. Very large pieces of data should not be written to the message context as this could potentially break the BizTalk Server large message support. Objects may be serialized into the message context providing they implement IPersistStream. Also, promoted properties are limited to 255 characters.

The message factory should always be used to create new messages. The following code snippet illustrates how to create a new BizTalk message from the data stream received by the adapter.

The IBaseMessage interface is documented in detail in the Managed Reference section of the Developers guide.

Messaging Engine Interfaces

There are three public interfaces that adapters can use to allow interaction with the Messaging Engine. These are outlined in the sections below.

IBTTransportProxy

Adapters always interact with the Messaging Engine via their own Transport Proxy. The Transport Proxy is used, amongst other things, to create batches, get the message factory, and register isolated receivers with the engine.

IBTTransportBatch

The interface exposed by the Messaging Engine's batch. Work done against the engine is performed using a batch. Batches are processed asynchronously by the Messaging Engine.

IBTDTCCommitConfirm

Adapters using explicit MSDTC transactions on batches must notify the engine of the outcome of the transaction using this interface.

Adapter Interfaces

There are three interfaces that custom adapters must implement, and two that are optional.

Mandatory interfaces

All adapters must implement these interfaces.

IBaseComponent

This interface details the **Name**, **Version** and the **Description** of the adapter.

IBTTransport

This interface details the **Transport Type** and the **ClassID** of the Adapter.

IBTBatchCallback

This interface is a call-back interface through which the adapter will be receive status and error information for a batch of messages it submits to the message engine.

Optional interfaces

Adapters may or may not implement these interfaces, depending on their needs.

IPersistPropertyBag

This is a configuration interface through which handler configuration will be delivered to the adapter. This interface is required only for adapters that have handler configuration information.

IBTTransportControl

Used to initialize and terminate an adapter. The Adapter's Transport Proxy is passed to it via this interface. This interface is not required for isolated adapters.

Transmitter Interfaces

In addition to the mandatory adapter interfaces, transmit (send) adapters, need to additionally implement either IBTTransmitter if they are non-batched or IBTBatchTransmitter if they are batched.

Receiver Interfaces

In addition to the standard adapter interfaces, receive adapters need to also implement IBTTransportConfig. This is the interface on which receive location configuration will be delivered to the adapter by the BizTalk Messaging Engine.

Request Response Adapters

Receive adapters may also be written send messages in some cases . There are usually referred to as two-way, or Request-Response adapters. To be able to send a receive adapter needs to implement IBTTransmitter.

About Adapter Variables

There are a number of variables that any custom adapter needs to handle. Values assigned to these variables influence custom logic that the adapter implements. Transport- or application-specific configuration properties also can play a part in supporting your solution. . The following table lists these common variables. For each line item, you must decide if you want to include it in your custom adapter.

Options

For each of the options, your adapter must implement specific interfaces. Following the table is a list of the possible variable options with links to more information about writing adapter code to support the design decisions you have made.

Adapter variable	Description
Communication direction	**Receive.** A receive adapter listens to a protocol-specific address for an incoming message. When a message is received, the receive adapter hands off the message to the

	Messaging Engine, which passes the message through a receive pipeline to eventually be persisted to the MessageBox database. **Send** (transmit). When the Messaging Engine needs to send a message to a specific endpoint, it passed it to the send pipeline A send adapter accepts a message from a send pipeline and sends it to the send port.
Adapter hosting	**Regular** (in-process). A regular adapter is created and hosted within the BizTalk service process, Btsntsvc.exe. This means that BizTalk creates and manages the lifetime of the adapter, initializes it with the Transport Proxy, services the adapter requests, and terminates the adapter upon service shutdown. In regular adapters, BizTalk also delivers any configuration to the adapter at run time, including handler, send port, and receive location configuration. In addition, aspects of configuration such as service windows are handled by the Messaging Engine so that the adapter does not need to ensure that a receive location or send port is outside of a service window. Note that all send adapters are regular adapters and must run in the BTSNTSVC.EXE process. **Isolated** (out-of-process). The adapter is created in an isolated host that is not part of the BizTalk runtime. An example of an isolated adapter is the HTTP receive adapter which runs in the process space of Internet Information Server (IIS). The IIS process model is such that IIS manages the lifetime of ASP.NET applications and ISAPI extensions. When it is not possible for BizTalk to manage the lifetime of the adapter, the adapter is referred to as an isolated adapter. Because isolated adapters instantiation is not managed by BizTalk Server, it is the responsibility of the adapter to create its own Transport Proxy and register itself with that Transport Proxy. Note that the BizTalk Server architecture eliminates any unnecessary inter-process communication. Because the isolated adapter and the BizTalk stack are in the same process, there is no inter-process communication when the

	adapter is calling the Messaging Engine. The only inter-process communication is between the messaging engine and the database, which is unavoidable. COM+ applications cannot host send adapters written in non-managed code.
Message exchange pattern	**One-way.** The message is either incoming or outgoing. **Request-Response** (two-way). Request-response adapters are always receive adapters. A request-response receive adapter receives a request message from the client, submits it to the server, waits for it to complete processing and for BizTalk Server to send it a response message, then sends the response message back to the client. **Solicit-Response** (two-way). Solicit-response adapters are always send adapters. A solicit-response send adapter sends a request message from BizTalk Server to a destination, waits for a response message, then submits the response message back to BizTalk Server.
Send port binding	**Dynamic Send Ports and Receive Locations.** The URI variable for the send port or receive location used in dynamic binding is determined at run time. **Static Send Ports and Receive Locations.** The URI variable for the send port or receive location is static and configured before runtime occurs.
Synchronous send versus asynchronous send adapters	**Synchronous send adapter.** While executing a send operation, the Messaging Engine blocks the transport proxy thread until it has sent the batch of messages and returned. The transport proxy takes care of the message removal after the message is transmitted, retried, suspended, or the move to the next operation is complete. Both receive adapters and send adapters can be synchronous. **Asynchronous send adapter.** The asynchronous send adapter does not block the transport proxy thread but rather uses a separate thread while performing send operations. Unlike the synchronous adapter, this adapter must implement all delete and retry logic itself; the transport proxy does not handle the logic. Adapters that send asynchronous messages allow better

	BizTalk performance than those that send synchronous messages. This is because threads spend a large amount of time waiting for inbound or outbound operations to complete. Because the Messaging Engine is highly bound to the CPU, blocking the Messaging Engine thread degrades the performance of the adapters. The use of asynchronous transmissions is strongly encouraged and improves performance.
Asynchronous receive adapters	All receive adapters are asynchronous.
Transactional support	**Transactional adapter.** Supports transactional send and receive of messages. On the send side, only asynchronous batched send adapters support transactions. **Non-transactional adapter.** An adapter that does not receive or send a message within the scope of an explicit transaction. Many adapters are non-transactional because they are sending to or from a system, such as the Windows file system, that does not support transactions.
Batch supported send adapters versus batch unsupported send adapters	**Batch supported adapter.** The send adapter can process messages in batches of operations. All adapters can collect all of the messages available for submission, and then submit them to the MessageBox database all at once, thus reducing the number of necessary database updates. In some cases, this means that the length of the batch is one. Similarly, send adapters collect all of the messages that are available for sending, extract them, and then send them to their destination all at once. In general, Microsoft® BizTalk® Server treats a batch as a unit of work for database updates. Within the same batch, adapters can submit one-way, request-response, solicit-response messages, suspend messages, delete messages, request that a message be retried for transmission, or request that a message be moved to the backup transport. Wherever possible, you should use batched transmissions. Both send and receive adapters can be batched, but only batched asynchronous send adapters support transactions. **Batch unsupported adapter.** Each message requires independent calls to the server.

Batch supported receive adapters	All receive adapters support batching.
Dynamic versus static design time	**Dynamic Adapter Design Time.** To enable an application adapter, such as SQL or SAP, to retrieve schemas by using the Add Adapter Wizard, the schema may need to change dynamically based on the destination subsystem. For dynamic adapters to work with the Add Adapter Wizard, you must develop a custom user interface that returns the service descriptions through a set of Web Services Description Language (WSDL) files. For more information, see Developing a Dynamic Design-Time Configuration for a Custom Adapter. **Static Adapter Design Time.** The schema does not change with the back-end subsystem or send port. For information about modifying the sample file adapter, see Developing a Static Design - Time Configuration for a Custom Adapter .
Transport adapters versus application adapters	**Transport adapters.** Transport adapters support a specific protocol and don't use a schema. Of the twelve native adapters, nine are transport adapters: MSMQT, BizTalk Message Queuing, MQ Series, File, FTP, HTTP, SMTP, POP3, and SOAP. **Application adapters.** Application adapters use data schemas to send data into the specified application. Of the twelve native adapters, three are application adapters: Base EDI, Windows Sharepoint Services, and SQL.

For information about which interfaces to implement in your custom adapter code, use the link in the following table for the type of adapter you plan to create. In some cases, you will have to combine the interfaces listed to meet one requirement with the interfaces listed on a separate page to meet another requirement.

Variable	**For more information**
Send adapters	Send Adapter Initialization
Synchronous	Synchronous Send Adapter Creation
Asynchronous	Asynchronous Send Adapter Creation
Synchronous batch supported	Synchronous Batch-Supported Send Adapter Creation
Asynchronous batch supported	Asynchronous Batch-Supported Send Adapter Creation

Transactional asynchronous batch supported	Transactional Asynchronous Batch-Supported Send Adapter Creation
Solicit-response	Solicit-Response Send Adapter Creation
Receive adapters	Receive Adapter Initialization
Regular	Regular Receive Adapter Creation
Isolated	Isolated Receive Adapter Creation
Batch supported	Batch-Supported Receive Adapter Creation
Transactional batch supported	Transactional Batch-Supported Receive Adapter Creation
Synchronous request-response	Synchronous Request-Response Receive Adapter Creation

Adapter Development with the Adapter Framework

This section contains information about the object interactions that take place during the adapter run time, as well as information about what interfaces are required to support specific adapter variables.

This section contains:

- Initializing the Adapter
- Custom Adapter Configuration
- Receiving messages
- Sending Messages
- Batching Messages
- Handling Large Messages
- Handling Transactions
- Handling failure
- Terminating the Adapter
- Security and SSO

- Performance tuning
- Testing and Debugging
- Localization Issues
- Design Issues

Initializing the Adapter

The following section describes how BizTalk Server instantiates and initializes an adapter.

Adapter Creation

Upon startup of the BizTalk Server service, all receive adapters that have one or more active receive locations configured are instantiated. By default send adapters are not instantiated until the Messaging Engine de-queues the first message to be sent via that send adapter. Should requirements dictate instantiation of a send adapter on service startup the **InitTransmitterOnServiceStart** property may be used. This directs the Messaging Engine to create the send adapter on service startup rather than using the default lazy creation. The default lazy creation approach helps to reduce the amount of systems resources used when adapters are not configured on endpoints.

It is recommended that adapters are developed using managed code, although they may be developed as native COM components. For COM components the adapter would be instantiated in the normal manner using **CoCreateInstance**().

For managed adapters the .NET **type** needs to be specified in the configuration file, and the assembly path is optional.

The following deployment options are possible:

.Net Type	Assembly Path	Assembly Deployment Method
Specified	Not specified	XCopy assembly to product directory or sub directory in the product directory with the same name as the assembly
Specified	Not specified	GAC assembly
Specified	Specified	XCopy assembly to directory specified

Trouble Shooting Tip: When developing managed code adapters, if creation fails, fuslogvw.exe is a useful tool to determine if there are references to assemblies that cannot be resolved. This is a common mistake.

The following diagram illustrates the logic for creating adapters depending on the configuration specified:

The following table gives an example of the configuration for a receive adapter and the three ways the run-time assembly maybe configured for creation.

Deployment Option	**InboundTypeName**	**InboundAssemblyPath**
Specify assembly location	Microsoft.Samples.MyReceiveAdapter	C:\MyAdapter\MyAdapter.dll
Specify .Net type	Microsoft.Samples. MyReceiveAdapter,MyAdapter	N/A
GAC assembly	Microsoft.Samples. MyReceiveAdapter,MyAdapter	N/A

This section contains

- Receive Adapter Creation
- Send Adapter Creation
- Creation of Isolated Adapters

Receive Adapter Creation

This section describes the object interactions that occur within receive adapters. You can use this information to guide custom adapter development when creating receive adapters, or to learn about how the native adapters work.

This section contains:

- Receive Adapter Initialization
- Regular Receive Adapter Creation
- Isolated Receive Adapter Creation
- Batch-Supported Receive Adapter Creation
- Transactional Batch-Supported Receive Adapter Creation

- Synchronous Request-Response Receive Adapter Creation

Receive Adapter Initialization

Immediately after a receive adapter is instantiated it is initialized by the Messaging Engine. First the Messaging Engine QueryInterface for the mandatory interface IBTTransportControl, The engine will then call Initialize() passing in the Adapters' Transport Proxy. Next it will call QueryInterface for IPersistPropertyBag. This is an optional interface which if the adapter implements it the handler configuration will be passed to the adapter in the Load() method call. The final stage of initializing a receive adapter involves passing the endpoint configuration to the adapter, during this phase the engine will call IBTTransportConfig.AddReceiveEndpoint() once for each active endpoint passing in the URI for the endpoint, the adapter specific configuration for the endpoint, and the BizTalk configuration for that endpoint.

The following diagram illustrates this sequence of API calls, the interfaces in blue are implemented by the adapter.

Implementation Tip: In general adapters should not block the engine in calls such as IBTTransportControl.Initialize(),IPersistPropertyBag.Load() and IBTTransportConfig.AddReceiveEndpoint(). Performing excessive processing in these calls can have a negative impact on service start up time.

All receive adapters that have one or more associated receive locations are created at service startup. All receive adapters are asynchronous and support batching. They can be regular (in-process) or isolated (non-creatable). For additional information about receive adapter variables, see About Adapter Variables.

Regular Receive Adapter Creation

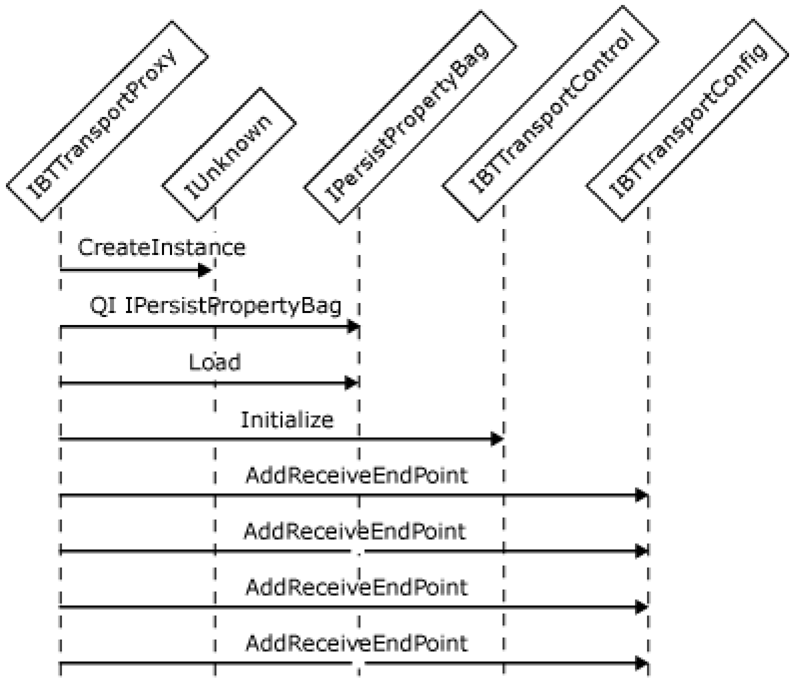
The Messaging Engine instantiates and configures in-process adapters, passing in the transport proxy to allowing access to its functionality the adapter. To enable configuration and binding to the transport proxy, adapters must implement the following configuration interfaces:

- IBTTransport
- IBTTransportControl
- IBTTransportConfig
- IBaseComponent
- IPersistPropertyBag

The Messaging Engine creates an instance of an adapter, initializes it, and sets the configuration of receive locations. The Messaging Engine passes a property bag to an adapter on the **AddReceiveEndpoint** method call. The property bag contains the configuration for the receive location and receive handler. The configuration is stored in the database in the form of an XML-styled property bag. The Messaging Engine reads the XML and rehydrates a property bag from the XML. After at least one endpoint (receive location) is added, the adapter can start submitting messages.

The following illustration shows the object interactions involved in creating a regular receive adapter.

Depicts the workflow for a creatable receive adapter



Isolated Receive Adapter Creation

Isolated receive adapters are hosted in a process space other than the BizTalk Server process.To interact with the Messaging Engine, an isolated receive adapter register itself on startup so that the engine can configure and control them.The adapter creates the transport proxy, querries for the interface **IBTTransportProxy**, and calls **IBTTransportProxy.RegisterIsolatedReceiver** to register its IBTTransportConfig callback interface with the Messaging Engine. This synchronous call occurs before the adapter submits its first message to BizTalk Server. This allows the Messaging Engine to call back into the adapter and tell it which of it's endpoints are active and should be listened on for incoming messages. Isolated adapters must implement the following interfaces:

- IBTTransport
- IBTTransportConfig

- IBaseComponent
- IPersistPropertyBag

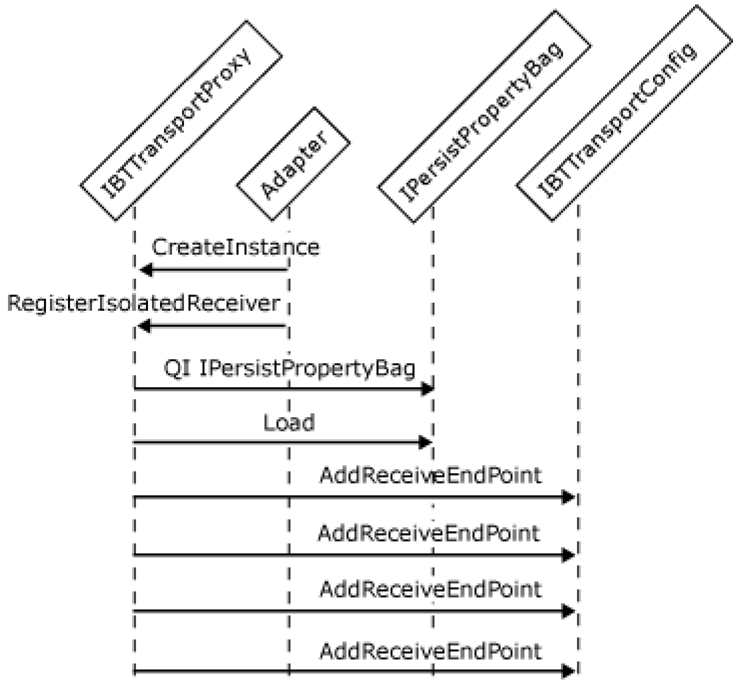
Registering the adapter requires that the adapter passes a configured and enabled receive location. The credentials of the adapter's host process must be a member of the BizTalk Isolated Host Users group. In addition, the adapter will be queried to ensure that it has the correct class ID and is running on the computer that was configured for that host instance.

After the adapter has successfully registered with the transport proxy, the Messaging Engine passes the configuration information and the other receive locations back to the adapter by calling the **Load** method of the **IPersistPropertyBag** interface and the **AddReceiveEndpoint** method of the **IBTTransportConfig** interface respectively.

When an isolated receive adapter ends the processing of messages and is going to be terminated, it must call the **TerminateIsolatedReceiver** method of the **IBTTransportProxy** interface.

The following illustration shows the object interactions involved in creating an isolated receive adapter.

Depicts the workflow for initializing a non-creatable receive adapter.



Batch-Supported Receive Adapter Creation

A receive adapter always submits messages in a batch. A batch is a unit of database operations that you can use for actions other than submission. For example, a receive adapter can submit one set of messages, suspend a different set of messages, and delete another set of messages in the same batch. Grouping these separate operations in the same batch optimizes performance by minimizing the number of database round trips required and is strongly encouraged.

Regular (in-process) and isolated receive adapters need to implement the following interfaces to submit batches of messages into the server:

- IBTTransport
- IBTTransportControl (in-process adapters only)
- IBTTransportConfig
- IBaseComponent
- IPersistPropertyBag
- IBTBatchCallBack

The following steps describe the sequence of actions that a receive adapter performs to submit messages into the server.

1. A receive adapter obtains the batch from the transport proxy by calling the **GetBatch** method of the **IBTTransportProxy** interface. In its call to GetBatch the adapter passes in a pointer to its IBTBatchCallback interface implementation.

2. An adapter adds the messages one at a time into the batch by calling the **SubmitMessage** method of the **IBTTransportBatch** interface.

3. When all the messages have been added to the batch, the adapter calls the **Done** method of the **IBTTransportBatch** interface to submit the batch to the transport proxy.

4. After the batch has been processed, the Messaging Engine invokes the adapter's **BatchComplete** callback method using the transport proxy to make the actual call. An array of **BTBatchOperationStatus** objects containing the status of the submission is passed to the adapter. Each object corresponds to an operation type and contains the overall status of the operation as well as the status for each message for which the operation was performed. The following sequence describes the actions the adapter needs to perform to analyze the status of batch processing.

a. Check the overall batch status hResult value passed as a parameter to the **BatchComplete** method. If it is a failure hResult none of the operations in the batch succeeded.

If the overall batch status succeeded, it means that all the messages that were given to the transport proxy were persisted to disk. However, it does not mean that the pipeline successfully processed all the messages. It is possible messages that failed in the pipeline were suspended. For these messages that fail in the pipeline, the overall batch status returned is successful because the data was written to disk.

b. Check the status for each operation type in the operationStatus parameter. If the status is **S_OK,** the submission for this operation succeeded and you do not check the status any further. . If set to **BTS_S_EPM_MESSAGE_SUSPENDED** some of the messages were suspended. **S_EPM_SECURITY_CHECK_FAILED** signifies some messages failed authentication in an authentication-required receive port. If **E_FAIL** or any HRESULT that is less than zero the message submission failed.

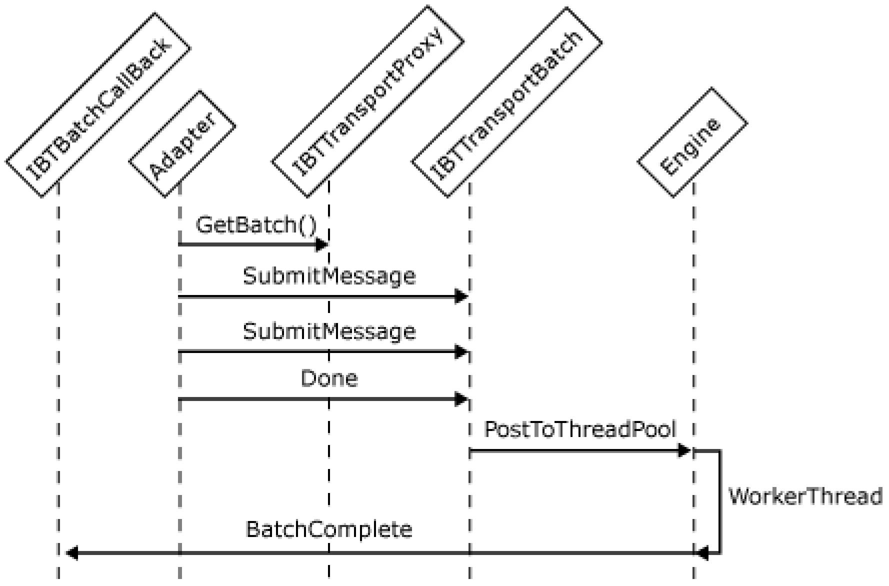
c. Check the status of individual messages for the operation type. For the submit operation type, the status of each message is set to **S_OK** if the submission succeeded **BTS_S_EPM_MESSAGE_SUSPENDED** is returned if the message was suspended. **S_EPM_SECURITY_CHECK_FAILED** is returned if the message failed authentication in an authentication-required receive port. **E_BTS_NO_SUBSCRIPTION** comes back if there were no subscribers for the published message, or **E_FAIL** or any HRESULT that is less than zero if submission failed.

d. Depending on your adapter, you may want to suspend messages that return **E_FAIL** or any failing HRESULT.

5. The **BatchComplete** method needs to return either **S_OK** or **E_FAIL** to indicate the result of execution. If the **BatchComplete** method returns **E_FAIL** or any negative HRESULT, the transport proxy logs an error.

The following illustration shows the object interactions involved in creating a batch-supported receive adapter.

Depicts the workflow for a receive adapter submitting a batch of messages



Note that this process is very similar to the workflow of an asynchronous batch-supported send adapter. The process differences can be seen when the following lists of methods are compared:

The receive adapter calls the following methods:

- SubmitMessage
- SubmitRequestMessage
- MoveToSuspendQ
- CancelResponseMessage

The asynchronous send adapter calls the following methods:

- SubmitResponseMessage
- Resubmit
- DeleteMessage
- MoveToNextTransport
- MoveToSuspendQ

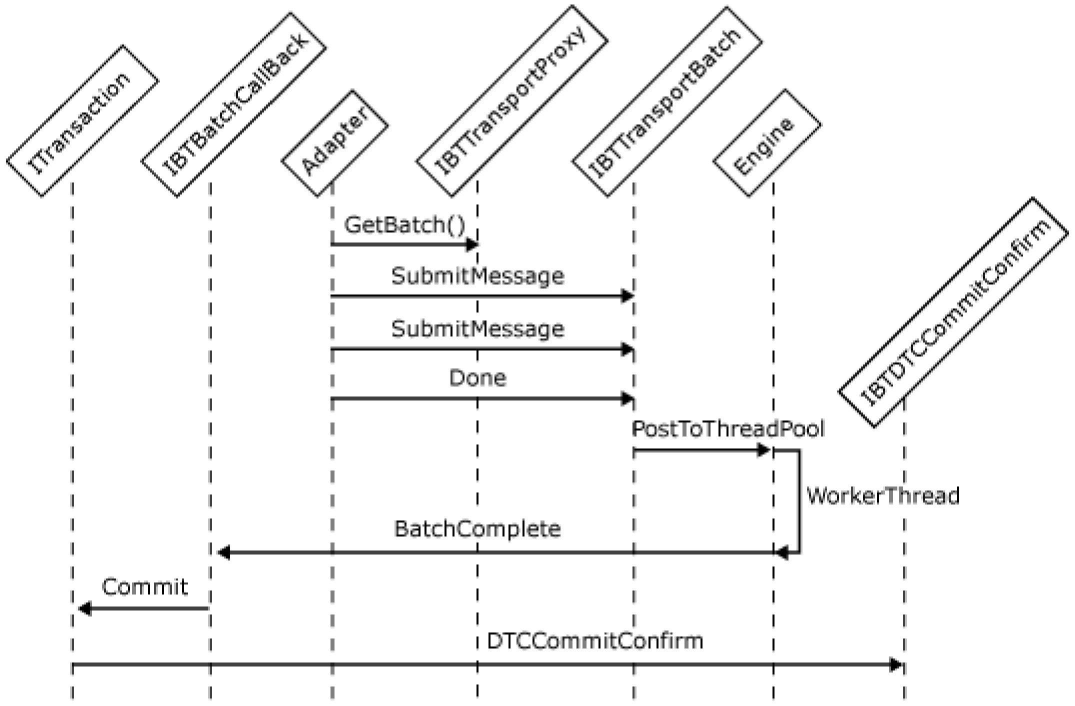
Transactional Batch-Supported Receive Adapter Creation

A receive adapter creates and controls transactions when the transactional submission of messages is required.

A transactional receive adapter creates and passes a pointer to a Microsoft Distributed Transaction Coordinator (MSDTC) transaction on the **Done** method of the **IBTTransportBatch** interface. This ensures that all batch operations are performed in the scope of that specific transaction object. When the batch submission completes, the adapter callback method commits or rolls back the transaction. Which action it takes depends upon the status returned from the transport proxy and possibly any other transaction related work done by the adapter not visible to the transport proxy. It is the adapter which has the final say as to whether or not the transaction failed or succeeded. The adapter reports the result of the transaction (commit or rollback) back to the transport proxy by using the DTCCommitConfirm method of the **IBTDTCCommitConfirm** interface. It passes in true for a successful transaction or false for a failure.

The following illustration shows the object interactions involved in creating a transactional batch-supported receive adapter.

Depicts the workflow for a receive adapter submitting a batch of messages using DTC transactions



Synchronous Request-Response Receive Adapter Creation

All receive adapters need to implement the following interfaces to work in request-response mode:

- IBTTransport
- IBTTransportControl (regular adapters only)
- IBTTransportConfig
- IBaseComponent
- IPersistPropertyBag
- IBTBatchCallBack
- IBTTransmitter

Receive adapters that support **request-response** protocols (for example, the HTTP receive adapter) perform the following actions when submitting request messages.

1. The receive adapter receives incoming request messages. It obtains a batch from the transport proxy by calling the **GetBatch** method of the **IBTTransportProxy** interface. In this call the adapter passes in a callback pointer to its implementation of the IBTBatchCallBack.BatchComplete method.

2. The adapter adds request messages into the batch by calling the **SubmitRequestMessage** method of the **IBTTransportBatch** interface, once for each request message.

3. When the all the messages have been added, the adapter calls the **Done** method of the **IBTTransportBatch** interface, which submits the batch to the Messaging Engine via the transport proxy.

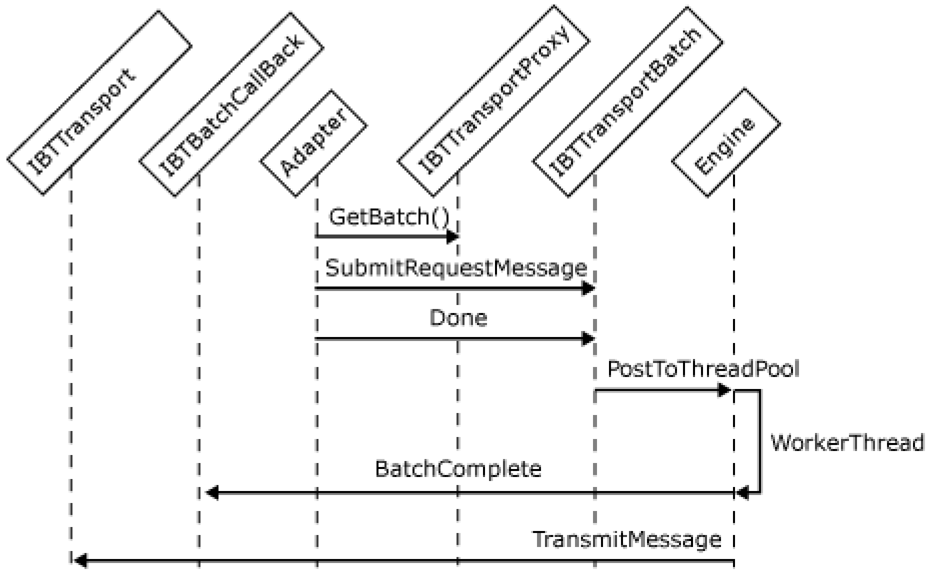
4. After the batch has been processed, the Messaging Enginer invokes the adapter's **IBTBatchCallBack.BatchComplete** callback method via the transport proxy.. The status of the submission is passed to the adapter as an array of HRESULTS corresponding to each message in the batch. If the batch fails, either in the pipeline or in the orchestration, the SOAP fault message is returned to the adapter as a response.

5. There incoming request messages may have orchestration subscribers. After the orchestration completes and the request message has been processed the Messaging Engine sends the response message via the transport proxy passes to the adapter by calling the adapter's **TransmitMessage** method from the **IBTTransmitter** interface.

6. The adapter sends a response message and deletes the original message from the MessageBox database.

   The following illustration shows the object interactions involved in creating a synchronous request-response receive adapter.

   Depicts the workflow for a receive adapter submitting a synchronous message



Send Adapter Creation

This section describes the object interactions that occur within send adapters. You can use this information to guide custom adapter development when creating send adapters, or to learn about how the native adapters work.

This section contains:

- Send Adapter Initialization
- Synchronous Send Adapter Creation
- Asynchronous Send Adapter Creation
- Synchronous Batch-Supported Send Adapter Creation
- Asynchronous Batch-Supported Send Adapter Creation
- Transactional Asynchronous Batch-Supported Send Adapter Creation
- Solicit-Response Send Adapter Creation

Send Adapter Initialization

By default, send adapters are not instantiated until the first message is delivered to them, known as lazy creation. The default lazy creation approach helps to conserve system resources. After they are created, they are cached and live until the BizTalk Server service is stopped.

The **InitTransmitterOnServiceStart** member of the **Capabilities** enumeration directs the Messaging Engine to create the send adapter on service startup, rather than using the default lazy creation if this is desired.

Sending a message is a different model than receiving a message with respect to batching of messages. In the receive case the adapter has incoming messages to insert into a batch obtained from the Messaging Engine. In the send case the Messaging Engine obtains a batch from the adapter and send message to the adapter to be transmitted. Both use the transport proxy to obtain the message batch objects.

Transmit Adapter Initialization

To enable configuration and binding to the transport proxy, adapters must implement the following configuration interfaces:

- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**
- **IPersistPropertyBag**

The following steps describe the sequence of events involved in initializing a send adapter.

1. When the Messaging Engine initializes a send adapter, it first performs a **QueryInterface** for **IPersistPropertyBag**, which is an optional interface. If the adapter implements the interface, the handler configuration is passed to the adapter in the **Load** method call. The adapter uses this information to ensure it is configured correctly.

2. The Messaging Engine performs a **QueryInterface** for **IBTTransportControl**, which is a mandatory interface.

3. The engine calls **IBTTransportControl.Initialize**, passing in the transport proxy for the adapter.

4. The Messaging Engine performs a **QueryInterface** for **IBTTransmitter**.

If the Messaging Engine discovers this interface, the adapter is treated as a batch-unaware transmitter.

If the Messaging Engine does not discover this interface, the Messaging Engine performs a **QueryInterface** for **IBTBatchTransmitter**, discovery of which indicates that the adapter is a batch-aware transmitter.

If the Messaging Engine discovers neither of these interfaces, an error condition results, causing the initialization to fail. The initialization fails if any mandatory interfaces are not discovered.

The following diagram illustrates this sequence of API calls; the interfaces in blue are implemented by the adapter.

The adapter can send messages as soon as it is initialized and configured.

The following figure shows the object interactions involved in initializing a send adapter.

Depicts the workflow for initializing a transmit adapter

ebiz_sdk_devadapter10#0828d9ff-e2cc-4ab0-9aed-65fa9ffcf86a

Note Adapters should not block the Messaging Engine in calls such as **IBTTransportControl.Initialize** and **IPersistPropertyBag.Load**. Performing excessive processing in these calls will have an impact on service startup time.

Synchronous Send Adapter Creation

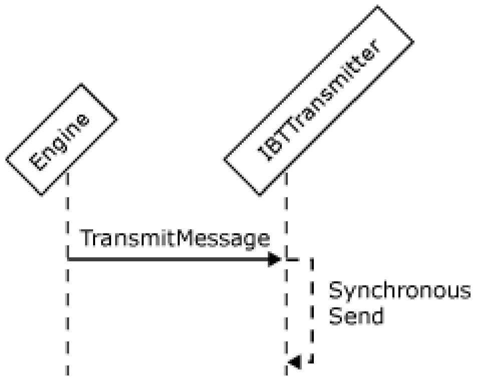
An adapter sends messages synchronously when it blocks the incoming Messaging Engine calling thread while performing its send operation. To be able to send messages synchronously, an adapter needs to implement the following interfaces:

- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**
- **IPersistPropertyBag**
- **IBTTransmitter**

In a synchronous send, the adapter sends the message while blocking **TransmitMessage**, and after successful transmission returns **True** for **bDeleteMessage**.

The following illustration shows the object interactions involved in creating a synchronous send adapter.

Depicts the workflow for sending a message synchronously



Asynchronous Send Adapter Creation

Adapters sending messages one at a time may send messages either synchronously or asynchronously. An adapter sends messages asynchronously when it does not block the transport proxy but rather uses its own separate thread while performing send operations. To be able to send messages asynchronously, an adapter needs to implement the following interfaces:

- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**
- **IPersistPropertyBag**
- **IBTTransmitter**

The following steps describe the sequence of actions that the send adapter performs to transmit messages out of the server at the request of the Messaging Engine.

1. The Messaging Engine uses the transport proxy to pass an outgoing message to a send adapter by calling the **TransmitMessage** method of the **IBTTransmitter** interface.

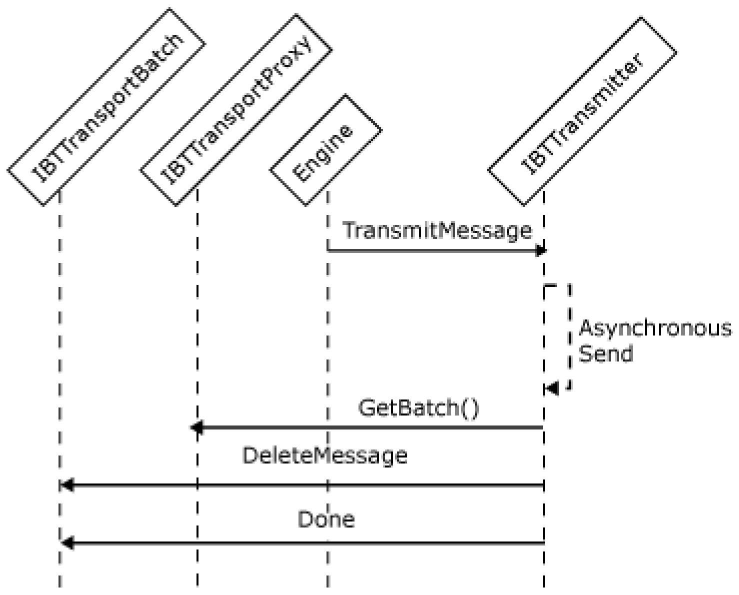
2. The adapter returns immediately from **TransmitMessage** after storing the message to be sent to some internal queue, and returns **False** for **bDeleteMessage**. This tells the Messaging Engine the message will be transmitted in an asynchronous manner.

3. The adapter sends the message using its own thread pool.

4. After the send operation completes, the adapter deletes the original message from the MessageBox database. It obtains a batch from the Messaging Engine using **IBTTransportBatch.GetBatch** method of the transport proxy, then calls **DeleteMessage**.

   The following illustration shows the object interactions involved in creating an asynchronous send adapter.

   Depicts the workflow for sending a message asynchronously



Synchronous Batch-Supported Send Adapter Creation

Batch-aware adapters may send messages synchronously or asynchronously, and may perform transacted send operations.

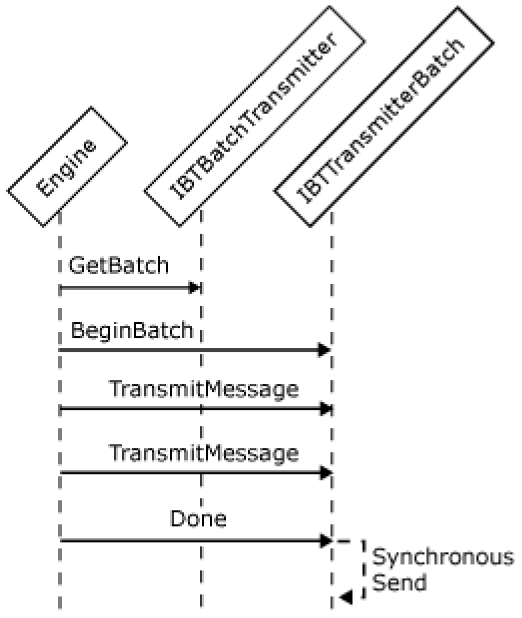
To send batches of messages, a send adapter must implement the following interfaces:

- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**
- **IPersistPropertyBag**
- **IBTBatchTransmitter**
- **IBTTransmitterBatch**

For the synchronous batch send, the Messaging Engine gets a batch from the adapter and adds messages to be transmitted to that batch. The Messaging Engine adds each message to the batch and sends the messages only when it calls the **Done** method on the batch. The adapter returns **True** for **bDeleteMessage** for each message that it intends to transmit synchronously. The adapter should save message data, as opposed to a message pointer, in its **TransmitMessage** implementation. This is because the message pointer is no longer valid after **True** is returned, and should not be used or cached for later use.

The following illustration shows the object interactions involved in creating a synchronous batch-supported send adapter.

Depicts the workflow for submitting a message synchronously



Asynchronous Batch-Supported Send Adapter Creation

Batch-aware adapters may send messages synchronously or asynchronously, and may perform transacted sends.

To send batches of messages, a send adapter must implement the following interfaces:

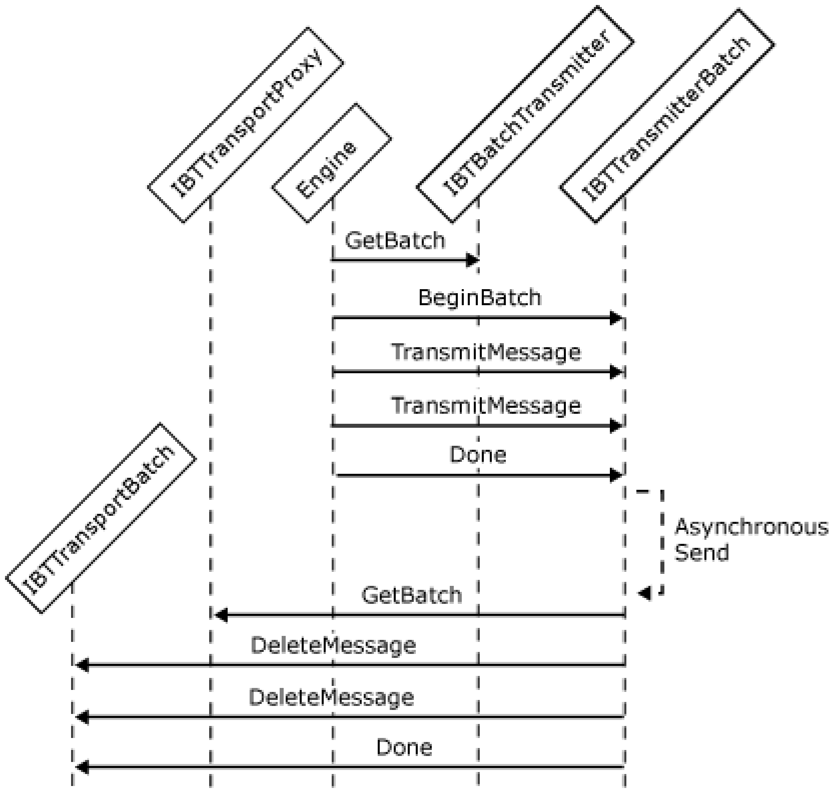
- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**
- **IPersistPropertyBag**

- **IBTBatchTransmitter**
- **IBTTransmitterBatch**

For the asynchronous batch send, the Messaging Engine gets a batch from the adapter and adds messages to be transmitted to that batch. The messages are only sent when the Messaging Engine calls the **Done** method on the batch. The adapter returns **False** for **bDeleteMessage** for each message that it intends to transmit asynchronously. The adapter then gets a batch from the adapter proxy and deletes those messages that it successfully transmitted.

The following illustration shows the object interactions involved in creating an asynchronous batch-supported send adapter.

Depicts the workflow for sending a message synchronously



Transactional Asynchronous Batch-Supported Send Adapter Creation

A send adapter can create and control transactions when transactional transmission of messages is required.

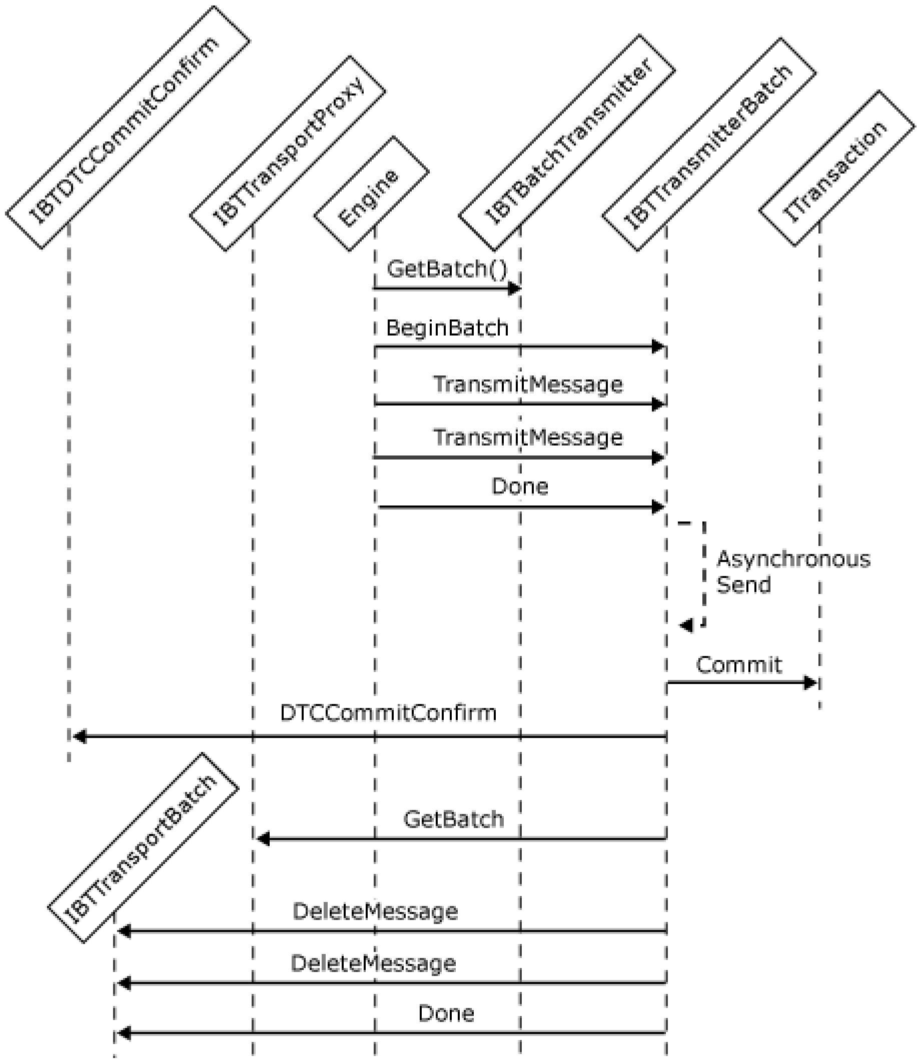
To support transactional send, an adapter needs to implement the following interfaces:

- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**
- **IPersistPropertyBag**
- **IBTBatchTransmitter**
- **IBTTransmitterBatch**
- **IBTBatchCallBack**

An adapter creates an MSDTC transaction and returns a pointer to that object in the call to the **BeginBatch** method of the **IBTTransmitterBatch** interface. This method is called by the Messaging Engine to obtain a batch with which it posts outgoing messages to the send adapter. When the adapter finishes the send operation and commits or rolls back a transaction, it notifies the Messaging Engine of the result of the transaction by using the **DTCCommitConfirm** method of the **IBTDTCCommitConfirm** interface.

The following figure shows the interaction between the transport proxy and the send adapter when performing a transactional send operation.

Depicts the workflow for sending a message asynchronously



Solicit-Response Send Adapter Creation

Send adapters use the same batch mechanism as receive adapters to submit response messages back into the server.

Send adapters need to implement the following interfaces to work in solicit-response mode:

- **IBTTransport**
- **IBaseComponent**
- **IBTTransportControl**

- **IPersistPropertyBag**
- **IBTTransmitter** (or **IBTTransmitterBatch** and **IBTBatchTransmitter** if send batching is required)
- **IBTBatchCallBack**

The steps involved in the object interaction are as follows.

1. After the adapters sends a solicit message, it eventually receives back a response message from that destination server. It then obtains a batch from the transport proxy.

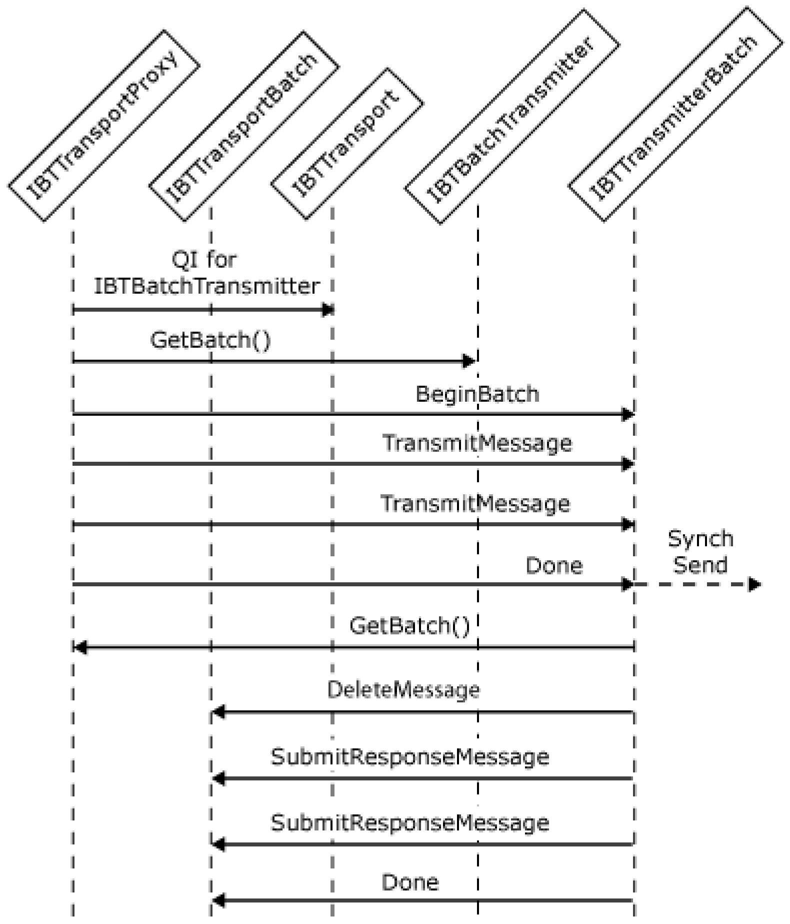
2. The adapter adds the response message to the batch by calling **IBTTransportProxy::SubmitResponseMessage**.

3. The adapter submits the batch by calling **IBTTransportProxy::Done** passing in a pointer to its IBTBatchComplete interface for the callback from the Messaging Engine.

4. The Messaging Enginer calls the adapter's **IBTBatchCallBack::BatchComplete** callback method using the transport proxy notifying it of the result of submission operation.

   The following illustration shows the object interactions involved in creating a solicit-response send adapter.

Interaction diagram for a solicit-response send adapter



Creation of Isolated Adapters

As discussed previously isolated adapters are not instantiated by BizTalk. Rather, they are instantiated and hosted in another process. It is thus the responsibility of the adapter to create its Transport Proxy, **QueryInterface** for **IBTTransportProxy**, and then call **IBTTransportProxy**.**RegisterIsolatedReceiver**() to register with the Messaging Engine.

Registration requires that the adapter pass one of it's configured and enabled receive locations to the Messaging Engine. The adapters host process credentials must be a member of the "BizTalk Isolated Host Users" group. it should be noted that simply using impersonation here is insufficient unless of course the user is a member of that group. In addition the adapter will be queried to ensure it has the correct ClassID and is running on the machine that was configured for that host instance. Once the adapter has successfully registered with its transport proxy, its configuration will be send to using the IPersistProperty bag interface.

The following diagram illustrates this sequence of API calls; the interfaces in blue are implemented by the adapter.

The following code snippet illustrates the registration API calls.

Implementation Tip: It is recommended that the adapter keeps a count of the work in progress. The adapter should block in **Terminate**() until the message count has reached zero. On the receive side this work will include any outstanding requests that have not been published to BizTalk. It should be noted that response messages will not be delivered to a receive adapter once **Terminate**() has been called.

For send adapters, messages that are in progress should be handled appropriately. This means any message that was successfully delivered should be deleted from the adapter's private application message queue to prevent messages from being double sent.

In general, once **Terminate**() is called the Messaging Engine will not accept requests to publish new messages with the exception of response messages for solicit-response pairs.

Custom Adapter Configuration

Adapter configuration is stored in the Single Sign-On (SSO) database when the user makes configuration changes during design-time. At run-time the Messaging Engine retrieves the adapter's configuration and delivers it to the adapter. There are four types of configuration information delivered to adapters:

- Receive handler configuration
- Receive location (endpoint) configuration
- Send handler configuration
- Send location (endpoint) configuration

Receive and Send Handler Configuration

An adapter's handler configuration is delivered to the adapter on it’s implementation of optional **IPersistPropertyBag**.**Load**()interface. The handler configuration is delivered once only so if adapter the handler configuration is changed after the BizTalk Service has started the adapter is not updated. In general, the common model is for the adapter to treat the handler configuration as the default configuration; endpoint configuration overrides the handler configuration on a per endpoint basis.

The following code snippet demonstrates parsing the configuration. The adapters’ configuration will be in the property passed in to the Load call in the string property **AdapterConfig.** This property value will contain an XML document representing the adapter's configuration. The adapter needs to load this configuration into a DOM or an XML Reader and use XPath to retrieve the individual properties.

Receive Location Configuration

Receive location configuration information is delivered to an adapter on it's implementation of **IBTTransportConfig**.This interface contains the three methods **AddReceiveEndpoint**, **UpdateEndpointConfig** and **RemoveReceiveEndpoint**. The Messaging Engine will notify the adapter of which endpoints it needs to listen on to receive messages. When the configuration for an individual endpoint is changed the adapter will be notified of the change for that endpoint. This is contrary to when handler configuration changes the adapter is not notified. Similarly, the adapter does not need to worry about service windows as BizTalk Server will will add or remove endpoints as service windows become active or inactive.

Trouble Shooting Tip: Adapter configuration is cached by the Messaging Engine for performance reasons. The engine will only look for changes in configuration every "cache refresh interval". For this reason, there is a delta between changing the endpoints configuration and the adapter being notified of this change, in general this delta is less than the cache refresh interval, but when the server is under load this delta can be slightly longer. The cache refresh interval is configured in the Admin MMC at the group level.

AddReceiveEndpoint

When an adapter needs to begin listening on an endpoint the engine will call **IBTTransportConfig.AddReceiveEndpoint()** passing in the receive locations URI, a property bag containing the adapters configuration for that endpoint, and a second property bag containing BizTalk Server-specific configuration for that endpoint. The URI passed needs to be written to the message context as the BizTalk Server system property **InboundTransportLocation** by the adapter.

Reading the receive location properties from the adapters property bag is no different from reading the handler configuration as detailed above. The BizTalk Server configuration passed into the adapter contains a single property, **TwoWayReceivePort**, this indicates whether the port is one way or two way. The following code snippet illustrates how to evaluate if the receive port is one-way or two-way from the BizTalk property bag:

UpdateEndpointConfig

When the configuration of a receive location that is already active is changed, the engine will notify the adapter that it needs to use different configuration via the API **UpdateEndpointConfig**(). All of the configuration will be delivered to the adapter, including the BizTalk specific configuration.

RemoveReceiveEndpoint

When a receive location is no longer active, the adapter will be notified via **RemoveReceiveEndpoint**(), once the adapter returns from **RemoveReceiveEndpoint**() it will no longer be allowed to submit messages into the engine using that uri.

Send Port Configuration

The Messaging Engine writes the configuration for the send port on to the message context in the adapter's namespace before delivering the message to the adapter. It is the adapters' responsibility to read and validate that configuration which it will subsequently use to control the transmission of the message. For transmit adapters that support batched sends, messages destined for different send ports may be in the same batch, so the adapter will need to handle these 'mixed' batches.

The following code snippet illustrates firstly how to read the **OutboundTransportLocation** which is the URI for the send port. It shows how to read the Xml blob containing the adapter's configuration and subsequently read the individual properties.

Implementation Tip: Adapters should in general use the **OutboundTransportLocation** message context property to determine the address which to send the message to. By doing this the adapter will be able to handle transmissions to both static and dynamic sends consistently and also, the modification of address in production binding files will be simplified.

XSD

Four XSD files included in the SDK File Adapter sample primarily handle adapter configuration: ReceiveHandler.xsd, ReceiveLocation.xsd, TransmitLocation.xsd, and TransmitHandler.xsd.

This section discusses each of these files and describes how you can modify them.

This section contains:

- Adapter Configuration Validation
- Adapter Registration
- Adapter Framework Configuration Schema Extensions
- Supported Adapter XSD Element Types
- Adapter XSD Element-Attribute Constructs
- Adapter XSD Data Type-Facet Constructs

- Advanced Configuration Components for Adapters

Adapter Configuration Validation

While adding the receive location and send port, you will be asked to configure your custom properties in the **<Adapter Name> Transport Properties** dialog box. The XSD schema files in the AdapterHarness project define these properties.

Validation of the configuration schema occurs in three parts:

1. When displaying a saved configuration, the framework validates the saved XML document against the schema before loading the document into the property page. The framework assumes that a document that is not valid indicates a change in the configuration schema definition. Only valid documents get loaded into the property page.

2. When saving a configuration if the adapter implements the **IAdapterConfigValidation** interface, the framework passes to the adapter the XML document constructed from serializing the property page data. The adapter then processes the document. Any errors should produce exceptions that are caught by the framework and displayed to the user. Any missing or generated values should be generated during validation. Using the <browsable show="false"> decoration suppresses showing an entry in the property grid, even though the value appears in the XML instance.

3. When saving a configuration before placing the value into the database, the framework again validates the XML document against the schema. This ensures that only valid data is persisted.

Adapter Registration

If you are developing a custom adapter, you can register it with BizTalk Server by modifying and running one of the registry files included with the Sample File Adapter in the Software Development Kit (SDK). Or you can use the Adapter Registration Wizard to create a registry file from scratch. This is located in the <BizTalk Server 2006 Installation Path\Utilities\AdapterRegistryWizard folder.

After you create registry entries, you can add the adapter in the BizTalk Administration console or programmatically by using BizTalk Windows® Management Instrumentation (WMI) methods. This topic discusses each of the registry entries and then shows you where and how to modify the existing registry files for your custom adapter.

For instructions on using the Adapter Registration Wizard, see Adapter Registration Wizard. For instructions on modifying the sample registry files included in the SDK, see Modifying the Adapter Registration File.

Registry keys

You need to create the following registry entries to deploy an adapter:

Registry key location

The location to write to in the registry.

Type name

Adapter type name identifies the type of adapter in the BizTalk Server computer.

This is a required key for any adapter.

Constraints

Adapter constraints define the capabilities of the adapter's functionality.

This is a required key for every adapter. Depending on the type of adapter you are creating, you may want to modify the bitmask value of the constraints.

The value that describes the capabilities of the adapter can be a combination of values shown in the following table.

Value	Hex value	Flag	Description
1	0x0001	eProtocolSupportsReceive	Adapter supports receive operations.
2	0x0002	eProtocolSupportsTransmit	Adapter supports send operations.
8	0x0008	eProtocolReceiveIsCreatable	Receive handler of adapter is hosted in-process.
128	0x0080	eProtocolSupportsRequestResponse	Adapter supports request-response operations.
256	0x0100	eProtocolSupportsSolicitResponse	Adapter supports solicit-response operations.
1024	0x4000	eOutboundProtocolRequiresContextInitialization	Indicates that the adapter uses the adapter framework user interface for send

			handler configuration.
2048	0x0800	eInboundProtocolRequiresContextInitialization	Indicates that the adapter uses adapter framework user interface for receive handler configuration.
4096	0x1000	eReceiveLocationRequiresContextInitialization	Indicates that the adapter uses adapter framework user interface for receive location configuration.
8192	0x2000	eTransmitLocationRequiresContextInitialization	Indicates that the adapter uses adapter framework user interface for send port configuration.
16384	0x4000	eSupportsOrderedDelivery	Indicates that the adapter supports ordered delivery of messages.
32768	0x8000	eInitTransmitterOnServiceStart	Send adapter starts when the service starts instead of when it sends the first message.

Namespace

Each adapter must define a namespace for its properties. BizTalk Server stores adapter-specific properties on the message context under this namespace. This is a required property for all adapters.

Aliases

Each adapter may have a set of prefixes that uniquely identify the adapter type within BizTalk Server. The adapter needs to specify the list of its prefixes at registration time. Prefixes must be unique within BizTalk Server.

Property pages for receive handlers, send handlers, receive locations and send ports

The adapter must have configuration property pages to configure its' receive locations and send ports. Each adapter registers its property pages by specifying their respective class IDs.

If the adapter uses the Adapter Framework's user interface for property page generation, it must specify the following values for the registry keys:

Note that if one of the endpoints is not required (the adapter is send or receive only), the unused registry keys can be deleted from the registry.

Runtime components registration

The adapter registers its runtime components by specifying their class IDs (for COM and .NET), type names, and assembly paths (for .NET) for receive and transmit runtime components.

Registration of adapter properties for SSO configuration store

The adapter needs to register its properties with the BizTalk Server Credential database to be able to store and retrieve the properties at design time and run time.

These values contain the definitions (schema) for the allowed properties of the corresponding entities related to the adapter, which can be stored in the configuration store. These definitions are kept as an XML string being which is de-serialized by the property bag containing property types but without values. A nonempty value of the property element means that the property is masked. (Masked means that it is write-only, and is not returned by the Secure Store API when called in administrative mode; the Secure Store API returns VT_NULL for such properties.)

Example

The HTTP adapter registers its properties for the HTTP send port by defining the **SendLocationPropertiesXML** registry key with the following value:

Registration of the component as a transport provider

Adapter Framework Configuration Schema Extensions

The BizTalk Adapter Framework supports the dynamic generation of user interfaces based on an XSD definition. The adapter supplies the required XSD and the BizTalk Adapter Framework creates a property page that allows the user to enter values.

To create custom user interfaces, the BizTalk Adapter Framework provides several extensions. To use these extensions, you must import the Adapter Framework schema (BizTalkAdapterFramework.xsd). By importing the schema, you can access and use the decorations and specialized types in the adapter configuration schema.

Use the decoration tags and built-in types discussed in this topic to customize the property grid for an adapter.

This section contains:

- Enabling Adapter Framework Configuration Extensions
- Adapter Framework Configuration Schema Decoration Tags
- Built-In Types for Adapter Framework Configuration Schemas
- Examples of Custom Configuration Schemas for Adapters

Enabling Adapter Framework Configuration Extensions

The BizTalk Adapter Framework provides several extensions to improve the user experience. To use these extensions, import the framework's schema BiztalkAdapterFramework.xsd. Importing the schema enables you to access decorations and specialized types and to use them in the adapter's configuration schema. (For more information, see "Import of BizTalk Adapter Framework extensions schema XSD" later in this document.) The following code shows how to import the schema.

Import of BizTalk Adapter Framework extensions schema XSD

By importing the Adapter Framework extensions schema XSD, you can use decorations such as <baf:FileName> as an element's type, which shows the file name pop-up when editing the element.

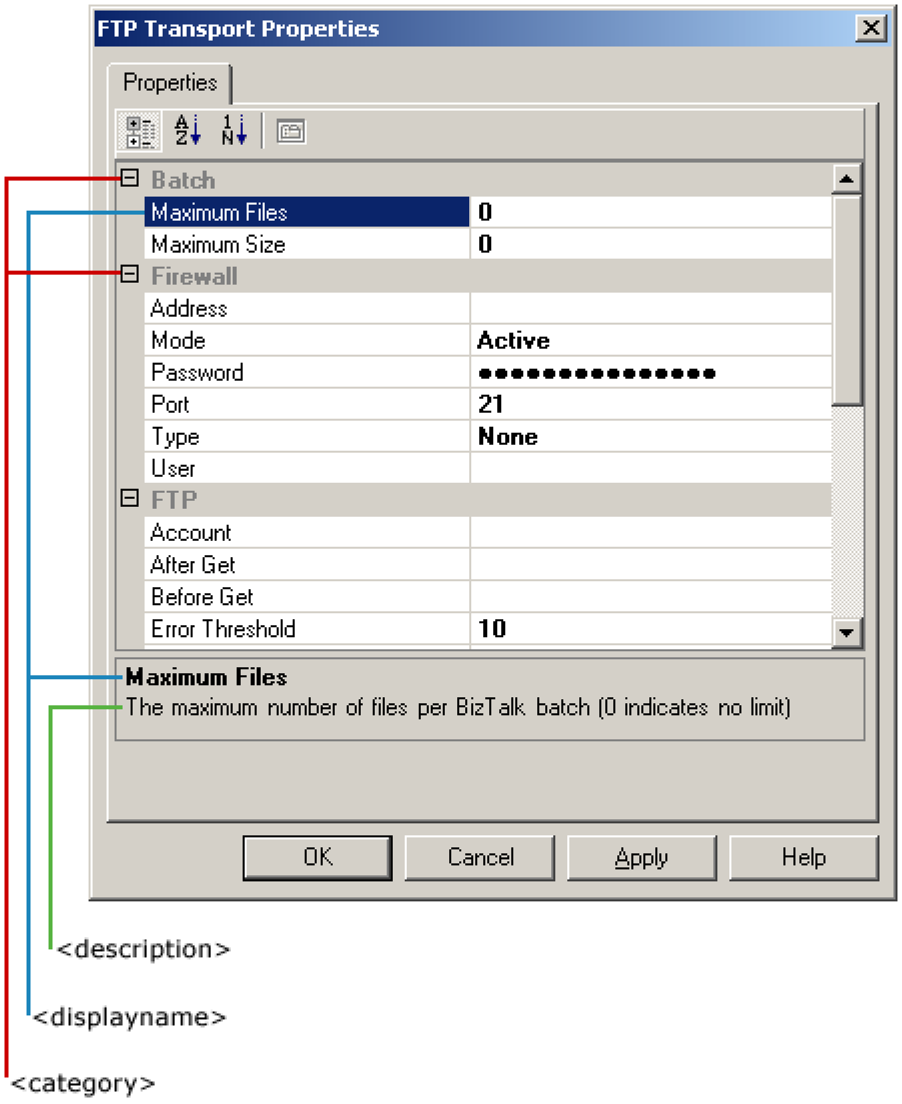
Additional decorations control the display of the property in the interface. The <baf:description>, for example, adds help text to the element. The <baf:description> decoration displays the text at the bottom of the property page. The <baf:browsable> hides an element from the interface. The following code shows how you can use these elements within a configuration schema.

Adapter Framework Configuration Schema Decoration Tags

You can use the tags described in this topic within the configuration schema files to display and organize data on the adapter property pages.

The following figure shows how the FTP receive location property page implements some of these tags.

Illustrates the <description>, <displayname> and <category> tags in the FTP adapter property page



Batch	
Maximum Files	0
Maximum Size	0
Firewall	
Address	
Mode	Active
Password	●●●●●●●●●●●●●●●
Port	21
Type	None
User	
FTP	
Account	
After Get	
Before Get	
Error Threshold	10

Maximum Files
The maximum number of files per BizTalk batch (0 indicates no limit)

<designer>

The BizTalk Adapter Framework decorations appear between a <baf:designer> tag and </baf:designer> end tag. The <baf:designer> tag helps distinguish between BizTalk Adapter Framework <appinfo> and other adapter-supplied <appinfo> information.

<category>

The <category> decoration contains the string used to separate entries in the property grid into groups. The decoration displays all entries with the same category string as subordinate entries of the category.

You can localize <category> entries.

<description>

The <description> decoration contains the string used to provide descriptive text for entries at the bottom of the property grid.

You can localize <description> entries.

<displayname>

The <displayname> decoration contains the string used for displaying the name of an entry. This enables you to use spaces, phrases, and so on when they would not be allowed for an <element> or <attribute> name.

You can localize <displayname> entries.

<readonly>

The <readonly fixed=""> decoration controls whether a field may be edited. A "fixed" attribute value of **true** (the default) makes a field read-only.

When implementing an external editor, implement an external **TypeConverter** and override the **GetStandardValuesExclusive(ITypeDescriptorContext)** method instead. Returning **true** makes a field read-only but preserves access to the custom editor.

<browsable>

The <browsable show=""> decoration controls whether a field appears in the property grid. A "show" attribute value of **True** (the default) makes a field appear in the grid.

<converter>

The <converter assembly=""> decoration specifies the desired **TypeConverter** class name for the <element> or <attribute>. The optional "assembly" attribute value specifies the path to the assembly containing the desired **TypeConverter**. With no "assembly" value specified, the class name must include the global assembly cache's assembly name, key, culture, and version values.

<editor>

The <editor assembly=""> decoration specifies the desired **UITypeEditor** class name for the <element> or <attribute>. The optional "assembly" attribute value specifies the path to the assembly containing the desired **UITypeEditor**. With no "assembly" value specified, the class name must include the global assembly cache's assembly name, key, culture, and version values.

Null values for optional enumerations

When using an optional enumeration, one of the values should be <none> to denote a null value. This is not a built-in tag, but may be achieved by providing an enumeration value that is treated as none if the enumeration is derived from xsd:string. This is not possible for enumerations derived from xsd:int, because the integer must hold a value.

Built-In Types for Adapter Framework Configuration Schemas

The BizTalk Adapter Framework also supplies some built-in simple types. These types, based on the string type, include a specialized **TypeConverter**, **UITypeEditor**, or both.

⊟baf:FileList

Setting an <element> or <attribute> type to "baf:FileList" places an ellipsis () button.

Clicking this button displays an **OpenFile** common control to select a file from the file system.

⊟baf:Password

Setting an <element> or <attribute> type to "baf:Password" displays a pop-up dialog box ellipsis () button.

Clicking this button displays a **Password** dialog box to specify a password.

By default, baf:Password allows 22 characters. Example 3 in Examples of Custom Configuration Schemas for Adapters shows how to limit this type to 8 characters.

baf:SSOList

Setting an <element> or <attribute> type to "baf:SSOList" displays a combo box of application names for use for Single Sign-On authentication. The list contains the entry "(none)" for an empty list of application names.

Examples of Custom Configuration Schemas for Adapters

The first example, Example 1,shows a complete custom XSD schema file representing an adapter property page using the baf:designer and baf:description extensions.

Example 1

The second example, Example 2, also uses the baf:designer and baf:description extensions to show how to create a custom property value window for the **BatchSize** property that will only accept values between 1 and 99, inclusive.

Example 2

The third example shows how to limit the baf:Password to 8 characters. By default, it allows 22 characters.

Example 3

The fourth sample shows how to implement pattern-matching constraints on property values because XSD patterns are not supported.

Fields such as ClientIdentifier are always a 3 character numeric string. A property value of 10 is not valid, whereas 010 is valid. The following configuration schema fragment defines a ClientIdentifier property. The ClientIdentifierConverter class, implemented in your adapter assembly, implements pattern matching. In this case, the custom type converter restricts values to a string of exactly three digits (000 to 999). In the configuration schema, make sure the baf:converter node has the assembly and type full name set correctly. If the user attempts to enter a value that is not valid, an exception message pops up when a validation error occurs in the property page.

Example 4

You can use the following code in the adapter property page configuration schemas.

You can place the following code in your adapter assembly.

Supported Adapter XSD Element Types

The following table lists the elements supported by the Adapter Framework. When defining a new element in your configuration schema, use any of the following types to replace `string` in the following example:

Type	**XML type**	**UI behavior**	**Other specifics**
string	None	Edit box accepting type only.	Attribute to constrain max/min
normalizedString	None	Edit box accepting type only.	Attribute to constrain max/min
integer	None	Edit box accepting type only.	Attribute to constrain max/min

positiveInteger	None	Edit box accepting type only.	Attribute to constrain max/min
negativeInteger	None	Edit box accepting type only.	Attribute to constrain max/min
nonNegativeInteger	None	Edit box accepting type only.	Attribute to constrain max/min
nonPositiveInteger	None	Edit box accepting type only.	Attribute to constrain max/min
int	None	Edit box accepting type only.	Attribute to constrain max/min
unsignedInt	None	Edit box accepting type only.	Attribute to constrain max/min
long	None	Edit box accepting type only and a decimal.	Attribute to constrain max/min
unsignedLong	None	Edit box accepting type only and a decimal.	Attribute to constraint max/min
short	None	Edit box accepting type only.	Attribute to constrain max/min
unsignedShort	None	Edit box accepting type only.	Attribute to constrain max/min
decimal	None	Edit box accepting type only.	Attribute to constrain max/min
float	None	Edit box accepting type only.	Attribute to constrain max/min
double	None	Edit box accepting type only.	Attribute to constrain max/min
boolean	None	Drop-down list populated with Boolean values.	None
time	None	Edit box accepting type only.	None
dateTime	None	Edit box accepting type only. An ellipsis appears at the end of the field	None

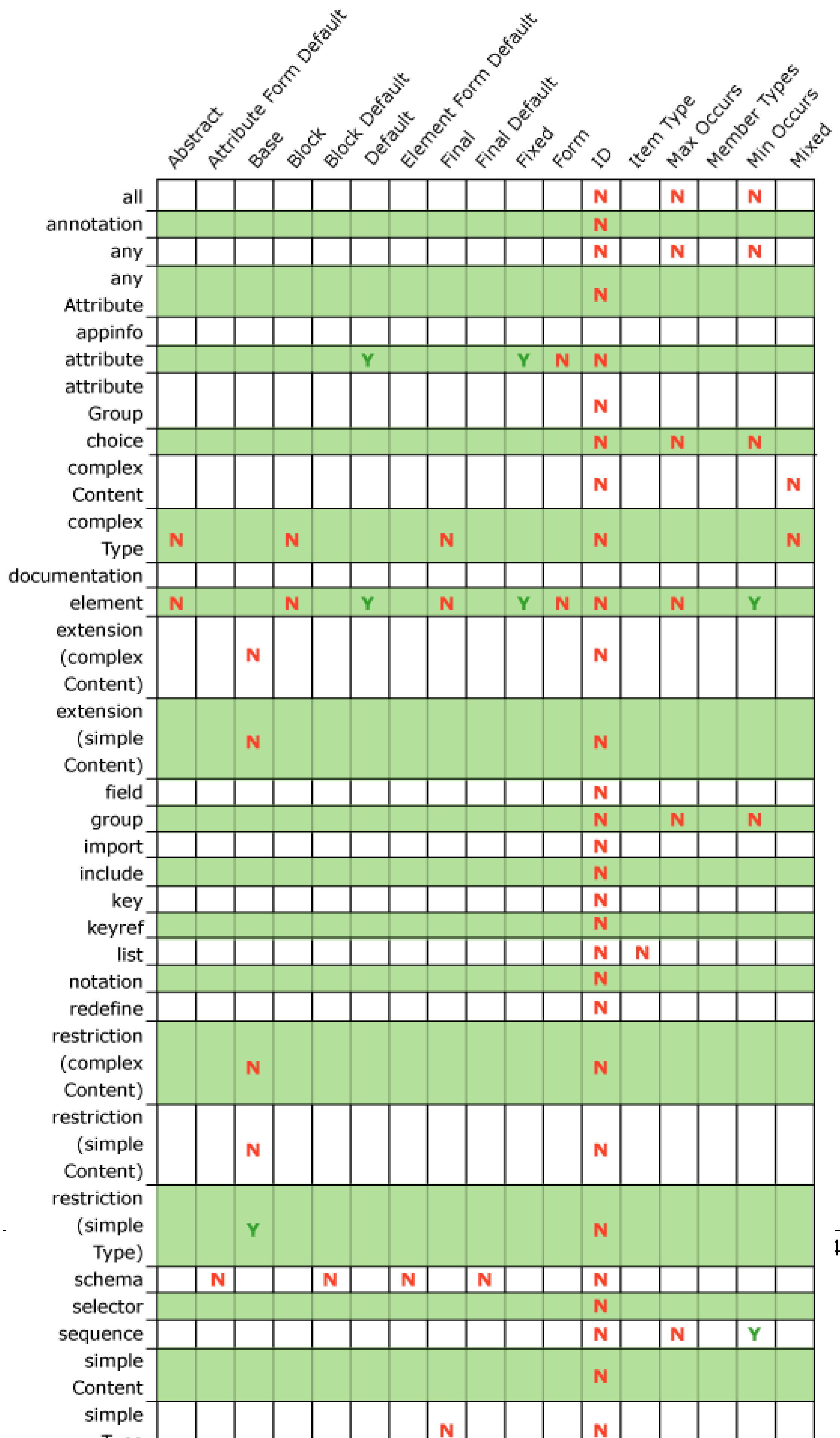
		area. Click the ellipsis and the calendar appears.	
date	None	Edit box accepting type only. An ellipsis appears at the end of the field area. Click the ellipsis and the calendar appears.	None
gMonth	None	Edit box accepting type only.	This value is a string and thus may not perform as expected. Consider using xsd:int types with restrictions to hold the month value instead.
gYear	None	Edit box accepting type only.	This value is a string and thus may not perform as expected. Consider using xsd:int types with restrictions to hold the year value instead.
gYearMonth	None	Edit box accepting type only.	This value is a string and thus may not perform as expected. Consider using xsd:int types with restrictions to hold the year and month value instead.
gDay	None	Edit box accepting type only.	This value is a string and thus may not perform as expected. Consider using xsd:int types with restrictions to hold the day value instead.
gMonthDay	None	Edit box accepting type only.	This value is a string and thus may not perform as expected. Consider using xsd:int types with restrictions to hold the month and day value instead.

Name	None	Edit box accepting type only.	None
NCName	None	Edit box accepting type only.	None
anyURI	None	Edit box accepting type only.	None
Sequence	"Sequence" Schema Element	None	None
Groups	None	A "+" or "-" sign that expands or collapses all fields within the group. No edit functionality on the right side of the property page.	None
File Name	FileName	An ellipsis appears at the end of the field area. Click the ellipsis and the **Windows FileOpen** dialog box appears.	None
SSO App ID	SSOAppID	Drop-down list populated with the SSO Application list	None
Password	Password	Edit box with "*" appearing instead of clear text.	None

Adapter XSD Element-Attribute Constructs

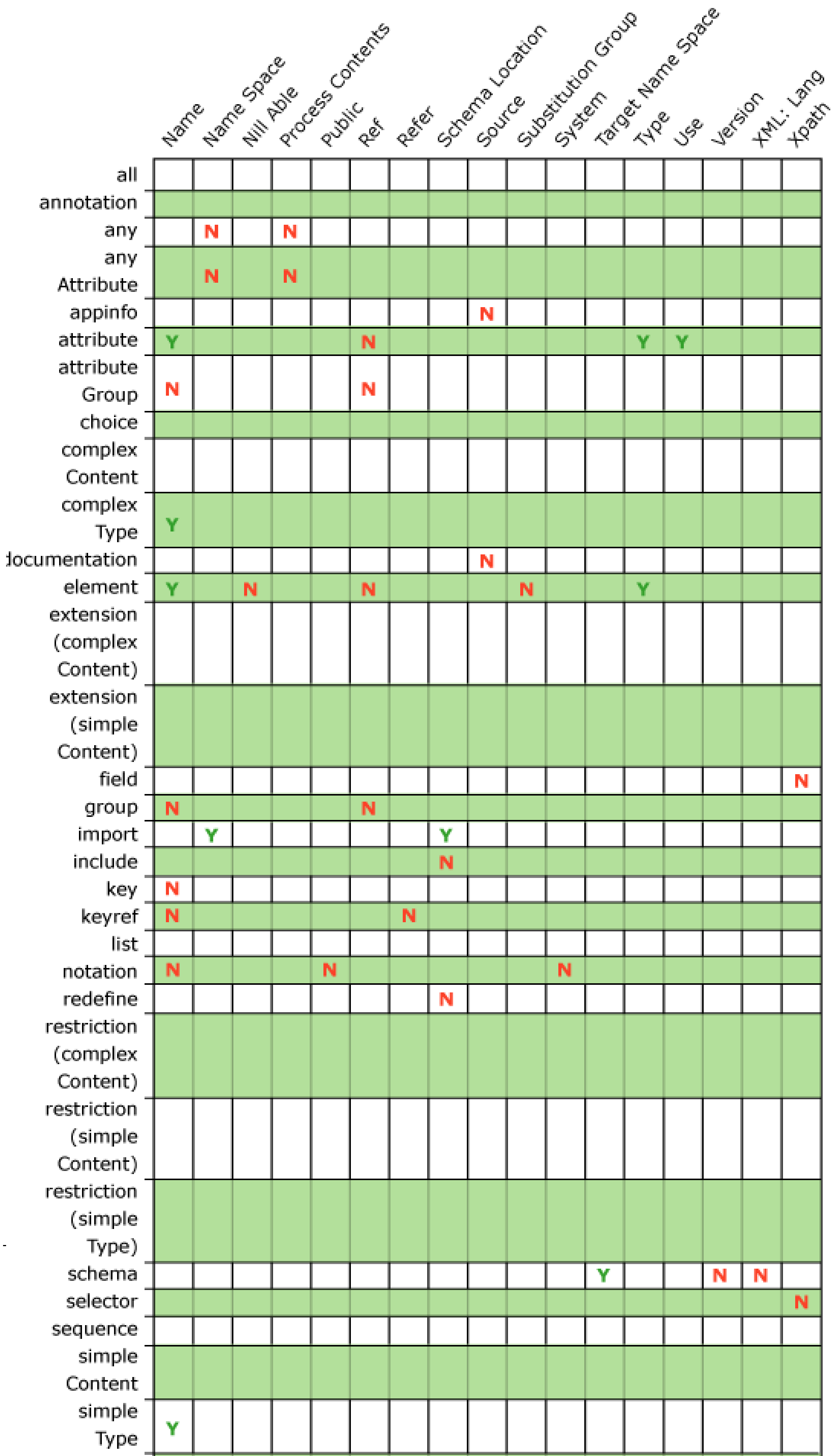
The following table lists XSD element-attribute constructs supported by the Adapter Framework. The vertical axis contains the supported element types, and the horizontal axis contains the valid attribute types for each element.

Illustrates the supported Adapter XSD Element-Attribute constructs



	Abstract	Attribute Form Default	Base	Block	Block Default	Default	Element Form Default	Final	Final Default	Fixed	Form	ID	Item Type	Max Occurs	Member Types	Min Occurs	Mixed
all												N		N		N	
annotation												N					
any												N		N		N	
any Attribute												N					
appinfo																	
attribute						Y				Y	N	N					
attribute Group												N					
choice												N		N		N	
complex Content												N					N
complex Type	N			N				N				N					N
documentation																	
element	N			N		Y		N		Y	N	N		N		Y	
extension (complex Content)			N									N					
extension (simple Content)			N									N					
field												N					
group												N		N		N	
import												N					
include												N					
key												N					
keyref												N					
list												N	N				
notation												N					
redefine												N					
restriction (complex Content)			N									N					
restriction (simple Content)			N									N					
restriction (simple Type)			Y									N					
schema		N			N		N		N			N					
selector												N					
sequence												N		N		Y	
simple Content												N					
simple								N				N					

Illustrates the supported Adapter XSD Element-Attribute constructs

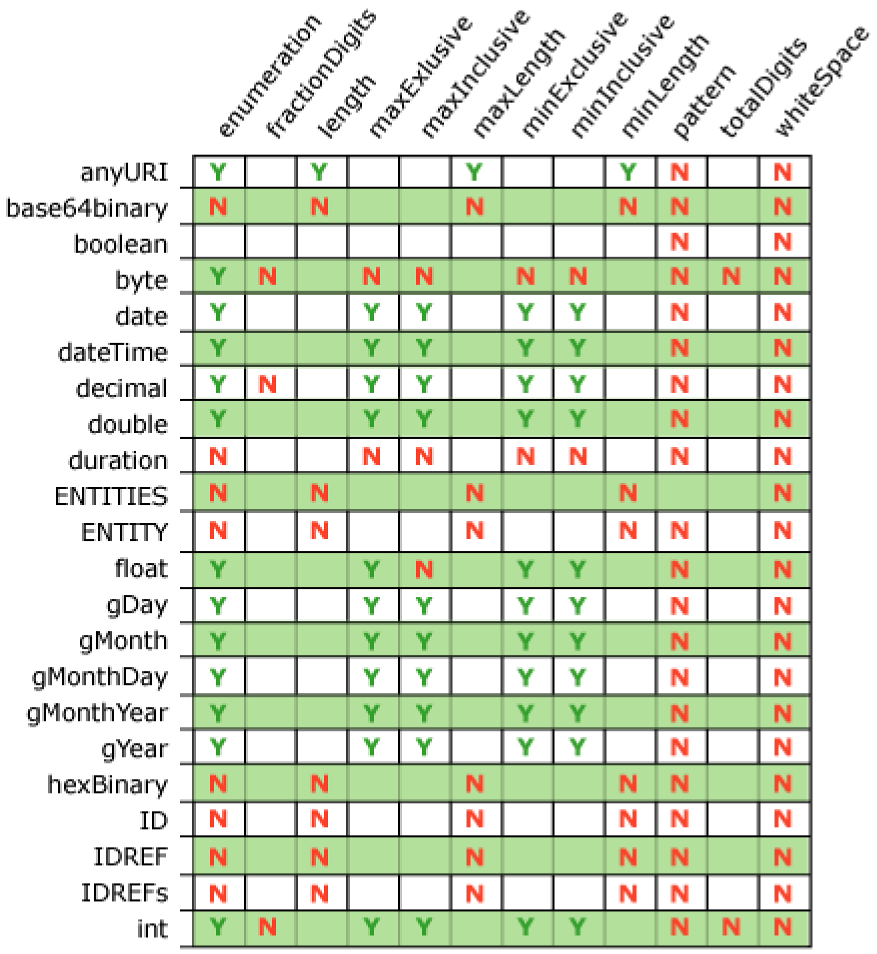


	Name	Name Space	Nill Able	Process Contents	Public	Ref	Refer	Schema Location	Source	Substitution Group	System	Target Name Space	Type	Use	Version	XML: Lang	Xpath
all																	
annotation																	
any		N		N													
any Attribute		N		N													
appinfo									N								
attribute	Y					N							Y	Y			
attribute Group	N					N											
choice																	
complex Content																	
complex Type	Y																
documentation									N								
element	Y		N			N				N			Y				
extension (complex Content)																	
extension (simple Content)																	
field																	N
group	N					N											
import		Y						Y									
include								N									
key	N																
keyref	N						N										
list																	
notation	N				N						N						
redefine								N									
restriction (complex Content)																	
restriction (simple Content)																	
restriction (simple Type)																	
schema												Y			N	N	
selector																	N
sequence																	
simple Content																	
simple Type	Y																

Adapter XSD Data Type-Facet Constructs

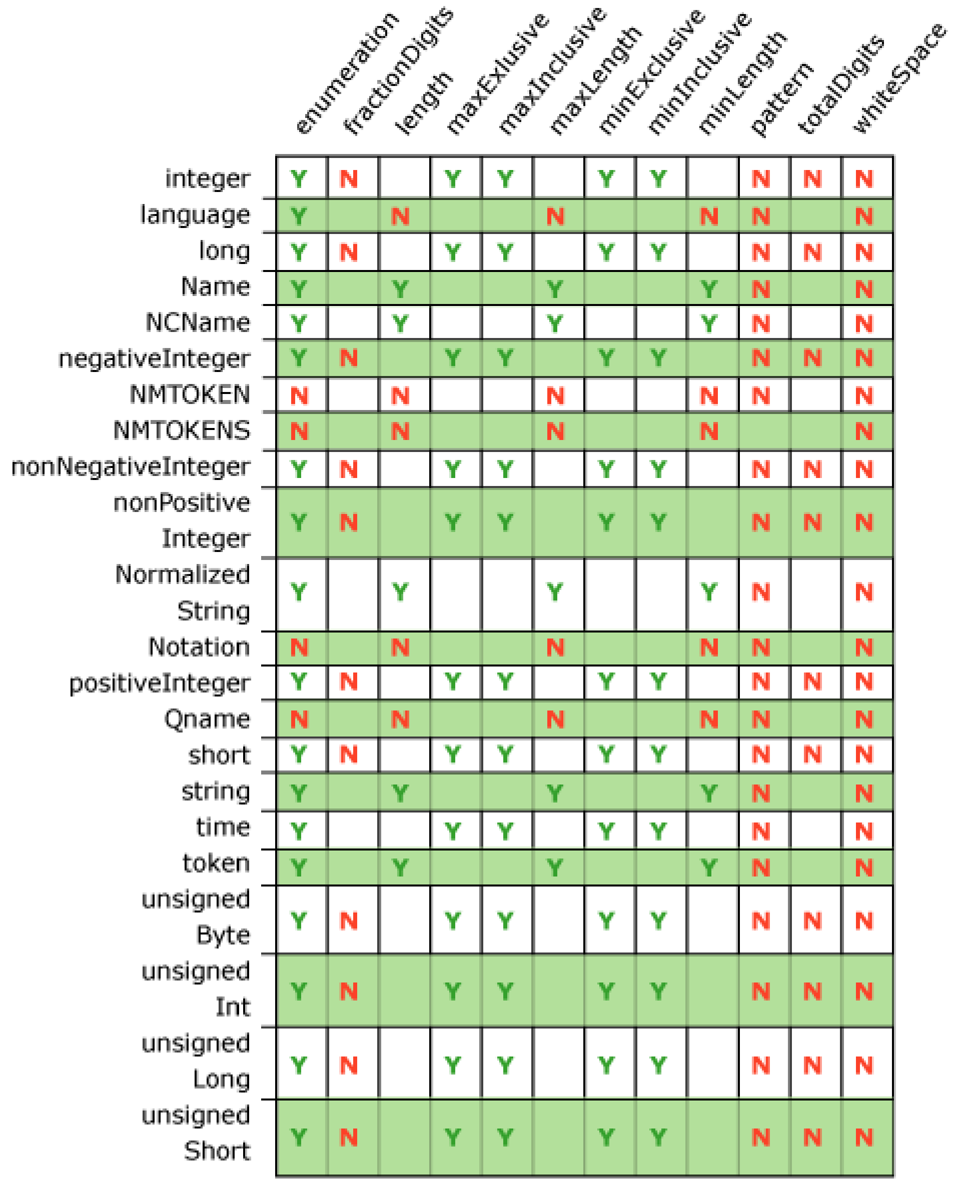
The following table lists XSD data type-facet constructs supported by the Adapter Framework. The vertical axis contains the supported data types, and the horizontal axis contains the valid facets for each data type.

Illustrates the supported XSD data type-facet constructs



	enumeration	fractionDigits	length	maxExlusive	maxInclusive	maxLength	minExclusive	minInclusive	minLength	pattern	totalDigits	whiteSpace
anyURI	Y		Y			Y			Y	N		N
base64binary	N		N			N			N	N		N
boolean										N		N
byte	Y	N		N	N		N	N		N	N	N
date	Y			Y	Y		Y	Y		N		N
dateTime	Y			Y	Y		Y	Y		N		N
decimal	Y	N		Y	Y		Y	Y		N		N
double	Y			Y	Y		Y	Y		N		N
duration	N			N	N		N	N		N		N
ENTITIES	N		N			N			N			N
ENTITY	N		N			N			N	N		N
float	Y			Y	N		Y	Y		N		N
gDay	Y			Y	Y		Y	Y		N		N
gMonth	Y			Y	Y		Y	Y		N		N
gMonthDay	Y			Y	Y		Y	Y		N		N
gMonthYear	Y			Y	Y		Y	Y		N		N
gYear	Y			Y	Y		Y	Y		N		N
hexBinary	N		N			N			N	N		N
ID	N		N			N			N	N		N
IDREF	N		N			N			N	N		N
IDREFs	N		N			N			N	N		N
int	Y	N		Y	Y		Y	Y		N	N	N

Illustrates the supported XSD data type-facet constructs



	enumeration	fractionDigits	length	maxExclusive	maxInclusive	maxLength	minExclusive	minInclusive	minLength	pattern	totalDigits	whiteSpace
integer	Y	N		Y	Y		Y	Y		N	N	N
language	Y		N			N			N	N		N
long	Y	N		Y	Y		Y	Y		N	N	N
Name	Y		Y			Y			Y	N		N
NCName	Y		Y			Y			Y	N		N
negativeInteger	Y	N		Y	Y		Y	Y		N	N	N
NMTOKEN	N		N			N			N	N		N
NMTOKENS	N		N			N			N			N
nonNegativeInteger	Y	N		Y	Y		Y	Y		N	N	N
nonPositive Integer	Y	N		Y	Y		Y	Y		N	N	N
Normalized String	Y		Y			Y			Y	N		N
Notation	N		N			N			N	N		N
positiveInteger	Y	N		Y	Y		Y	Y		N	N	N
Qname	N		N			N			N	N		N
short	Y	N		Y	Y		Y	Y		N	N	N
string	Y		Y			Y			Y	N		N
time	Y			Y	Y		Y	Y		N		N
token	Y		Y			Y			Y	N		N
unsigned Byte	Y	N		Y	Y		Y	Y		N	N	N
unsigned Int	Y	N		Y	Y		Y	Y		N	N	N
unsigned Long	Y	N		Y	Y		Y	Y		N	N	N
unsigned Short	Y	N		Y	Y		Y	Y		N	N	N

Advanced Configuration Components for Adapters

The BizTalk Adapter Framework has support for a custom drop-down editor, a custom modal dialog editor, and a custom type converter. These custom design components are especially useful when taking user name and password information as input.

You can invoke a custom editor or type converter for a specific data field (element or attribute) in the configuration schema. As described in the Microsoft® Visual Studio® .NET Help, the editor must be derived from

System.Drawing.Design.UITypeEditor and the type converter from **System.ComponentModel.TypeConverter**.

An editor minimally overrides the **GetEditStyle** method to specify the kind of editor (**Modal** dialog, **DropDown** control, or **None** of the above) and the **EditValue** method to change the value with the editor.

A type converter typically overrides the **ConvertFrom**, **CanConvertFrom**, **ConvertTo,** **CanConvertTo**, **GetStandardValues**, **GetStandardValuesSupported**, and **GetStandardValuesExclusive** methods of the .NET Framework Class Library.

This section contains:

- Custom Adapter Configuration Designer
- Custom Drop-Down Editor for Adapter Configuration
- Custom Modal Dialog Editor for Adapter Configuration
- Custom Type Converter for Adapter Configuration

Custom Adapter Configuration Designer

You will need to build the custom designers into a .Net class library. You may incorporate them into the DLL for the adapter or build a separate DLL. After you build a designer assembly, you must reference it through decorations, just like a description or a category. The reference includes a specification of the assembly and a fully qualified class name to use.

These decorations support two ways of referencing the specific custom designer: as a global assembly in the global assembly cache or as an external assembly located on the disk.

Global assembly cache designer use

The global assembly cache stores assemblies by assembly name, public key, version, and culture. Because of this, it is recommended that you:

1. Generate a public key file and add this file to the AssemblyInfo.cs file.
2. Specify a specific version in the AssemblyInfo.cs file.

You can either drag the assembly into the global assembly cache or use GACUTIL to add it to the global assembly cache.

To use this designer, specify the fully qualified class name, a comma, and the global assembly cache assembly entry (assembly name, version, culture, and public key token) as the value of the decoration. Use <editor> decorations for **UITypeEditor** implementations and <converter> decorations for **TypeConverter** implementations.

The following code shows how to initialize the custom designers in an XSD file.

External assembly installation and use

For external assemblies, the decoration contains an optional attribute assembly that specifies the full path and name of the assembly containing the desired designer.

The following code shows how to initialize the custom designers contained in external assemblies.

Custom Drop-Down Editor for Adapter Configuration

The code for the custom editor shows an editor derived from the **System.Drawing.Design.UITypeEditor** class that displays a drop-down text box for entering a password. The **GetEditStyle** override returns **UIEditorEditStyle.DropDown** to indicate a drop-down subcontrol. The service methods **DropDownControl** and **CloseDropDown** manage the control created with **CreatePassword**.

Custom Modal Dialog Editor for Adapter Configuration

The code for the custom editor shows an editor derived from the **System.Drawing.Design.UITypeEditor** class that displays a modal pop-up dialog box for entering a password. The **GetEditStyle** method override returns **UIEditorEditStyle.Modal** to indicate a modal form subcontrol. The service method **ShowDialog** manages the control created with **CreatePassword**. **ShowDialog** returns a **DialogResult** that is handled in the usual way (for example, a switch statement) with the **DialogResult.OK** case changing value only.

Custom Type Converter for Adapter Configuration

Like the custom editor, the custom type converter overrides the **System.ComponentModel.TypeConverter** class of one of its children. Here, the converter adds formatting to the value to be persisted but does not appear on the property page. The **ConvertFrom** method adds square brackets around the string value and the **ConvertTo** method removes them.

Receiving messages

Receive Adapters receive data from the ‘wire’ and submit it as a message into BizTalk. This submittal process can be a one-way or a two-way message exchange pattern.

One way Submit

For a receive adapter to submit a message into the engine, it first needs to create a new BizTalk message. The code sample in the **IBaseMessage** section illustrates how this is done. The stream that the adapter sets as the message body should typically be a forward-only stream, meaning that it should not cache the data that it has previously read into memory. Before the adapter submits the message into the engine, it must write the InboundTransportLocation message context property in the system namespace onto the BizTalk message. This is illustrated in the following code snippet. In addition, the adapter may define its own property schema and write message context properties pertaining to the endpoint over which it received the message. For example, and HTTP adapter may wish to write the HTTP headers to the message context, and a SMTP receiver may want to write the subject of the mail to the message context. This information may be useful to components down stream such as pipeline components, orchestration schedules or a transmit adapter.

Assembly References:

Microsoft.XLANGs.BaseTypes.dll

Microsoft.BizTalk.Pipeline.dll

Microsoft.BizTalk.GlobalPropertySchemas.dll

Once the messages are prepared they can be submitted into the Messaging Engine. The code sample in SubmitDirect (BizTalk Server Sample) indicates how a one way receive adapter would submit a message into the engine.

The following object interaction diagram illustrates the API calls:

Request-Response

Two-way receive adapters may typically be used on one-way or a two-way receive ports. The adapter determines whether the receive location it is servicing is a one-way or two-way port by inspecting the BizTalk configuration property bag as detailed in **AddReceiveEndpoint**.

The object interaction diagram below illustrates the process of performing a request-response. The adapter request's a new batch from it's transport proxy and passes in it's reference to an IBTTransmitter interface calls via the API SubmitRequestMessage(). The Messaging Engine will deliver the response message on this interface using it's TransmitMessage method.

It should be noted that because the engine is processing messages asynchronously, it is possible for (a) the BatchComplete() callback to take place before Done() has returned, and (b) the call to TransmitMessage() could be made before BatchComplete() and even Done(). Whilst both of these scenarios are rare, the adapter should protect itself against this.

It is recommended that the response message is transmitted using a non-blocking transmission as detailed in the Sending Messages sections.

The BaseAdapter in the SDK refresh has a utility class, StandardRequestResponseHandler that encapsulates the request-response semantics as detailed above.

Request-Response Message Timeouts

When an adapter submits a request-request message it needs to specify the timeout of the request message on the IBTTransportBatch.SubmitRequestMessage() API A response message will only be delivered to the adapter within this timeout period. After the timeout expires a Negative Acknowledgment (NACK) will be delivered to the adapter instead of the response message. If the adapter does not specify a timeout value the engine will use the default value of 20 minutes.

The default timeout for request-response messages maybe controlled using the following registry key for receive adapters:

DWORD

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\BTSSvc{Host Guid}\MessagingReqRespTTL

The registry key is in a different location for Isolated receive adapters:

DWORD

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\BTSSvc.3.0\Messaging ReqRespTTL

NACK's are SOAP Faults and there are potentially two ways to handle them. Typically the adapter will return the NACK back to the client and the client will deal with the NACK. Alternatively the transmit pipeline handling the response message may be configured to change the shape of the response message, either using a map or a custom pipeline component.

Sending Messages

Transmit adapters are delivered messages from the engine to be transmitted over the 'wire', These messages may be sent using a one-way or two-way message exchange pattern. The latter is term a solicit-response adapter.

Blocking vs Non-Blocking Transmissions

The Messaging Engine will deliver messages to the transmit adapter either in IBTTransmitter.TransmitMessage() or IBTTransmitterBatch.TransmitMessage() depending on whether or not the adapter is batch aware or not. Both versions of the method have a boolean return value which indicates how the adapter transmitted the message. If it returns true, it has completely sent the message before returning. In this case the Messaging Engine will delete the message from the MessageBox database on behalf of the adapter. If the adapter returns false, it started message transmission and returned before the transmission has completed. In this case, the adapter is responsible not only for deleting the message from its application queue but also for handling any transmissions failures that require the message to be retried for transmission or suspended.

The adapter returning false is a non-blocking call meaning that the TransmitMessage() implementation code does not block the calling thread of the messaging engine. The adapter simply adds the message to an in-memory queue ready to be transmitted and returns. The adapter should have its own thread pool which will service the in-memory queue, transmit the message, and then notify the engine of the outcome of the transmission.

The Messaging Engine’s threads are is typically more CPU bound than the threads used to send data over the wire. Mixing these two types of threads have a negative impact on performance. Non-blocking sends enable the decoupling of these two types of thread usage and yield a significant performance improvement over blocking calls.

Performance Tip: For the best performance, transmit adapters should be non-blocking and batch aware. When the BizTalk File adapter was changed from blocking + non-batch aware to non-blocking + batch aware a three times performance gain was realized.

Trouble Shooting Tip: Blocking transmits can cause a performance degradation of an entire host instance. If the adapter does excessive blocking in TransmitMessage() it will be preventing engine threads from delivering messages to other adapters.

Non-Batched Sends

Adapters that are not batch aware should implement IBTTransmitter as detailed in Transmitter Interfaces section. For each message that the adapter needs to transmit the Messaging Engine will call IBTTransmitter.TransmitMessage(). The object interaction diagram below details the typical approach for transmitting messages, the steps involved are as follows:

1. The engine delivers the message to the adapter

2. The adapter en-queues the message to an in-memory queue ready to be transmitted

3. A thread from the adapters thread pool de-queues the message from the queue, it reads the configuration for the message and transmits the message over the ‘wire’

4. The adapter gets a new batch from the engine

5. The adapter calls DeleteMessage() on the batch passing in the message that it has just transmitted

6. The adapter calls Done() on the batch

7. The engine processes the batch and deletes the message from the application queue

8. The engine calls back the adapter to notify it of outcome of the DeleteMessage() operation

The above object interaction diagram shows the adapter deleting a single message from the application queue. Ideally the adapter would batch up operations on the batch as opposed to simply deleting a single message at a time.

Batched Sends

Adapters that are batch aware should implement IBTBatchTransmitter and IBTTransmitterBatch as detailed in Transmitter Interfaces section. When the engine has messages for the adapter to transmit the engine gets a new batch from the adapter by calling IBTBatchTransmitter.GetBatch(). The adapter returns a new batch object which implements IBTTransmitterBatch. The engine will then start the batch by calling IBTTransmitterBatch.BeginBatch(). This API has an out parameter which allows the adapter to specify the maximum number of messages that it will accept on the batch. The adapter may optionally return a DTC transaction. The engine will then call IBTTransmitterBatch.TransmitMessage() once for each outgoing message to be added to the batch. The number of times this is called will be greater than zero but less than or equal to the maximum size of the batch as indicated by the adapter. When all the messages have been added to the batch, the adapter will call IBTTransmitterBatch.Done(). At this point the adapter should typically en-queue all the messages in the batch to it's in-memory queue. The adapter will transmit the messages from a thread or threads in its own thread pool as in the case of non-batch aware adapters, after which it will notify the engine of the outcome of the transmission.

The following object interaction diagram illustrates the transmission of two messages by a batched transmit adapter.

Batched sends.

Handling Transmission Failures

The recommended semantics for transmissions failures are illustrated in the diagram below. It should be noted that these are recommendations and not enforced by the Messaging Engine. An adapter can deviate from these if there are valid reasons for doing so but care should be taken if doing so. For example, in general an adapter should always move messages to the backup transport once all retries have been exhausted. Failure to do so deviates from the design time configuration setting whereby a user configures a backup transport for a port that would effectively never be used for that transport by the adapter. This is misleading the user to the level of transport resilience.

More commonly a transport may need to use additional retries than are configured. While this is subtlety different it is considered acceptable since the resilience of the transport layer is being increased. In general the API's exposed by the Messaging Engine are designed to give the adapter the maximum control where possible. With this control comes a greater level of responsibility.

The adapter determines the number of reties available on a message by checking the system context property RetryCount. The adapter calls the Resubmit() API once for each retry attempt passing in the message to be resubmitted. Along with the message is passed the time stamp indicating when the engine should deliver the message back to the adapter. This value of should typically be now + RetryInterval. RetryInterval is a system context property the units of which are minutes. Both the RetryCount and RetryInterval in the message context are of course the values that are configured on the send port. Consider a scaled out deployment with instances of the same BizTalk host deployed on multiple machines. If the message is delivered after the retry interval has expired the message may be delivered to the any one of the host instances on any of the machines where they are configured to run. For this reason the adapter should not hold any state associated with a message to be used on the retry attempt since there is no guarantee that same instance of the adapter will be responsible for the transmission at a later time.

The adapter should only attempt to move the message to the backup transport after the retry count is less than or equal to zero. An attempt to move the message to the backup transport will fail if there is no backup transport configured for the port. In this case the message should be suspended.

The following code snippet illustrates how to determine the retry count and interval from the message context, and the subsequent resubmit or move to backup transport.

Throwing an Exception from TransmitMessage

If the adapter throws an exception on any of the following API's IBTTransmitter.TransmitMessage(), IBTTransmitterBatch.TransmitMessage(), IBTTransmitterBatch.Done(), the engine will treat the transmission of those messages involved as transmission failures and take the appropriate action for the message (as detailed in Handling Transmission Failures).

For batch aware transmitters, throwing an exception on the TransmitMessage() API will result in the entire batch being cleared and the default transmit failure actions being performed for all messages in that batch.

Solicit-Response

Two-way send adapters typically support both one-way and two-way transmissions. The send adapter determines whether the message should be transmitted as a one-way or two-way send by inspecting the IsSolicitResponse system context property in the message context.

The following code snippet demonstrates this:

During a solicit-response transmission the adapter will transmit the solicit message. Once completed it will submit the associated response and tell the Messaging Enginer to delete the original solicit message from the MessageBox database. The action of deleting the message from the application queue should be performed in the same Transport Proxy batch as the submission of the response message. This ensures atomicity of the delete and submission of the response. Of course for complete atomicity the adapter should use a DTC transaction whereby the transmission of the solicit message to a transaction aware resource, submission of the response message and the deletion of the solicit message are all in the context of the same DTC transaction. As always, it is of course recommended that the solicit message is transmitted using a non-blocking send.

The following object interaction diagram illustrates the API calls for a solicit-response.

The following code snippet illustrates the main aspects of a two-way send. When IBTTransmitter.TransmitMessage() is called by the engine the adapter simply en-queues the message to be transmitted to an in memory queue. The adapter returns false to indicate that it is performing a non-blocking send. The adapters' thread pool (WorkerThreadThunk()) services the in-memory queue and de-queues a message to hand it off to a helper method. This method is responsible for sending the solicit message and receiving the response message. The implementation of this helper method is outside of this text. The response message is submitted into the engine, and the solicit message is deleted from the application queue.

Dynamic Sends

Dynamic send ports do not have adapter configuration associated with them. Instead they use handler configuration for any ‘default’ properties that the adapter needs in order to transmit messages on a dynamic port. For example, an HTTP adapter may need to use a proxy and need to provide credentials. The username, password and port could be specified in the handler configuration which the adapter caches at run time.

In order for the engine to determine the transport that a dynamic message is to be sent over, the OutboundTransportLocation will be prefixed with the adapters’ alias. An adapter can register one or more aliases with BizTalk at install time. The engine parses the OutboundTransportLocation at run-time in order to find a match. It is the adapters’ responsibility to handle the OutboundTransportLocation with or with the alias pre pended to it. The following table shows some examples of aliases registered for out of the box BizTalk adapters.

Adapter Alias	Adapter	OutboundTransportLocation example
HTTP://	HTTP	http://www. MyCompany.com/bar
HTTPS://	HTTP	https://www. MyCompany.com/bar
mailto:	SMTP	mailto:A.User@MyCompany.com
FILE://	FILE	FILE://C:\ MyCompany \%MessageID%.xml
DIRECT=	MSMQT	DIRECT=.\MyQueue

Batching Messages

When your adapter has a series of operations to perform at one time, you should *batch* these operations to optimize performance.

Programmatically, *message batches* are collections of operations with associated messages.

BizTalk Server uses batching to:

- Amortize the cost of the transaction across many messages.
- Increase speed by reducing the internal number of database round trips.
- Make more efficient use of the BizTalk Server thread pool by using the BizTalk Server Asynchronous API.

Batches are atomic

A batch is a unit of work that is atomic; that is, it either succeeds or fails. If one operation in a batch succeeds but another operation fails, all the operations that make up a batch are invalidated and must be repeated.

This means that an adapter must do two things in response to a failed batch:

- Determine what to do with the failed message
- Re-submit the successful messages

An adapter may provide the transaction object for a batch

There are two kinds of batches defined previously in this section: transactional batches and normal batches. These types are used in the following ways:

Normal batches

Not all adapters need to manage transactions externally. The FILE adapter is an example of an adapter that does not require access to the transaction, because the external file operations it manages are not transactional. In this case, the adapter does not provide a transaction object to BizTalk Server. If the adapter does not provide a transaction object to BizTalk Server as part of a batch of messages, BizTalk Server creates an internal transaction for it automatically. This can be thought of as a "normal" batch.

Transactional batches

Other adapters (such as the SQL adapter supplied with BizTalk Server) must coordinate an external SQL Server transaction with an internal BizTalk Server transaction. To do this, the adapter needs access to the BizTalk Server transaction object. A transaction object is created and and associated with the batch before the batch is submitted to BizTalk Server. This is a transactional batch. By supplying your own transaction object, you can achieve the "guaranteed, once and once only", delivery of data into and out of BizTalk Server.

Transactional database adapters like the SQL adapter that shipped with BizTalk Server 2006 have the potential for deadlocks in the external database because of the single transaction used for the batch. This is why the SQL adapter hard-coded its batch size to one.

Batch Call Backs

Since batches are processed asynchronously, the Adapter needs a mechanism to tie a callback to some state in the Adapter so that the Adapter can perform the necessary cleanup actions. The callback semantics are flexible such that Adapters can use one or a combination of three approaches, these are:

- All callbacks made on the same object instance that implements IBTBatchCallBack,
- The cookie passed into the engine when requesting a new batch is used to correlate the callback to the state in the Adapters
- Having a different callback object for each batch that implements IBTBatchCallBack, each object holds all the state necessary

A combination of the first two.

The Adapter is called back on it's implementation of IBTBatchCallBack.BatchComplete(). The overall status of the batch is indicated by the first parameter, status. This is an HRESULT, since batches are atomic, if this value is greater than or equal to zero, then the batch was successful in the sense that the engine has ownership of the data and the Adapter is free to delete that data from the 'wire'. A negative status indicates that the batch failed, this means that none of the operations in the batch were successful and the Adapter is responsible for handling the failure.

If the batch failed then the Adapter may need to know which item in which operation failed. For example, if an the Adapter was reading twenty files from disc and submitting them into BizTalk using a single batch, if the tenth file was corrupted, the Adapter would need to suspend that one file and resubmit the remaining nineteen. This information is available to the adapter in the second and third parameters, opCount of type short and operationStatus which is of type BTBatchOperationStatus[]. The operation count indicates how many types of operations were in the batch (the size of the array BTBatchOperationStatus array). Each element in the operation status array corresponds to a given operation type, using the BTBatchOperationStatus the Adapter can determine which item in a given operation failed by looking at BTBatchOperationStatus.MessageStatus[]. In this particular scenario, the adapter would then need to create a new batch containing nineteen submits and one suspend. The MessageStatus array is again an array of HRESULT's meaning that a failure code will be negative.

Batching messages

The following code snippet illustrates how an Adapter would requests a new batch from the engine via it's Transport Proxy, and submits a single message into the engine, this snippet uses the cookie approach:

Information: The BizTalk Server SDK at the time of writing this article includes three Adapter samples, FILE, HTTP and MSMQ, all three of these Adapters are built on top of a common building block call the BaseAdapter. In v1.0.1 of the BaseAdapter all of the relevant code to parse the operation status and rebuild a new batch to submit is included.

Batch Status codes

As mentioned the overall batch status code indicates the outcome of the batch, drilling down to the operation level gives the individual item's status. Batches can fail for various reasons, in general when the engine is shutting down, the engine will not accept any new work but will allow in flight work to be completed. The following table indicates some common HRESULT's that returned either in the batch status or the operation status it also indicates whether these are SUCCESS or FAILURE codes.

The proper handling of these codes is described more fully in the topic "Handling failure" later in this section.

Code (defined in the class BTTransportProxy)	Success/Failure Code	Description
BTS_S_EPM_SECURITY_CHECK_FAILED	Success	The port was configured to perform a security check and drop messages that failed authentication. Adapters should not

		suspend batches that return this status code
BTS_S_EPM_MESSAGE_SUSPENDED	Success	Indicates that one or more messages were suspended, the engine has ownership of the data, hence the success code
E_BTS_URL_DISALLOWED	Failure	A message was submitted with an invalid InboundTransportLocation
BTS_E_MESSAGING_SHUTTING_DOWN	Failure	The engine is shutting down and will therefore not accept new work

⊟Batch Operations

As the previous section illustrated Adapters add work to a given batch using the API's of the batch. The following table details the methods and there corresponding operations:

Method Name	**Operation Type**	**Description**
SubmitMessage	Submit	Submits a message into the engine
SubmitResponseMessage	Submit	Submits a response message into the engine, this is the response message in a solicit-response pair
DeleteMessage	Delete	Deletes a message that an Adapter has successfully transmitted over the wire using a non-blocking send. Adapters that use blocking sends do not need to call this API
MoveToSuspendQ	MoveToSuspendQ	Suspends a message
Resubmit	Resubmit	Requests that a message should be retried for transmission at a later time, typically called after a transmission attempt had

		failed
MoveToNextTransport	MoveToNextTransport	Requests that a message be transmitted using it's backup transport, typically called after all retries have been exhausted. If no backup transport is present this API will fail
SubmitRequestMessage	SubmitRequest	Submits a request message, this is a request message in a request-response pair
CancelRequestForResponse	CancelRequestForResponse	Notify the engine that the Adapter no longer wishes to wait for the response message in a request-response pair
Clear	NA	Clears all the work from the batch
Done	NA	Request's that the engine processes this batch of work

Releasing the Batch

Batches are implemented in native code in the engine, for this reason an Adapter written in managed code should release the runtime callable wrapper (RCW) after it has finished with the batch.

This is done in managed code using Marshal.ReleaseComObject(), it is important to remember that this API should be called in a while loop until the returned reference count reaches zero.

Messages are processed synchronously with a batch but many batches may be processed concurrently

Messages are processed synchronously within a batch by BizTalk Server, but many batches may be processed concurrently. This may provide an opportunity for some optimization in the adapter. There may be a optimal batch size to look for in the application domain; for example, all the files on an FTP site (until some limit is hit). In the case of SAP you might process a single stream into a number of messages which are then submitted as a batch.

The adapter determines the messages it puts in the different batches

The batch used by an adapter to pass operations back to BizTalk Server does not need to exactly correspond to the list of messages that BizTalk Server gives to the adapter. In other words, when doing transactional sends you must split the resubmit, MoveToNextTransport, and MoveToSuspendQ actions into separate batches.

Many adapters will sort a batch of messages it has been given for multiple endpoints into separate list of messages for further processing.

The point is that there are no rules (besides those associated with the transaction) that are associated with message batching in BizTalk Server. Batching is simply an implementation-specific way to chunk messages into and out of BizTalk Server.

An adapter can batch up messages from multiple smaller batches that BizTalk Server has given the adapter into a larger batch for the response to BizTalk Server. This might be a significant optimization in transactional adapters that are dealing with large numbers of very small messages.

Typically BizTalk Server 2006 produces send-side batches of between 5 and 10 messages. If these were very small messages, an adapter might choose to batch up to 100 messages or more before it submitted a transactional batch of deletes back to BizTalk Server.

An optimization like this would not be easy to implement; you must make sure that messages never got stuck in the adapter memory, waiting endlessly for some threshold to be met.

The significance of this batching can be seen in the performance numbers for BizTalk Server for the high throughput adapters like MQSeries. In these adapters, messages are received at least twice as often as they are sent. By default the receive uses batches of 100 messages and the send just uses the BizTalk Server batch it was given.

Use a separate batch for response messages

When using transactions on the send side, the transaction created by BizTalk Server is used against the target system and then used for the delete back on BizTalk Server. If anything fails, the transaction can be ended in which case the delete is ended, and the data remains in BizTalk Server and not in the target system. But the adapter does not just end the transaction; it must also handle the state of the messages it was given correctly. Specifically, the adapter should call resubmit , MoveToNextTransport, and MoveToSuspendQ appropriately.

This is why, to make use of transactions on the send side, you must use the asynchronous batching API. Transactions are only supported for asynchronous send adapters. You should not try to use transactions with synchronous send adapters

It is very important to place the delete operations and SubmitResponse operations in a batch together that uses the transaction. Failure is handled by ending the transaction (to ensure that data is only submitted once to a external system), but you still want to resubmit or run MoveToNextTransport for the message back on BizTalk Server. The trick is to use a separate normal (non-transactional) batch for these types of operations.

The following figure shows the use of separate batches for response messages.

Transactional adapters must call DTCommitConfirm()

When you write an adapter that creates a transaction object and hands it to BizTalk Server, you are accepting responsibility for writing code that does several things:

- Decides the final outcome of the batch operation: to either commit or end the transaction.

- Informs BizTalk Server of the final outcome by calling the method, DTCConfirmCommit()

Using DTCConfirmCommit()

The adapter must inform BizTalk Server about the final outcome of the transaction in order to maintain its internal tracking data. The adapter informs BizTalk Server of the outcome by calling DTCConfirmCommit. If the adapter does not do this, a significant memory leak occurs.

A possible race condition

The two tasks listed above (resolve errors and decide the final outcome) seem simple enough, but in fact they rely on information from different threads:

- The adapter processes errors based on information passed by BizTalk Server to the BatchComplete callback in the adapter. This callback is on the adapter's thread.

- DTCConfirmCommit is a method on the IBTDTCCommitConfirm object. An instance of the IBTDTCCommitConfirm object is returned by the batch IBTTransportBatch::Done() call. This instance is on the same thread as the IBTTransportBatch::Done() call, which is different from the adapter's thread.

- For every call that adapter makes to IBTTransportBatch::Done() there is a corresponding callback BatchComplete() that is called by the messaging engine in a separate thread to report the result of the batch submission. In BatchComplete() the adapter needs to commit or roll back the transaction based on whether the batch passed or failed. In either case, the adapter should then call DTCConfirmCommit() to report the status of the transaction.

A possible race exists because the adapter’s implementation of BatchComplete can assume that the IBTDTCCommitConfirm object returned by IBTTransportBatch::Done() is always available when BatchComplete executes. However, BatchComplete() can be called in a separate messaging engine thread, even before IBTTransportBatch::Done() returns. It is possible that when the adapter tries to access IBTDTCCommitConfirm object as a part of the BatchComplete implementation, there is an access violation. Use the following solution to avoid this condition.

Use events to avoid a race condition

The problem is solved with an event in the following example. Here the interface pointer is accessed through a property that uses the event. The get always waits for the set.

Now assign the return value from IBTTransportBatch::Done() to this property and use it in the BatchComplete call.

Sort the send-side transactional batches by endpoint

Batches of messages sent by BizTalk Server to the adapter can span multiple send ports (or endpoints). Because the adapter typically only wants to have a transaction to a single endpoint, the adapter must sort the messages based on send port (SPName or OutboundTransportLocation). This way, the adapter can create a transaction that only spans a particular send port.

For example, when an FTP send adapter receives a batch of messages from BizTalk Server, it gets a mixed batch of messages for all the currently active FTP send ports. This happens because the API is singleton based, meaning that there is only a single FTP adapter loaded, not one per send port.

This leaves the adapter with some work to do. The adapter must first sort the batch of messages it was given by BizTalk Server into separate batches, one for each endpoint. After it has done that it can deal with each endpoint in turn and will probably construct delete batches for each endpoint. The BaseAdapter generic reusable classes in the SDK sample code works in the same way.

Dynamic send

A BizTalk Server orchestration can send a message to a port that has not been configured as long as it provides sufficient configuration details in the message header and in the URL itself. The protocol of the URL must be recognized by BizTalk Server.

Sorting for dynamic send

When sorting messages, you should take care to establish what defines an endpoint. This is especially true in the case of *dynamic send*. If it’s just the URI that defines the endpoint, then things are quite simple. However, in an FTP session the username logon details might be used by the FTP server to define the true endpoint. In this case, if the adapter logs in as a different account, it may be connected to a different directory.

In some cases, the true endpoint is not known until you have run the Enterprise Single Sign-On (SSO) command ValidateAndRedeemTicket.

In the case of MQSeries, the determination of whether to use transactions was made configurable. Given the architecture and the use of a remote COM+ object, it turned out that it was best to regard a transactional endpoint as distinct from a non-transactional endpoint.

To summarize, the sorting of messages into their single endpoint batches is sometimes a non-trivial task and may involve such extra steps as considering the context values and even the result of a call to SSO.

Sorting for static send

If the endpoint is a configured endpoint rather than a dynamic send, then there is a unique GUID on the message context called the static port ID (SPID) that can be used for sorting the endpoint. The following code can be used to retrieve it:

This should come as something of a relief when you consider the problems introduced by the XML Schema Definition (XSD)-based configuration framework. With this you have a property that might be part of the endpoint key buried inside XML in a single property on the context. This property might well co-exist as a context property in itself. You should understand that if you have a SPID on the context, you can use that. Otherwise you are doing a dynamic send and you need to construct an alternative key to sort the batch by.

The following figure shows message sorting by endpoint.

You should keep in mind that the retry count of a message is not aware of the success or failure of a batch.

On the send side, a batch of messages may fail because a few messages in the batch have failed. The adapter must make a determination for every message that it receives and in the failed batch scenario, you might assume that every message is resubmitted. However, if all the messages in a failing batch are resubmitted, the retry count (which is maintained by the BizTalk Server engine) is incorrectly incremented even for the nonfailing messages because they happen to be in the same batch as the bad messages. In this case, an adapter could reform the outbound batch and retry the good messages against the external system.

Use the right ID for a message

BizTalk Server provides a number of different IDs for a message and you should be careful to use the correct ID at the correct time.

- A MessageID is specific for an instance of a message in memory, which means that a retry of a message comes with a different MessageID.
- A WorkID stays constant across message retries.
- An InterchangeID also stays constant across retries but could be shared with other messages in the system. If you need an ID which will stay constant across message retries, do **not** use the Interchange ID, because it is common to every message that results from a disassembly in the receive pipeline. This ID is used if a target system depends upon duplicate ID detection to enforce once-only delivery. The Microsoft® BizTalk® Adapter v2.0 for mySAP™ Business Suite makes use of the WorkID for this purpose.

Handling Large Messages

The BizTalk engine has been engineered in a manner such that it is capable of processing very large messages, officially the supported message size is 2GB, however internally the engine can handle significantly larger files in some scenarios. Under most scenarios the engine will maintain a flat memory model regardless of the size of the messages being processed. There are some caveats to this around large message interchanges, the Performance Characteristics white paper details this.

Stream Based Processing

It is important to develop adapters with large message handling in mind, loading the entire data stream in to memory regardless of it's size is strongly discouraged as this could potentially bring the BizTalk Server process down depending on the size of the message and the number of messages that the engine is processing at any given time. Instead, messages should be processed in a streaming fashion, for inbound messages this typically means that the network stream is attached to the BizTalk message leaving the 'pulling' of the stream to the BizTalk Messaging Engine.

Similarly, on the outbound side, the adapter is responsible for pulling the stream, this effectively pulls the stream from the Message Box and though the transmit pipeline, the adapter should send the data over the wire in a streaming fashion.

The diagram below illustrates the stream based processing on the receive side of the Messaging Engine. When an adapter submits a message to the engine it should attach its data stream to the BizTalk message, for some adapters this may mean implementing a network stream. When the message is submitted, the engine will execute the receive pipeline, during the pipeline execution the pipeline components that wish to change the data will clone the message, wiring up the stream from the new message to the stream on the previous message. After the pipeline has been executed the Messaging Engine will get a message out of the pipeline, the engine will then execute a loop reading the stream on that message.

This reading of the stream will in turn invoke a read on the previous stream which will in turn invoke a read on the previous stream and so on back to the network stream. The engine will periodically flush the data to the Message Box in order to maintain a flat memory model.

Trouble Shooting Tip: On the send side, since the adapter is responsible for reading the stream, it should be noted that if the transmit adapter wishes to read any message context properties that are promoted or written in the transmit pipeline, these properties may not be written until the stream is read in it entirety since this is the point at which the adapter can be sure that all of the pipeline components have finished executing.

Stream Seekability

In many scenarios the data stream is not seek-able, for example consider an HTTP adapter receiving data using chunked encoding, for the data stream to be made seek-able the adapter would need to cache the data as it is read, either in memory or onto disc, clearly this is not optimal and requires additional resources. Further, many of the out of the box pipeline components operate in a forward only streaming fashion.

There are scenarios when an adapter may need to seek the stream back to the beginning in order to handled failed messages that need to be suspended. For example, an HTTP adapter that is receiving data using chunked encoding in order to submit the response message in a solicit-response pair. For these scenarios the BaseAdapter in the SDK has a useful helper class, called a virtual stream. The idea behind a virtual stream is that the data in the stream is cached in a memory stream until it reaches a threshold, over which the data will be overflowed to a secure location on disc. One the stream is closed the disc file is automatically deleted. This approach allows forward only streams to be made seek-able.

Handling Transactions

Adapters may provide a DTC transaction to ensure atomicity between the adapter and the Message Box, this only makes sense if the adapter is reading from and writing to a data source that can be enlisted in a DTC transaction, for example an adapter that writes to SQL Server.

It should be noted that the adapter always owns the transaction, transacted adapters always own the root transaction, and the adapter will always commit or abort the transaction, while the engine will simply vote.

The engine needs to be informed of the outcome of transactions that are supplied by adapters to the engine. When an adapter passes a DTC transaction to the engine, the engine will return an IDTCCommitConfirm interface to the adapter, the adapter needs to call IDTCCommitConfirm.DTCCommitConfirm() passing in the transaction and a boolean to indicate whether the transaction was ultimately committed or aborted.

Transacted Receivers

The main different between transacted receivers from non-transacted receivers is that they use a DTC transaction to ensure atomicity between their data source and the BizTalk Message Box, in general every other aspect of the adapter will be the same.

It should be noted that a request-response receive adapter will only use a transaction for the submission of the request message a different transaction will need to be used for the transmission of the response. This is because the scope of the transaction is in essence between the adapter and the Message Box the request message will not be published until the transaction for the request message is committed.

The object interaction diagram below illustrates the interaction between the adapter and the Messaging Engine. In this example, the following sequence of interactions take place:

1. The adapter gets a new batch from the engine
2. The adapter creates a new DTC transaction
3. The adapter does a destructive read from it's data source that has been enlisted in the transaction
4. The adapter submits the message
5. The adapter calls Done on the batch passing in it's DTC transactions, the engine returns an IBTDTCCommitConfirm interface
6. The engine processes the batch and calls the adapter back on it's BatchComplete implementation
7. Since the batch was successful the adapter commits the transactions
8. Since the commit of the transaction was successful the adapter calls the IBTDTCCommitConfirm.DTCCommitConfirm() API

Transacted Transmitters

Again transacted adapters are for the most part very similar to non-transacted adapters, the main difference being that the adapter will send the data in the message to a resource that it has enlisted in a DTC transaction.

Implementation Tip: For transacted sends, the adapter should use the same DTC transaction for the IBTTransportBatch.DeleteMessage() API and for writing the data to the destination, only these two operations need to be transacted. Any other operations such as IBTTransportBatch.Resubmit(), IBTTransportBatch.MoveToNextTransport(), and IBTTransportBatch.MoveToSuspendQ() do not need to be transacted, this is because the engine will use a transaction under the covers and these types of operation do not need to be atomic with respect to the destination.

The following object interaction diagram illustrates the interactions between the adapter and the engine. The sequence of events is as follows:

1. The engine gets a new batch from the adapter
2. The engine starts the new batch
3. The engine delivers two messages to the batch
4. The engine calls Done() on the batch, causing the adapter to post the batch to it's internal transmit queue that is serviced by it's thread pool
5. The adapter creates a new DTC transaction
6. The adapter transmits the messages enlisting the destination in the DTC transaction
7. The adapter gets a new batch from the engine
8. The adapter calls DeleteMessage() for the messages that it has successfully transmitted
9. The adapter calls Done() on the batch passing in it's DTC transaction, the engine returns an IBTDTCCommitConfirm interface
10. The engine processes the batch, deleting the messages from the application queue
11. The engine calls back the adapter
12. The adapter commits the transactions since the batch was successful
13. The adapter calls IBTDTCCommitConfirm.DTCCommitConfirm() to inform the transaction was successfully committed

Transacted Solicit-Response Adapters

Unlike two-way receives, two-sends maybe performed using the same DTC transaction. Transacted solicit-response adapters should use the same IBTTransportBatch for the SubmitResponseMessage() and DeleteMessage() operations, this batch should use the same DTC transaction that is used to send and receive the solicit-response message pair, this will ensure atomicity for the solicit-response message exchange.

Service Components and BYOT

As we have seen the engine API's require a DTC transaction to be supplied, however some.Net components are designed to be used as serviced components and do not allow the transaction to be programmatically committed or aborted, but instead it is automatically committed by the platform.

For these scenarios, the adapter should use Bring Your Own Transaction (BYOT), this allows the adapter to create a DTC transaction, create the .Net component that uses the transaction and allow that component to inherit the created transaction rather than create it's own transaction. The .Net framework provides System.EnterpriseServices.BYOT for this purpose, however there is a bug than means that BYOT will AV on W2K (up to SP3) if used outside of a COM+ application. The SDK BaseAdapter provides a helper class for this purpose, BYOTTransaction.

Avoiding Race Conditions

When you write an adapter that creates a transaction object and hands it to BizTalk Server, you are accepting responsibility for writing code that does several things:

- Decides the final outcome of the batch operation: to either commit or end the transaction.

- Informs BizTalk Server of the final outcome by calling the method, DTCConfirmCommit()

The adapter must inform BizTalk Server about the final outcome of the transaction in order to maintain its internal tracking data. The adapter informs BizTalk Server of the outcome by calling DTCConfirmCommit. If the adapter does not do this, a significant memory leak occurs.

The two tasks listed above (resolve errors and decide the final outcome) seem simple enough, but in fact they rely on information from different threads:

- The adapter processes errors based on information passed by BizTalk Server to the BatchComplete callback in the adapter. This callback is on the adapter's thread.

- DTCConfirmCommit is a method on the IBTDTCCommitConfirm object. An instance of the IBTDTCCommitConfirm object is returned by the batch IBTTransportBatch::Done() call. This instance is on the same thread as the IBTTransportBatch::Done() call, which is different from the adapter's thread.

  For every call that adapter makes to IBTTransportBatch::Done() there is a corresponding callback BatchComplete() that is called by the messaging engine in a separate thread to report the result of the batch submission. In BatchComplete() the adapter needs to commit or roll back the transaction based on whether the batch passed or failed. In either case, the adapter should then call

DTCConfirmCommit() to report the status of the transaction to the messaging engine.

A possible race exists because the adapter's implementation of BatchComplete can assume that the IBTDTCCommitConfirm object returned by IBTTransportBatch::Done() is always available when BatchComplete executes. However, BatchComplete() can be called in a separate messaging engine thread, even before IBTTransportBatch::Done() returns. It is possible that when the adapter tries to access IBTDTCCommitConfirm object as a part of the BatchComplete implementation, there is an access violation.

The problem is solved with an event in the following example. Here the interface pointer is accessed through a property that uses the event. The get always waits for the set.

```
protected IBTDTCCommitConfirm CommitConfirm

{

set

{

this.commitConfirm = value;

this.commitConfirmEvent.Set();

}

get

{

this.commitConfirmEvent.WaitOne();

return this.commitConfirm;

}

}

protected IBTDTCCommitConfirm commitConfirm = null;

private ManualResetEvent commitConfirmEvent = new
ManualResetEvent(false);
```

Now assign the return value from IBTTransportBatch::Done() to this property and use it in the BatchComplete call.

Handling failure

In general adapters should suspend messages that they cannot process, for example, receive adapters that experience submit failures should typically suspend these messages, though this does depend on the adapter. However there are security considerations around this, sending bad data to an endpoint whereby the adapter blindly suspends messages that fail submit could render the adapter open to a denial of service attack from a malicious user by filling up the BizTalk suspend queue. Further adapters such as HTTP for example are capable of returning a failure code to the client, indicating that the request has been rejected, for these types of adapters often is does not make sense to suspend the message, but rather return a failure code. Typically transmit adapters will only suspend messages after all of the retries have been exhausted for both primary and secondary transports.

Error processing should be associated with an individual operation and not with the batch that contains the operation

Performance issues aside, the batching of messages should be invisible to the user of the adapter. This means that the failure of one operation in a batch should not affect any other operation in any way. However, batches are atomic, so the failure of one message results in an error for the batch, and no operations are processed.

Your write the code that is responsible for handling the error, re-submitting the successful messages, and suspending the unsuccessful ones. Fortunately, BizTalk Server provides a detailed error structure that enables your adapter to determine the specific operation that failed, and to construct further batches in which the successful operations are resubmitted and the unsuccessful ones suspended.

The final durable state of the operation should not be affected by the batching that happened to go on within the adapter.

Use SetErrorInfo to report failure to BizTalk Server

If you are suspending a message, you must provide failure information to BizTalk Server from the previous message context. BizTalk Server provides error reporting capabilities using the SetErrorInfo method on both the IBaseMessage interface as well as on ITransportProxy:

- When a failure occurs while processing a message, set the exception using SetErrorInfo(exception) on the message to be suspended. This way, the engine preserves this error with the message for later diagnosis, and logs it to the event log to alert the administrator.

- If you encounter an error during initialization or internal bookkeeping (not during message processing) you should call SetErrorInfo(exception) on the ITransportProxy pointer passed to you during initialization. If your adapter is based on a BaseAdapter implementation, you should always have access to this pointer. Otherwise, you should be certain that you cache it.

Reporting an error with either of these methods results in the error getting to the event log. It is important that you associate the error with the related message if you are able to do so.

Database Off-Line

The BizTalk service will fail fast in the event of one of the BizTalk databases going off-line, this will cause the service to recycle under these circumstances. The Messaging Engine will make a best effort to shutdown all of the receive locations before recycling the service, if this takes longer than 60 seconds then the service will terminate, since the engine is transacted this will not cause data loss.

This time out can be tuned in the registry using the

For isolated adapters, since BizTalk does not own the process, the receive locations will be disabled when one of the BizTalk databases go off-line, once the database comes back on-line those receive locations will be re-enabled.

Writing to the Event Log / Tracing

Event log entries may be written by the adapter by using the IBTTransportProxy interface passing in an exception, adapters developed in native code will need to pass in an IErrorInfo interface), IBTTransportProxy.SetErrorInfo(Exception e).

The Messaging Engine will write to the event log on behalf of the adapter for events such as when an adapter reties a message after transmission failure, when the adapter moves a message to its backup transport or suspends a message. For operations such as these all the adapter needs to do is set the exception on the message prior to calling the API, the follow code snippet demonstrates this:

Handling Receive-Specific Batch Errors

Handling receive failures

BizTalk Server does not handle all failures; the adapter must be configured to handle errors like "no subscription."

When an adapter submits an operation (or batch of operations) to BizTalk Server there can be various reasons for failure. The two most significant are:

- The receive pipeline failed.
- A routing failure occurred while publishing a message.

The messaging engine automatically tries to suspend the message when it gets a receive pipeline failure. The suspend operation may not always be successful due to the following issues.

If the messaging engine hits a routing failure while publishing a message, then the messaging engine will not even try to suspend the message.

It is always possible that a message will fail. In such a situation, the adapter should explicitly call the MoveToSuspendQ() API and should try to suspend the message. When an adapter tries to suspend a message, one of the following should be true:

- The same message object that it had submitted (recommended) should be suspended.

- If the adapter has to create a new message, then it should set the message context of the new message with the pointer to the message context of the message that was originally submitted. The reason for this is that the message context of a message has a lot of valuable information about the message and the failure. Tthis information is required to debug the failed message.

Some adapters, such as the HTTP adapter provided with BizTalk Server, do not require that the message be suspended. These adapters can just return an error back to their client.

Causes of failure

A simple cause of failure are the errors that can occur as the batch is constructed or when IBTTransportBatch::Done() is called.

- **Submit failure**: The Submit call can fail for a limited number of reasons, and all of them are fatal. These reasons include:

- Out-of-memory.

- The schema assembly has been dropped from the deployment. In this case, the Submit fails with a cryptic error. In the MQSeries adapter, the generic failure exception from BizTalk Server is caught, and an extended error message is written in the system event log. This message suggests that one of the possible causes of the error is that the schema assembly has somehow been dropped from the deployment.

  - In general, if submit fails you should try to suspend the message using the same transaction.

- **IBTTransportBatch::Done() failure:** The IBTTransportBatch::Done() call can fail for one of several reasons. In general, you should always attempt one suspend operation and only abort if that fails. One of the error codes you might receive from the failure of IBTTransportBatch::Done() is that BizTalk Server is trying to shut down. In this case, you should just end the transaction and leave it because the Terminate call is probably happening concurrently. Other scenarios occur when you have successfully constructed the batch and successfully executed IBTTransportBatch::Done(). In these cases, the errors are returned in

BatchComplete and the adapter must work out what to do with them. The rest of the topic "Handling receive specific batch errors" deals with this case.

Processing BatchComplete errors

BatchCompelete() is a callback provided by the adapter that is invoked by BizTalk Server to indicate the completion status of a batch operation.

The most important parameter passed to BatchComplete is the batch status hResult. This indicates success or failure for the batch. If the batch failed, it means that none of the operations in the batch have succeeded. (See the topic, "Batches are Atomic.") The adapter goes through the batch status structure and determines which messages failed (this is known as *filtering the batch*).

Non-transactional BatchComplete errors

For non-transactional adapters, you must choose your response if a failure occurs for a SubmitMessage()/ SubmitRequestMessage() or SubmitResponseMessage() operation. Typically adapters suspend the message by calling MoveToSuspendQ().

The following operations are always expected to pass: DeleteMessage(), MoveToSuspendq(), ResubmitMessage(). If these operations fail, it typically means that there is a bug in the adapter. You don't have to write code to handle a failure in these cases. However if the batch failed because another operation failed, then these operations must be re-executed in a fresh batch.

If the adapter had called MovetoBackupTransport() and if that fails (because there was no backup transport), then the adapter should call MoveToSuspendq() to suspend the message

Transactional BatchComplete errors

When you submit batches to BizTalk Server using a transaction created by the adapter, you should follow one of these two scenarios:

- **Use Single message batches:** Send a single-message batch to BizTalk Server. If that single message fails, then you can legally send BizTalk Server a second batch under this same transaction, this time making sure to move the offending message to the Suspend Queue rather than submitting it. This second batch should succeed and you can then commit the transaction, of course waiting for BizTalk Server to confirm that that second batch was successful. If it wasn't, the adapter must end the transaction, or find somewhere else to place that message. In this scenario, you immediately take a significant performance hit. (Particularly so because you are using transactions.)

  There are some techniques that can still be used rescue the performance of the adapter. For example, the MQSeries adapter adapts its approach dynamically at runtime. It runs with 100 message batches. If it hits an error, it must end the batch, but it switches to single message batches for a short while as it eats past the bad message. It then lets the throttle out and reverts to 100 message batches. Of course, if it hits the error again, it again slows down.

Unfortunately with this adapter, nothing could be done to stop the pipeline from seeing a failed message as a submit operation; an unavoidable consequence of this approach. Pipeline components (such as BAM pipeline reporting) see the message on the submit even though the message might actually be suspended.

- **Use preemptive suspension:** Construct a multi-message batch in which the erroneous messages are preemptively suspended. The batch contains a mix of Submits and MoveToSuspendQ, and it will be the first and only batch under the transaction. It should succeed, given that the bad data was preemptively suspended, and the transaction can be committed (after waiting to receive the confirmation from BizTalk Server.)

  This would seem to require looking into the future, but this technique has actually been used in the MSMQ adapter. The trick is having reliable, unique message IDs. This adapter constructs a batch of messages; if anything fails it rolls back the transaction (and so the batch), but remembers the message ID in a temporary data structure. (In order to protect this structure from growing indefinitely, items in it are removed after some fixed time delay.) Before each batch is submitted, the adapter checks the list of bad message IDs. If it sees one, it knows that that message will fail (because it failed once in the past) and preemptively suspends it rather than trying to submit it.

  Not every adapter has a reliably unique message ID. (And if it's a transactional store, it is less likely to have one). Because of this, many transactional adapters are restricted to sending single-message batches.

Processing other errors

In all other cases (such as failures in suspending messages), the adapter must end the transaction. Any other outcome results in either duplicate or dropped messages.

Whenever the adapter can, it should abort the transaction if a batch fails. However there are scenarios, where the adapter cannot abort the transaction. In such a scenario it should suspend the message using the same transaction.

Process errors on transactional receive

A common transactional processing pattern is to end a transaction when an error occurs: everything returns to its previous state and no data is lost. However, if you are consuming data from some kind of transactional feed, for example, pulling a row at a time from some staging table on a database, or pulling one message at a time from a queuing product like MQSeries or MSMQ, then this might not be enough. If you simply end the transaction and go back and pick up the same data again, the same error is likely to occur and the system becomes stuck in a repeated loop.

The SQL adapter in BizTalk Server 2006 actually shipped with this behavior. The problem was that the adapter would see an error from BizTalk Server and end the transaction. However, soon after release the adapter was changed to attempt to suspend a failed message and commit the transaction. Moving a message to the suspend queue under the same transaction and then committing the transaction saves the data from ever being lost and also allows the adapter to eat past bad data.

When the receive portion of an adapter is passed an error message in response to a Submit message operation, the adapter should process that error and move the message to the Suspend Queue.

In the case of transactional batches in which the adapter has created the transaction object and submits messages under the transaction, the adapter should logically move the message to the Suspend Queue under the same transaction when failures occur. It is the transaction that ensures that data is not dropped. (And even data that is causing an error should never be dropped.)

BizTalk Server does not suspend a message if there is a routing failure

BizTalk Server does not accept a message to be published in its MessageBox if there are no subscriptions defined to accept it. Subscriptions are either orchestrations or send ports. Multiple subscriptions can be defined, in which case the message is sent to multiple destinations. If there are no subscriptions BizTalk Server rejects the message outright, and it will not attempt to suspend it. If the adapter does not handle this error and explicitly suspends the message, then the message is dropped and its data potentially lost. Of course a transactional adapter may end the transaction and return the message to its destination.

Your receive stream should support Seek ()

The receive side stream must support the **Seek()** method for BizTalk Server to be able to suspend the message on a pipeline failure. If the message stream is not seekable, then BizTalk Server will still try to run Seek(), and will generate an error.

In many cases supporting Seek is not easy. When streaming data from a network for example, it may be difficult to go back to the network resource and request the data again.

The trick that a number of adapters that shipped with BizTalk Server 2006 employed is to spool the message data onto disk at the same time as BizTalk Server reads the data. If BizTalk Server hits an error (in the pipeline processing of the message data for example), then it can use Seek() on the stream and the data will be read from the disk. Internally it uses the ReadOnlySeekableStream class that wraps an incoming non-seekable stream and overflows to disk when a configurable threshold is reached. For messages smaller than the threshold size, the disk is never hit, for large messages memory bloat is avoided.

Consider user-configurable options to customize adapter error handling

Sometimes there is no one correct response to an error. In this case, you should consider a user-configurable option to choose between behaviors. The MQSeries adapter does this.

The problem with having the adapter suspend messages when it sees an error is that the suspend queue in BizTalk Server 2006 is something of a “black hole." It is relatively easy to get messages in the queue, but a little tricky to get them out again.

Some users of the adapter might not want anything in the suspend queue. For example, in the case of the MQSeries adapter, the user is offered a configuration option to do one of the following:

- Set the adapter to end the current transaction and disable itself when it sees an error.
- Suspend the failed message and commit the transaction. The adapter does this even when BizTalk Server has successfully suspended the message. This action meets the requirements of the customer even if it causes the event log to not be strictly correct.

Implement receive ordering by using a single thread and waiting on BatchComplete

The interface to BizTalk Server is designed for performance and the ability to scale out by supporting concurrency. However, if you want a strictly ordered receive of messages (as is sometimes required when receiving messages from a message queue product like MQSeries or MSMQ) then you must do some additional work in the adapter to disable some of that concurrency. This can be done in two steps:

1. You must use a single thread for all the data processing in the adapter.
2. You must wait for BizTalk Server to have completely processed each batch. This requirement is important and can be accomplished by using .NET thread synchronization primitives. For example, using an AutoResetEvent, you would:
    - Declare the event object where it can be accessed by both the main worker thread and the BatchComplete callback object.
    - On the main worker thread submit the messages to the batch as usual but then call AutoResetEvent.Reset() on the event object just before the call to the batch IBTTransportBatch::Done().
    - Call AutoResetEvent.WaitOne() on the event object from this same thread. This would cause the main worker thread to block. In the BatchComplete callback from BizTalk Server you would then call

AutoResetEvent.Set() on the same event object to unblock the worker thread so it is ready to process another message.

It is strongly suggested that *receive ordering* like this be made configurable because it causes significant performance degradation. Many, if not most, user scenarios don't actually require ordering of messages. Suspending messages can also break ordering. Exactly what to do in this case is really application-dependent, so the best thing for your adapter to do is to offer the user a configuration point.

In ordered scenarios, some customers have stated that they would rather stop the processing, that is, disable the adapter, rather than break ordering. The MQSeries adapter, which supports ordered receive provides this option to the user.

⊟Handling Send-Specific Batch Errors

Handle send retry and batching

Here is a typical example of send-side batching:

- BizTalk Server gives a batch of messages to the adapter.
- When the adapter has determined that it has given the message to its destination correctly, it executes delete back on BizTalk Server indicating that it is done. (As usual, several delete messages can be arbitrarily batched up to improve performance.)

If the send-side adapter fails to process a message then it may do one of several things with that message:

- The adapter should tell BizTalk Server that it wants a message retried. BizTalk Server does not automatically retry a message. BizTalk Server keeps a count of the retries and this count can be seen in the message context.
- An adapter may determine that it can't process a message. In this case, the adapter might move it to the next transport. The adapter does this with the MoveToNextTransport() call on the Batch object.
- The adapter is also free to move the message to the Suspend Queue.

The adapter determines what happens to the message. However, it is recommended that you have adapters behave in a consistent manner because this makes a BizTalk Server installation easier to support.

It is highly recommended that adapters behave in the following way, as the adapters that are shipped with BizTalk Server do.

Recommended behavior for handling send errors in a batch:

The send adapter receives some messages and it submits them to BizTalk Server.

For each successful message the adapter should delete that message on BizTalk Server. All communication back to BizTalk Server is done through batches and the deletes can be batched up. They do not have to be the same batch that BizTalk Server created on the adapter. If there are any response messages (as in a SolicitResponse scenario), then they should be submitted back to BizTalk Server (with SubmitResponse) along with the associated delete.

- If the message processing in the adapter was unsuccessful, check the retry count.
  - If the retry count was not exceeded, resubmit the message to BizTalk Server remembering to set the retry time on the message. The message context provides the retry count and the retry interval the adapter should use.
  - If the retry count has been exceeded, then the adapter should attempt to move the message using MoveToNextTransport.The resubmit and MoveToNextTransport messages can be mixed altogether along with the deletes in the same batch back to BizTalk Server. This is not required, but can be a useful step.
- The resubmit and the MoveToNextTransport are ways for the adapter to deal with failures. But there can be a failure within the processing of the failure. In this case, in processing the response from BizTalk Server (this is done in the BatchComplete method) the adapter must create another batch against BizTalk Server to indicate what to do with that failure.

  Follow these steps when processing a failure that occurs within the processing of another failure:

  - If the resubmit fails, use MoveToNextTransport.
  - If the MoveToNextTransport fails, use MoveToSuspendQ.

Ultimately you must keep creating batches on BizTalk Server until you receive a successful action back on BizTalk Server.

Terminating the Adapter

The following topics provide guidance on the proper shutdown of an adapter.

Adapter Termination

When the Messaging Engine is shutting down it will call **IBTTransportControl**.**Terminate**() on each adapter. Once this method returns the adapter will be destroyed, for managed adapters this is less deterministic due to the .Net garbage collection, regardless the adapter should block in **Terminate**() until it is ready to be destroyed. Any cleanup work should be done in **Terminate**() the adapter should block until this work has been completed.

Terminating Isolated Receive Adapters

Isolated receive adapters will not have **Terminate**() called on them since they are not hosted in the BizTalk service. Instead, they should call **IBTTransportProxy**.**TerminateIsolatedReceiver**() to notify the Messaging Engine that they are about to shutdown.

Clean up COM objects using Marshal.ReleaseComObject()

Use **Marshal**.**ReleaseComObject**() to clean up COM objects, particularly **IBTTransportBatch**.

When writing managed code that uses COM objects, the Common Language Runtime (CLR) runtime generates proxy objects that hold the actual references to the COM object. The proxy objects are managed objects and subjected to the usual rules of garbage collection. A problem arises in that the garbage collector only sees memory that it allocated, and has no awareness of the COM object itself. With the proxy objects being small a situation could exist where a large COM object is left in memory simply because the CLR garbage collector is oblivious to it.

To avoid this problem, explicitly release the underlying COM object if you are certain you are done with it. This is done by calling **Marshal**.**ReleaseComObject**().

In the case of the adapters that shipped with BizTalk Server, the programmers found that explicitly releasing the **IBTTransportBatch** object returned from **TransportProxy GetBatch** can make a significant improvement to performance.

Be sure to use Terminate() when closing the adapter

In order for BizTalk Server to recognize your code as an adapter, you must implement an interface called **IBTTransportControl**. This interface defines how BizTalk Server communicates with your adapter, and is defined as follows:

The interface contains two methods, **Initialize** () and **Terminate** ():

Initialize()

BizTalk Server calls the **Initialize** method after it loads the adapter assembly. It does this to pass the Transport Proxy (the main handle to BizTalk Server) to the adapter. The implementation of Initialize is straightforward; simply store the Transport Proxy in a member variable.

Terminate()

Terminate is called by BizTalk Server on shutdown, to give the adapter time to finish the execution of all batches. This makes the implementation of the **Terminate()** method much more involved.

The adapter should not return from a **Terminate()** call until any pending work it has is complete. When BizTalk Server calls Terminate, the adapter should try to stop all its current tasks and not start any new ones.

Because the **Terminate** call is called as part of the service shutdown, the service control manager ends the process if the adapter perpetually blocks in **Terminate**. In this case, you see the warning from the service control manager as it kills the BizTalk Server service. This should, of course, be avoided. If the adapter does not handle this appropriately, and still has threads running when the process starts to shut down, then you will occasionally see an access violation from BizTalk Server on shutdown.

Because of the asynchronous nature of the interface to BizTalk Server, it is likely that under load there will be many batches and therefore threads still being executed. The **Terminate** call should be implemented to wait on the conclusion of every batch the adapter has successfully executed on BizTalk Server. The conclusion of the batch is signaled by the **BatchComplete** callback from BizTalk Server. So the **Terminate** call should wait on every pending **BatchComplete** to happen. However, the execution of the batch must be successful; that is, the call to **IBTTransportBatch**::**Done**() must not have failed. If the call to **IBTTransportBatch**::**Done**() fails, there is no batch Callback.

After you realize that you have to add synchronization code to your adapter, the implementation is fairly straightforward.

One simple approach is to implement a compound synchronization object with **enter()** and **leave()** methods for the worker threads and a **terminate()** method that blocks while a thread is still within the protected execution. (Incidentally, the solution is very similar to the familiar multiple reader, single writer structure where the worker threads can be thought of as readers and the terminate method as the writer.)

The **terminate** method is as follows:

And for each worker thread:

And finally in the callback from BizTalk Server:

BizTalk Server 2006 shipped with sample code, ControlledTermination, for the synchronization object described here.

Security and SSO

Enterprise Single Sign-On (SSO) provides services to store and transmit encrypted user credentials across local and network boundaries, including domain boundaries. Transport adapter writers can leverage SSO API's to handle the user credentials that transport adapter uses to access back-end applications.

Usually, transport adapters that do not support SSO are required to be configured with a single set of credentials that they will use for accessing backend applications. However for many backend systems single account authentication may not satisfy all its security enforcements. Many applications may provide different access rights depending on the user who is accessing them. SSO allows adapters to dynamically choose the credentials to use for the endpoint based on the user who is trying to access it.

SSO Support for Receive Adapters

Receive adapter that supports SSO performs the following steps after receiving a message and before publishing it to the server:

Adapter impersonates the sender and obtains the SSO ticket on behalf of sender by using ISSOTicket.IssueTicket API.

After successfully obtaining SSO ticket adapter stores it on the message context property "SSOTicket" under system namespace (http://schemas.microsoft.com/BizTalk/2003/system-properties)

The following code snippet demonstrates how the ticket is obtained and how it is stored on the message context.

SSO Support for Send Adapters

Send adapter that supports SSO validates and redeems the ticket and obtains the user credentials from the SSO system by using ISSOTicket.ValidateAndRedeemTicket API. The obtained credentials are used by adapter to authenticate with the destination endpoint.

The following code snippet demonstrates how send adapter obtains the user credentials from the SSO system:

Party Resolution

The Party Resolution pipeline component is responsible for mapping the sender certificate or the sender security identifier (SID) to the corresponding configured BizTalk Server party. Adapters that have this information available to them should set the two system message context properties, WindowsUser and SignatureCertificate to be consumed by the down stream party resolution component if configured.

The WindowsUser property is populated with the domain user of the sender, for example redmond\myBtsUser. The SignatureCertificate is populated with the thumbprint of the client authentication certificate.

Be careful managing passwords

If you put credentials directly in the properties of an endpoint, the password field will be blanked out when you need to export a binding file. This will require your user to re-key in the password as an administrator. You can avoid this difficulty by using SSO for credentials.

If the adapter endpoint has a Password property, be aware that the actual value is stored in clear text in the SSO Configure Store database despite the fact that it is displayed in the UI as "*". This property is also transferred through the network and a simple script using the BizTalk Server sample ExplorerOM will be able to read it.

Performance tuning

Where possible adapters should be batch aware, this is both with regard to submitting batches of messages, transmitting batches and also generally performing operations on messages in batches. It is recommended that the size of these batches be configurable from the adapters' design-time.

As mentioned earlier, transmitters should always perform non-blocking sends to avoid degrading the performance of the send host. Further blocking of engine API's is not recommended in general. Adapters should endeavor to expose performance related behavior via the UI, for example the size of a receive adapters batch to be submitted, perhaps the number of bytes within a given batch.

Writing to and reading from the message context is not a 'free' operation in terms of run-time performance, in general adapters should avoid reading, writing and promoting excessive numbers of message context properties. Further, the number of properties that are promoted when submitting messages has a cost associated with it due to the subscription evaluation, an adapter would need to promote a lot of properties to impact the performance, as a good practice it is recommended that only those properties that are required to be promoted are promoted.

Throttling Send and Receive

When the load on the BizTalk engine exceeds the configured threshold the engine will throttle adapters and orchestrations in order to ensure optimum performance. On the receive side, when the work load on the engine exceeds a given threshold the adapters call to IBTTransportBatch.Done() will be blocked until the load has decreased sufficiently, thereby stopping the adapter to submit new work into the engine. One the send side, when the engine is throttling adapters sending messages outbound, the engine will simply not deliver new messages to be transmitted until the load on the engine is reduced. For these reasons, the adapter does not need to be concerned with throttling unless it is required for example to limit the number of connections to a backend system, for these types of scenarios, neither the engine nor the Adapter Framework provides any support.

You can handle throttling the number of messages sent from a custom send adapter in numerous ways depending on whether you control the source code for the adapter.

Send-side throttling improves performance

If you control the source code for the adapter, you can determine from heuristics the maximum number of messages that you want to have in the queue to send at any time. When the Messaging Engine calls the **TransmitMessage** method and passes the send adapter a new message, you can choose either to block the thread or check to see if the number of messages in the queue is larger than the maximum value you determined previously. If the maximum number of messages has been exceeded, you can use the **Resubmit** method to resubmit the message back to the Messaging Engine. Note that if the adapter is synchronous, the message would already be blocked.

If you do not control the source code for the adapter, you can change the number of queued messages by changing the **Highwatermark** value in the Adm_serviceclass table in the BizTalk Management database. The maximum value for the Highwatermark property is 200. You can also change the value for the **Lowwatermark** to a smaller value.

Remember that the value of the **Highwatermark** property for asynchronous adapters accounts for the number of messages that the Messaging Engine has given to the adapter through the **TransmitMessage** method, for which the adapter has not made a corresponding call to the **DeleteMessage**, **Resubmit**, **MoveToNextTransport**, or **MoveToSuspendQ** methods. For synchronous adapters, the **Highwatermark** property only accounts for the number of messages the Messaging Engine has passed to the adapter by using the **TransmitMessage** method.

If you are writing a send adapter for a protocol that is inherently slow in nature (such as HTTP, FTP or two-way SOAP), consider the following:

- Such an adapter might receive messages for transmission from the BizTalk Server messaging engine faster than it can transmit them. This discrepancy cause problems at various levels. The messages under transmission remain in memory and take up the virtual memory, slowing down the entire system.

- The adapter might take up protocol-specific resources. For example, it might open too many concurrent connections to the server, which could disrupt the remote server.

- The adapter might affect other adapters. For example, if too many messages queue up for a particular adapter, the BizTalk Server messaging engine will stop issuing requests to other send adapters in that process.

A solution is to put the slow and fast adapters in separate BizTalk Server hosts and control the number of messages using the "High Watermark" and "Low Watermark" settings.

Receive-side throttling improves performance

Why would a customer want to slow down a receive adapter? There are numerous situations in which a receive adapter receives messages faster than the rate at which the rest of the system can process the messages. When such a situation occurs, the MessageBox database becomes backlogged. When this happens, the performance of the whole system drops dramatically.

If this is happening with your adapter, you can use one of the following techniques to reduce the speed of the receive adapter.

1. Reduce the messaging engine thread pool size. A user can control the number of threads used by the messaging engine to publish messages into the MessageBox. By reducing the number of threads used by the messaging engine we reduce the rate at which the receive adapter will receive messages into the MessageBox. This setting only needs to be done for the host corresponding to the receive handler for the adapter. You should not set this for the host corresponding to the send handler for the adapter, unless you want to slow down the send adapter as well.

2. Reduce the adapter batch size. Most fast receive adapters publish messages to the MessageBox in batches. The size of these batches is usually configurable in the receive location property page. By decreasing the batch size we can decrease the overall throughput of messages coming into the system.

3. Change other adapter specific settings. After you complete the two previous steps, you can try adjusting other adapter parameters to further decrease throughput. Some adapters expose internal parameters that can be used to decrease throughput. For example, MQSeries adapter has a setting for “Ordered Delivery.” Ordered Delivery specifies that the adapter will publish a batch of messages, wait for it to complete, and then publish the next batch. By enabling this setting, you essentially remove all parallelism from the receive adapter. Conversely, tuning the parameters in the opposite way can be used to increase the receiving rate of a receive adapter.

An adapter can submit as many batches as required to the transport proxy. When the system is heavily stressed, a call to the **Done** method of the **IBTTransportBatch** interface will block the message until the required resources are released to the system.

Plan for asynchronous receive and send

The BizTalk Server messaging APIs have rich support for asynchronous programming. If you want to write a scalable adapter, plan on using the asynchronous model from the start because the asynchronous model provides better concurrency.

On the receive side, when an adapter submits a batch of messages to the BizTalk Server messaging engine (by calling IBTTransportBatch::Done()), the messaging engine queues up the work using its internal thread pool and returns immediately. The messaging engine processes the messages on a separate thread, leaving the adapter free to read more messages from its source and submit them without waiting for the previous message processing to complete.

On the send side your adapter can be either asynchronous or synchronous. However, if your protocol supports asynchronous operations, you should use this support to write a scalable adapter. For example, FILE and HTTP send adapters are fully asynchronous and they perform very few blocking/synchronous operations.

Asynchronous operations ensure that both the BizTalk Server messaging engine and your adapter will continue to do their respective work in parallel and not wait on each other for normal message processing.

Use batching to improve performance

Batching is the best starting point for writing a scalable adapter. This is true for both send-side and receive-side adapters. As explained in the Batching section of this document, every batch goes through a database transaction within BizTalk Server even if your adapter is non-transactional. Because there is a fixed delay associated with each transaction, you should try to minimize the number of these transactions by combining more than one operation into a single batch.

Don't starve the .NET thread pool

Writing BizTalk Server adapters is an exercise in writing .NET runtime code; writing .NET runtime code is invariably an exercise in asynchronous programming.

While starving the .NET thread pool is a risk to all asynchronous programming in .NET, it is particularly important for the BizTalk Server adapter programmer to watch out for this. It has impacted many BizTalk Server adapters: take great care not to starve the .NET thread pool.

The .NET thread pool is a limited but widely shared resource. It is very easy to write code that uses one of its threads and holds onto it for ages and in so doing blocks other work items from ever being executed.

Whenever you use BeginInvoke or use a Timer, you are using a .NET thread pool thread. If you have multiple pieces of work to do (for example copying messages out of MQSeries into BizTalk Server), you should execute one work item (one batch of messages into BizTalk Server) and simply requeue in the thread pool if there is more work to do. What ever you do, don't sit in a **while** loop on the thread.

In concrete terms this means replacing while loops with repeated calls to BeginInvoke.

This simple change will dramatically improve the responsiveness and ability to scale out for the whole implementation. It is possible to have a mainframe dump a million messages on an MQSeries queue ready to be consumed by BizTalk Server: in that situation you don't want to be sitting on a while loop eating through those messages.

Choose the right measurement when limiting batch size

If you are submitting messages to BizTalk Server in batches, don't limit the batch just based on the message count. Consider what happens when an adapter has been configured to batch based on just message count: If the batch size is two and the adapter gets four messages of size 4KB, 8KB, 1MB and 5MB respectively, the first batch will be of size 12KB, and second batch will be of size 6MB.

Because the BizTalk Server messaging engine processes all messages in a single batch sequentially, the second batch will be processed much more slowly than the first batch. This is effectively reducing throughput. A better way to handle this problem is to batch based on **both** message count and total bytes in the batch (that is, batch size in bytes). There is no magic number for total bytes. However, in a normal processing scenario, if the batch size exceeds 1MB, you will start seeing poor concurrency and throughput.

Generally adapters are message agnostic and they do not know what the size of the messages will be in the production environment. The sizes of the incoming messages are likely to vary quite significantly. Because of this, always use message count and total bytes to build the batch.

Testing and Debugging

Debugging run-time problems often requires a multi faceted approach, requiring data to be gathered from multiple sources such as software tracing, performance counters, the event log entries, WMI events and debugging source code in order to determine the cause of problems or software bugs.

Since receive and send adapters run in the address space of the BizTalk Service the debugger needs to be attached to the BtsNtSvc.exe process in order to debug an adapter. However for isolated receive adapters, the debugger will need to be attached to what ever process it is hosted in, this is of course different depending on the adapter.

The adapter run-time code should be instrumented with tracing code in order to capture run-time diagnostics for trouble shooting. This may be achieved using the Microsoft Enterprise Instrumentation Framework (EIF). Typically it is useful to add trace statements for different levels of severity, for example tracing for error conditions only, tracing for errors and warnings, and finally tracing for errors, warnings and informational statements. Further, more complex adapters may have separate sub-systems which need to be traced in isolation to each other. For example, an adapter may have a network sub-subsystem and a core sub-system, enabling tracing for all sub-systems may generate an excessive amount of ‘noise’ in some scenarios.

Performance counters should be added for the adapter to capture rates and values at run-time to further give a view into the run-time behavior of the adapter, for example an adapter may wish to publish performance counters for the raw numbers of messages sent on a per-endpoint as well as the rate of message sent per second.

WMI events may also be useful for some critical error scenarios, for example, if an adapter hits a critical error which causes it to shutdown a receive location, firing a WMI event would allow an administrator to listen on that even and take some appropriate action.

Adapter Testing

Before shipping a custom adapter for BizTalk Server it should be remembered that an adapter should be developed to enterprise software quality, while it is not the intention to completely detail how adapters should be tested, this section gives a flavor for what needs to be tested. In general the testing of run-time code such as adapters should cover three broad categories, functional testing, stress testing and performance testing.

Function Testing

The adapter should be tested for all permutations of its functionality, this includes both positive tests and negative tests, positive tests should include but not be limited to the following scenarios:

- Receive a message(s)
- Transmit a message(s)
- Transmit a message using a dynamic port
- Disable receive locations
- Update configuration
- Ensure service windows are working for both receive and send adapters
- Ensure transactional integrity for transacted adapters
- Negative tests should include but not be limited to:
- Receive a bad message(s)

- Receive a mixed batch of messages, some good and some bad
- Transmit failure:
- Retry successful
- Retry fails, move to next transport successful
- Move to next transport fails, suspend message
- Transmit a mixed batch of messages
- Database fail over

Stress Testing

Adapters are run-time components and as such should meet the stringent requirements for run-time software. Typically stress testing is carried out by running scenarios under load for a period of time, further high stress and low stress mean time between failure test should be performed, whereby the adapter is run under load for a measured time period.

Some rough guide lines for these might be running the adapter at high stress for something in the order of 72 hours; where by the number of messages through the adapter causes the CPU utilization to be in the region of 80 to 90%. For low stress, the CPU utilization would be in the region of 30 to 40% for something in the order of 120 hours.

Performance Testing

Adapters should be developed with performance in mind, this paper has high lighted many areas where performance maybe optimized. Before releasing an adapter its performance should be validated, it is important to ensure its performance scales up and scales out, in other words, adding more CPU's will lead to a performance increase as will adding more machines. Profiling the code can help to eliminate performance bottlenecks.

Adapter Certification Program

Adapters that are to be commercially marketed and sold should be run through the BizTalk Server 2006 Certification Program, this is an independent validation program that is run by Unisys. The program was developed in conjunction with Microsoft and has a number of targeted tests designed to find deficiencies in adapters.

The program covers the following:

- BizTalk Server Adapter Framework Compliance

- Setup and Configuration
- Reliability Under Stress
- Fault Tolerance
- Security
- Globalization
- Diagnostics

More information can be found regarding the certification program by emailing mailto:biztalkcertification@unisys.com or visiting the web site http://www.unisys.com/about__unisys/partners/alphabetical__listing/microsoft/microsoft__programs

Localization Issues

The following topics cover localization issues that you may encounter when developing custom adapters.

XSD issues

By using the Adapter Framework, adapter developers can implement adapter property pages with XML Schema Definition (XSD) schemas.

If your adapter has no globalization or localization requirement, then you can hardcode the XSD schema string inside the IDynamicAdapterConfig:GetConfigSchema() function.

If your adapter has globalization or localization requirements, you can implement the XSD schema in one of two ways.

- Use separate XSD files outside the design-time binary. Make the whole text of the schema a manifest resource.
- Dynamically replace the Property Names and Description from the resource:
  - Add a _locID to each element that you want to localize.
  - Use an xpath to pull back all the nodes in the schema that have a _locID attribute.
  - Look up the resources for a string indexed by the value of the _locID.
  - Replace the node text with the result.

Design Issues

This section contains hints and tips that adapter developers have learned while designing adapters.

Handler properties should be strings if used as default configurations

Although it seems attractive to use the properties on the XSD-generated handler property sheet as defaults for their Location properties, there are several issues that make this less useful.

This seems attractive because if the value is not set in the Location the runtime automatically uses the value set in the handler.

The problem comes with knowing whether the value presented to the runtime is to be overridden or not. The typical way of doing this would be to have some notion of NULL defined for values and then a test could be made against that value. The problem when using the XSD based property sheets in BizTalk Server is that NULL is only supported for strings.

Even if you want your adapter to have default settings through the use of this NULL test and are willing to restrict the adapter to string types, it is still exposed to a very odd piece of user interface.

The XSD-generated property sheets only support the setting of a property back to NULL by right clicking on the property at which point a **nullify?** context menu appears and the property can be set to NULL. There is no visual feedback at all as to whether a property is NULL or not.

Implementing Schema Generation Wizards

Ideally programmers like to code against strongly typed object models. Manipulating XML in code can at first seem very awkward and prone to error, but a couple of tricks and smart use of the support offered by the .NET Framework can dramatically simplify matters.

Do not create XML documents with string concatenation

One of the worst mistakes to make with XML is to try and generate it from string concatenation and print statements in memory. Even for the most trivial XML snippet, it is easier to use a tool like XmlWriter or the DOM.

If you are using XmlWriter, you should never be tempted to use the raw write capability because the writer immediately loses the state of the document, and you are left on your own: that is how to get incorrect XML out of an XmlWriter!

At run time, the XmlWriter is preferred over the Xml DOM because of memory bloat issues associated with the DOM. However, at configuration or design time you may have more room for movement. Using the DOM facilitates the use of xpath queries which can be a very nice additional tool.

Consider defining the skeleton of your XML document as a resource

If you are generating a large XML document from a design tool and that generated document always follows the same basic structure, consider placing the whole skeletal XML file as a resource in the project. Now it is easy to edit it when you need to.

Load the code into a DOM and then add the necessary flesh to the bones of the document by using xpath to pick out the node you want to add it to.

This is exactly the technique used by the SQL adapter Schema Wizard that shipped with BizTalk Server 2006. In this case, you are creating a WSDL file, so the wizard stores the skeletal WSDL file in a resource and adds the generated XML Schema Definition (XSD) child parts by using selectNode with an xpath to find the right parent. This is user interface code so performance is not an issue; the resulting implementation is very clean, robust, and maintainable.

Consider placing processing steps in the BizTalk Server pipeline

In general the adapters built at Microsoft move message format-based processing out of the adapter and into the BizTalk Server pipeline. A good example is an adapter to a structured but non-XML data source.

In this case, it just gets the data and the BizTalk Server pipeline is used to parse it and convert it into an XML equivalent. The benefit is that the pipeline component itself becomes a reusable piece of the architecture.

Make adapter behavior configurable

One of the lessons learned from the MQSeries adapter beta program was that not all customers were happy with the same behavior, particularly when it came to how to handle errors and ordering.

The solution was to make the behavior configurable: you can specify whether the adapter is to support ordering, whether failures are moved to the suspend queue, or whether they cause the adapter to stop processing and disable itself.

Making such behaviors configurable can significantly simplify customers' lives where they would otherwise have to write complex orchestrations or scripts external to BizTalk Server to achieve the same result.

Support correlation with message queues

Many messaging platforms support the notion of a correlation ID in the message header to support an application level request response scenario. Examples include

MQSeries, MSMQ, and SQL Service Broker. It would seem attractive to map the request response pattern of the external messaging system to a send-response adapter in BizTalk Server, however this doesn't make sense because of where the transactions lie. Specifically, the send to the external messaging system requires a transactional commit before the other end of the queue sees the data. The receive must also a separate transaction.

The solution in BizTalk Server is to:

- Use correlation sets in orchestration
- Configure two separate ports: one for the send and one for the receive

In a simple case the orchestration specifies the correlation ID that will be associated with the message by the adapter (it would be passed to the adapter as a context property on the message). In a more complex case, the scenario calls for the external messaging system to allocate the ID; in this case it can be passed back from the send port to the orchestration with a response message. This response message is just to pass back the ID and not the true message response.

Deploying a Custom Adapter

A complete adapter installation process consists at least following steps:

- Check dependencies. For example, the MSMQ adapter installer checks for the existence of the local MSMQ Service because the adapter runtime depends on it.
- Setup the local file system. This includes creation of installation folders and copying binary and support files.
- Put managed assemblies into the system global assembly cache.
- Create registry keys for the adapter.
- Add the adapter to BizTalk Server. This means to make new entries in the BizTalk management database for the adapter as well as for the adapter context property schema. This step is sometimes done manually using the BizTalk Server Administration plug-in for the Microsoft Management Console (MMC).

Exceptions

The adapter installer may not need to check the dependencies in partial BizTalk Server installations. For example, in the BizTalk Server Administration only installation, the adapter installer does not need to check adapter runtime dependencies because the adapter runtime is not used. The same is true in the BizTalk Server runtime only installation, where the adapter installer does not need to check the adapter design time dependencies.

In multiple BizTalk Server environments, step 5 in the list immediately above (Add the adapter to BizTalk Server) only happens in the adapter installation on the first BizTalk Server, because all BizTalk Server service instances share the same BizTalk management database (with the default name, BizTalkMgmtDB). If step 5 is performed more than once in these environments, the additional steps fail. An adapter installer should keep this in mind.

Install the adapter assemblies into the Global Assembly Cache (GAC)

To support multiple BizTalk Server host environment with different adapter installation paths, the adapter assemblies must be installed into system global assembly cache.

During the adapter installation and configuration, a new entry is created in the adm_adapter table of the BizTalk management database (with the default name BizTalkMgmtDB) for this newly added adapter. This entry contains all reference information used by BizTalk Server for both runtime and design time.

Two fields, InBoundAssemblyPath and OutboundAssemblyPath are used by BizTalk Server runtime to find out where to load the adapter binaries. Because there is only one BizTalk Management database for a BizTalk Server group, all BizTalk servers in the group must share this same piece of information. If you want to install the adapter on different BizTalk Servers within the group, you must use the same installation path on each of those servers. If the local installation path is different from the path stored in the BizTalk Management database, BizTalk Server will not be able to load the adapter binaries.

Therefore, always sign the adapter with a strong name and use Gacutil.exe to put the adapter assembly into system global assembly cache during the adapter installation. Make sure that you leave the two fields InBoundAssemblyPath and OutboundAssemblyPath null, or empty.

Using the Framework for Adapter Configuration

BizTalk Server 2006 introduced a mechanism for generating property sheets for the adapter configuration user interface based on an XML Schema Definition (XSD) definition of the configuration properties. The use of this framework introduces a number of significant challenges for the adapter writer. This section covers the various problems this technology introduced and suggests effective workarounds.

Consider custom property editors for all your key properties

It is impossible to put significant property validation on top of properties in the property sheet. The framework attempts to use XML Schema Definition (XSD) simpleType restrictions to define the validation rules for the UI. The simple rules for maximum and minimum value are applied but the regular expression restriction for string fields is not supported.

Also, the data is converted into CLR types before the restriction is applied so the user of the UI can sometime get a cryptic type conversion error rather than an out of range error. All in all, the validation logic is very weak.

The solution many adapter writers have adopted is to write custom property editors for the main properties on their property sheet. If there are a number of cross dependent properties (as is often the case), then the custom property editor can mangle the result into a single field and the runtime and then un-mangle it. This is the only way to get the necessary (though still generally trivial) custom code into the interface.

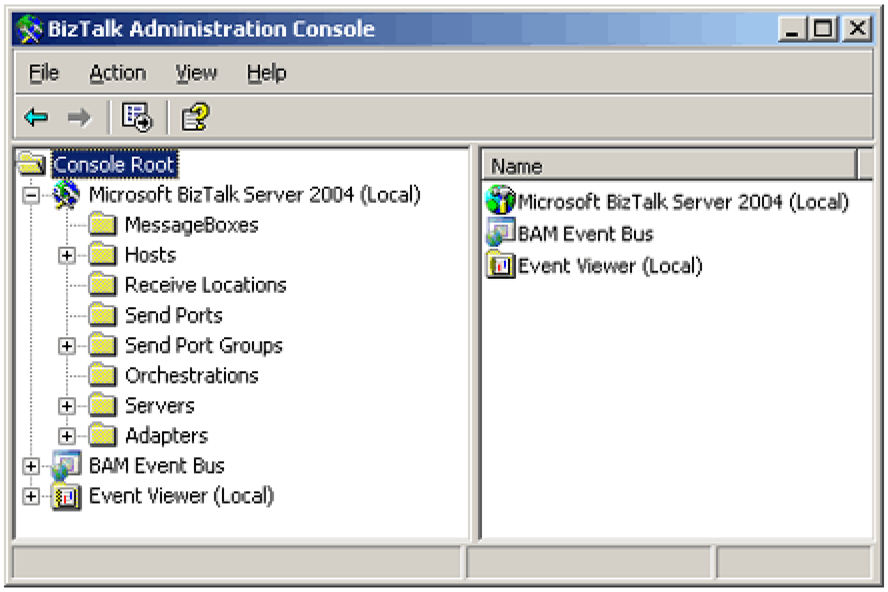
Custom Property editors are relatively straightforward to write and the property in the XSD can be annotated to use them. The annotation references an assembly which exposed the property editor entry point, and it is coded to show a CLR Form. You can now write a regular user interface and use the traditional interface generation tools that Visual Studio has to offer.

Using the Adapter Framework Tools

The Adapter Framework uses the following tools and views to manage and develop custom adapters:

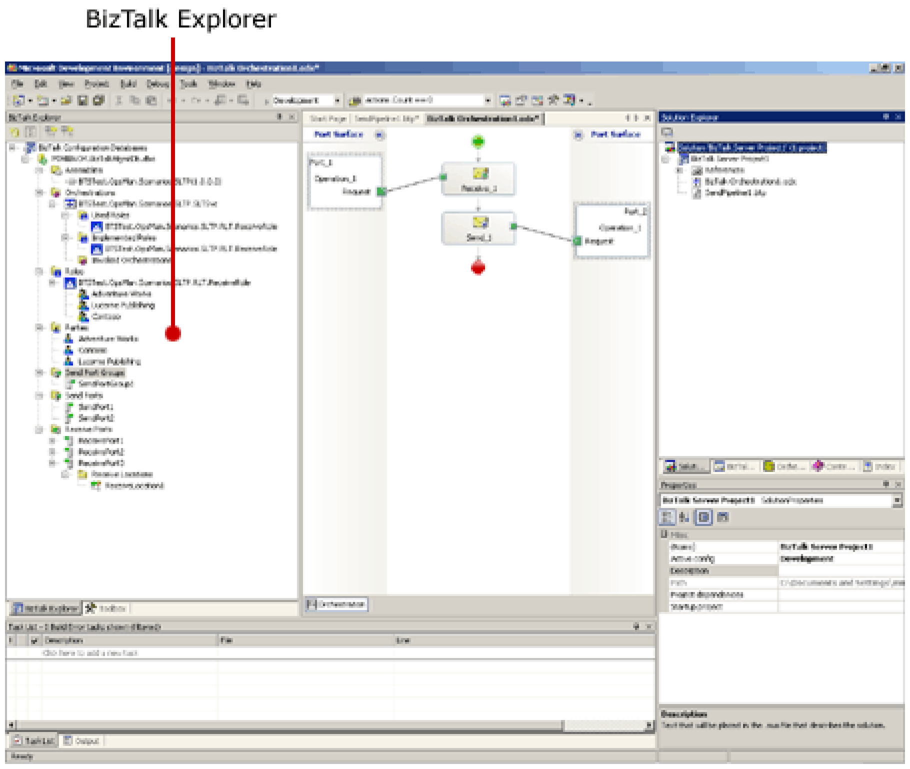
- **BizTalk Administration Management Console.** This is an independent application, available from the **Start** menu, used to install new adapters and to manage their receive and send handlers.

  Illustrates the BizTalk Administration console



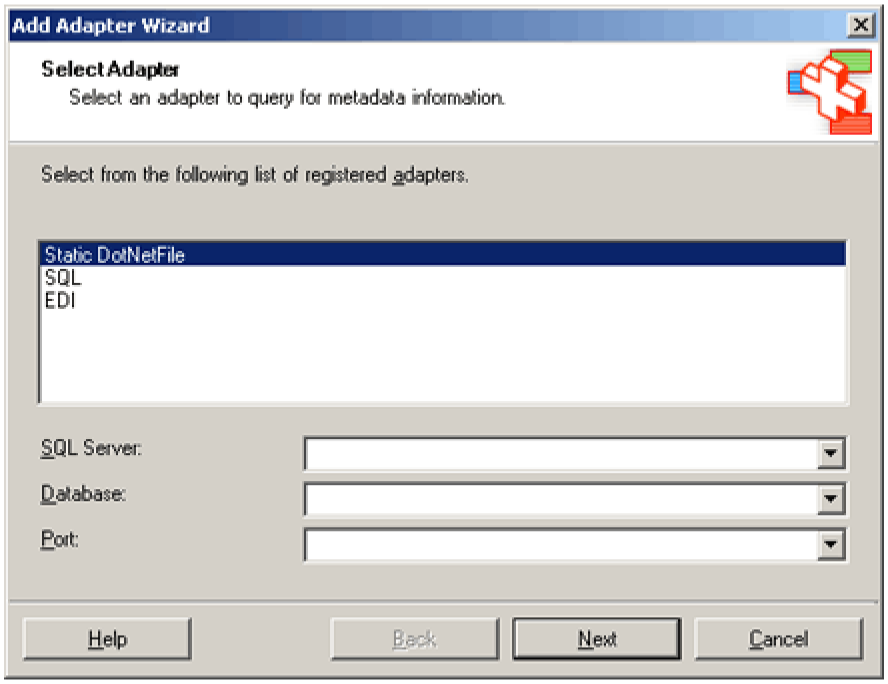
- **BizTalk Explorer.** This tool is available in Microsoft® Visual Studio® 2003. It is available from the **View** menu, and appears near the toolbox on the left side of the screen. Use it to configure all of the send ports and receive locations associated with an adapter.

Illustrates the BizTalk Explorer



- **Add Adapter Wizard.** This wizard is available from within a BizTalk project in Microsoft Visual Studio 2003. It is available by right-clicking the project name in the Solution Explorer window in Microsoft Visual Studio .NET, and then clicking **Add Generated Items**. The Add Adapter Wizard is available for developers that want to communicate with application adapters like SQL or SAP. Use this wizard to pull schemas corresponding to these application adapters into the system.

Illustrates the Select Adapter page of the Add Adapter Wizard


Add Adapter Wizard

Select Adapter

Select an adapter to query for metadata information.

Select from the following list of registered adapters.

Static DotNetFile
SQL
EDI

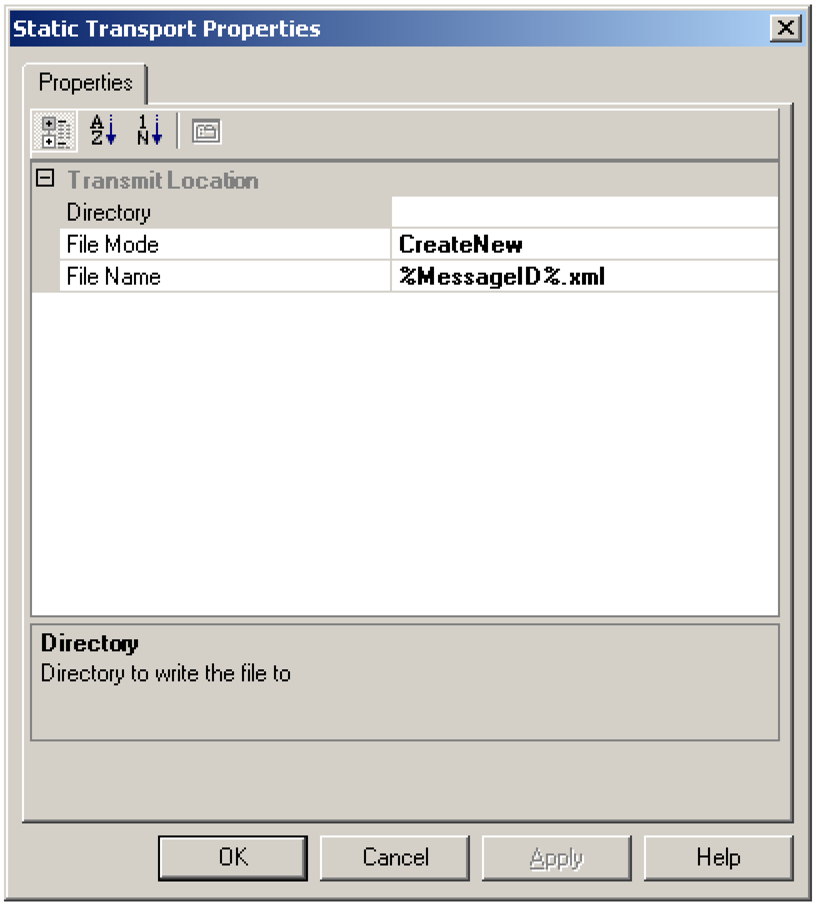
SQL Server:

Database:

Port:

- **Adapter property pages.** These are available through the BizTalk Administration console or BizTalk Explorer. Use these pages to configure receive handlers, receive locations, send handlers, and send ports for your adapter.

Illustrates a send port property page



Transmit Location

Directory	
File Mode	CreateNew
File Name	%MessageID%.xml

Directory
Directory to write the file to

This section provides tips and shortcuts for using these tools. For instructions on configuring a native adapter, see Using Adapters .

This section contains:

- Working Within Adapter Property Pages
- Adding Adapter Metadata to a BizTalk Project

Working Within Adapter Property Pages

Adapter property pages are named <Adapter name> Transport Properties, and are accessed through BizTalk Explorer or the BizTalk Administration console to set properties for the native or custom adapter.

This topic provides instructions for using property page features common to all adapters.

To set the value of an optional string property to null

1. In the property grid, right-click the property name field (not the property value field), and then click **Nullify <property name>**.

To change the size of the description pane on an adapter property page

1. The size of the description pane does not change based on the size of the description; therefore long descriptions may appear truncated. To view the entire description, resize the description pane by moving the double-headed arrow that appears on the top border of the pane.

Adding Adapter Metadata to a BizTalk Project

The Add Adapter Wizard enables you to add adapter metadata, such as schemas, message types, and port types needed to communicate with an adapter from an orchestration, to a BizTalk project. Use the Add Adapter Wizard with application adapters, such as SQL and SAP, to pull schemas corresponding to these application adapters into the system.

The following procedure walks you through the generic steps in the adapter. For specific instructions on using the wizard with the SQL adapter, see Adding SQL Adapter Schemas to a BizTalk Project.

To add adapter metadata to a BizTalk project

1. In your Visual Studio .NET BizTalk project, in Solution Explorer, right-click your project, click **Add**, and then click **Add Generated Items**.

2. In the **Add Generated Items - <Project name>** dialog box, in the **Templates** section, select **Add Adapter**, and then click **Open**.

3. In the Add Adapter Wizard, on the **Select Adapter** page, do the following.

Use this	**To do this**
Adapter list box	Select the registered adapter to add to the project.
SQL Server	Enter the BizTalk Database Server name.
Database	Displays the list of BizTalk Management databases for the chosen server.
Port	Optional. Displays the list of ports created and stored in the BizTalk Managment database. Only ports configured to work with the selected adapter are shown.

4. Click **Next**.

5. If you are using a static adapter, on the **Select Services to Import** page, select a set of available services from the tree view, and then click **Finish**.

   –Or–

   If you are using a dynamic adapter, follow the steps in the custom user interface to complete the wizard.

   After you complete the wizard, Visual Studio lists the .odx and .xsd files in Solution Explorer.

Modifying the Design-Time Configuration of the Sample File Adapter

The Microsoft® BizTalk® Server 2006 Software Development Kit (SDK) includes a sample file adapter that you can use as a template on which to can create and customize your own adapter solutions. This section explains how to modify the sample file adapter provided in the SDK to create your own custom adapter.

This section provides two sets of instructions. The first set walks you through the changes you will have to make to create a custom static design-time adapter. A custom static design-time adapter uses a fixed set of interfaces to gather data. The next set of instructions walks you through the changes you will have to make to create a custom dynamic design-time adapter. A dynamic design-time adapter uses a dynamic or variable set of schemas to retrieve data. Dynamic adapters are commonly application specific, and are designed to gather SAP or SQL schemas dynamically based on the subsystem.

A prerequisite to creating a custom adapter by using the steps in this section is to install, build, and run the sample file adapter provided in the SDK. For instructions on installing, building, and running the sample file adapter, see File Adapter (BizTalk Server Sample).

This section contains:

- Developing a Static Design - Time Configuration for a Custom Adapter
- Developing a Dynamic Design-Time Configuration for a Custom Adapter

Developing a Static Design - Time Configuration for a Custom Adapter

This section explains how to modify the sample adapter to create a static design-time configuration for your custom adapter. This procedure only calls out the places in the sample file adapter where modifications need to occur; the changes you decide to make will be based on the needs of the applications with which the adapter communicates and the logic that the adapter needs to implement. Links to sections of BizTalk Server Help that describe these steps in more detail or provide additional background are provided when available.

Overview of the development process

1. Install, build, and run the sample file adapter.

2. For information about installing and building the sample file adapter, see File Adapter (BizTalk Server Sample).

3. Create a list of adapter requirements.

4. To determine what kind of adapter you want to develop, review the list of adapter variables in About Adapter Variables.

5. Create a list of adapter configuration requirements.

6. Determine the configuration parameters that will need to be set. Determine if the parameters are global (for all receive locations and send ports) or port specific.

7. Modify the registry keys of the adapter.

8. For more information about this step, see Modifying the Adapter Registration File.

9. Modify the adapter property pages to account for any new configuration parameters.

10. To modify the handler configuration files, see Modifying the Adapter Handler Configuration Schemas. To modify the receive location or send port configuration files, see Modifying the Adapter Send Port and Receive Location Configuration Schemas.

11. Test the changes made to the adapter property pages.

12. For more information about this step, see Testing the Adapter Configuration XSD Files.

13. Modify the tree view of the schema categories in the Add Adapter Wizard.

14. For more information about this step, see Modifying the Schema Categories in the Add Adapter Wizard.

15. Modify the sample code to return schemas as Web Services Description Language (WSDL) files.

16. For more information about this step, see Modifying the GetServiceDescription Method.

17. Modify the existing WSDL files or create new WSDL files.

18. For more information about this step, see Modifying the WSDL Files.

19. Modify the sample code to return any additional XSD files needed by the adapter that are not included in the WSDL.

20. For more information about this step, see Modifying the GetSchema Method.

21. Rebuild and test the Add Adapter Wizard.

22. For more information about this step, see Rebuilding and Testing the Adapter Management Project.

This section contains:

- Modifying the Adapter Registration File
- Modifying the Adapter Handler Configuration Schemas
- Modifying the Adapter Send Port and Receive Location Configuration Schemas
- Testing the Adapter Configuration XSD Files
- Modifying the Schema Categories in the Add Adapter Wizard
- Modifying the GetServiceDescription Method
- Modifying the WSDL Files
- Modifying the GetSchema Method
- Rebuilding and Testing the Adapter Management Project

Modifying the Adapter Registration File

The StaticAdapterManagement.reg and DynamicAdapterManagement.reg files register either the static or dynamic sample file adapter with the global assembly cache. You can modify each of these files to register your custom adapter.

If your custom adapter is similar to the sample adapter, you may want to modify the existing registry files. Otherwise, you can create a new registry file by using the Adapter Registration Wizard. The Adapter Registration Wizard gives you all the same options as creating a registry file from scratch; it just reduces the likelihood of errors in the file. For more information about the Adapter Registration Wizard, see Adapter Registration Wizard.

If you decide to modify the existing registry files instead of using the Adapter Registration Wizard, open and modify the following properties in the StaticAdapterManagement.reg or DynamicAdapterManagement.reg file located at <*drive*>:\Program Files\Microsoft BizTalk Server 2006\SDK\Samples\Adapters\File Adapter:

- Constraints
- InboundTypeName
- InboundAssemblyPath
- OutboundTypeName
- OutboundAssemblyPath
- AdapterMgmtTypeName
- AdapterMgmtAssemblyPath
- PropertyNameSpace

The following code is from the StaticAdapterManagement.reg file. Each of the properties that you need to modify is in bold. For more information about each of these properties, see Adapter Registration.

Modifying the Adapter Handler Configuration Schemas

The sample file adapter included in the SDK has a set of XSD files used to configure the receive location, send port, receive handler, and send handler of the adapter. Modify these XSD files so that your custom adapter receives the configuration properties it requires.

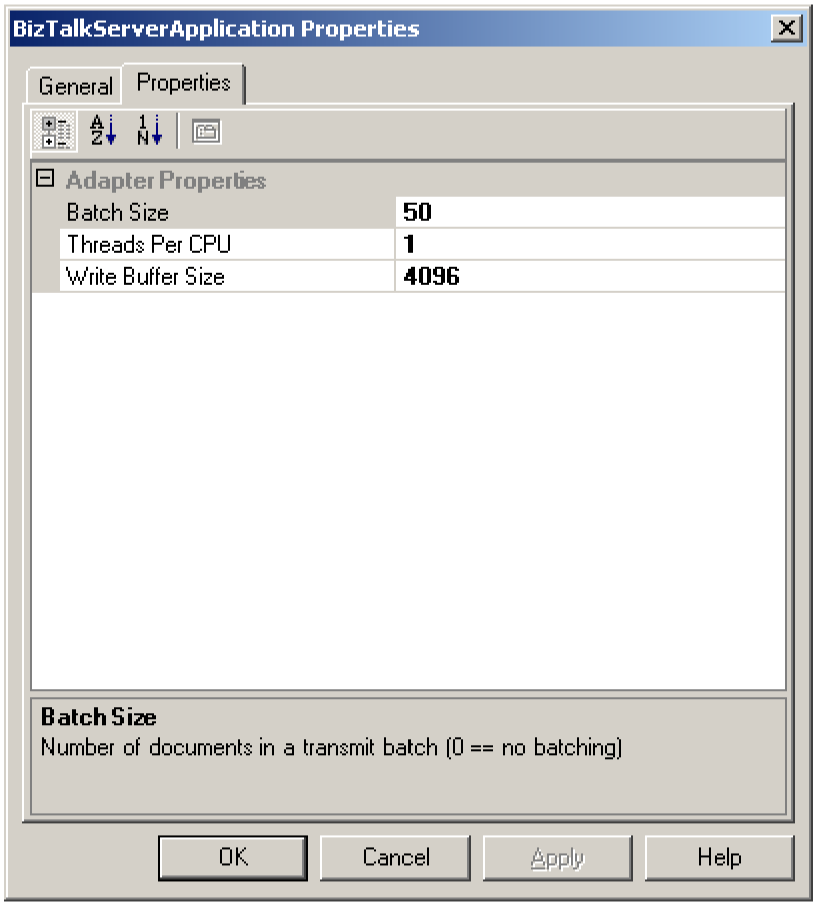
Using your adapter requirements, create a list of configuration properties required for each of the endpoints. If all of your configuration properties are global, you may only need to modify the send and receive handler configuration. If the adapter properties need to be set for each port, you have to modify the receive location and send handler configuration files as well.

The files included with the sample file adapter that you need to modify are the TransmitHandler.xsd and ReceiveHandler.xsd schema files, which configure the send handler and receive handler respectively. These files are located in the <*drive*>:\Program Files\Microsoft BizTalk Server 2006\SDK\Samples\Adapters\File Adapter\Design Time\Adapter Management directory. These XSD files control the property pages used to configure the send and receive handlers in the BizTalk Administration console.

The Adapter Framework provides schema extensions and advanced configuration options to support common adapter configuration requirements, and provides extensions that are not in the schema included with the sample file adapter. For more information about the Adapter Framework schema extensions, see Adapter Framework Configuration Schema Extensions. For more information about advanced configuration options such as custom drop-down editors and custom type converters, see Advanced Configuration Components for Adapters.

The code that follows is from the TransmitHandler.xsd file, and produces the following property page.

Illustrates the send handler property page created by the TransmitHandler.xsd file



Adapter Properties

Batch Size	50
Threads Per CPU	1
Write Buffer Size	4096

Batch Size
Number of documents in a transmit batch (0 == no batching)

Note the use of the <baf:designer>, <baf:displayname>, and <baf:description> tags. These are custom decorations provided by the Adapter Framework to make the generation of these property pages faster. For a list of all of the decorations available for use within the Adapter Framework, see Adapter Framework Configuration Schema Decoration Tags.

Also, note that the schema has only one element and does not contain a URI element. Because of this, you cannot use this schema to configure a send port or receive location through BizTalk Explorer, where setting the URI of the send port or receive location is required.

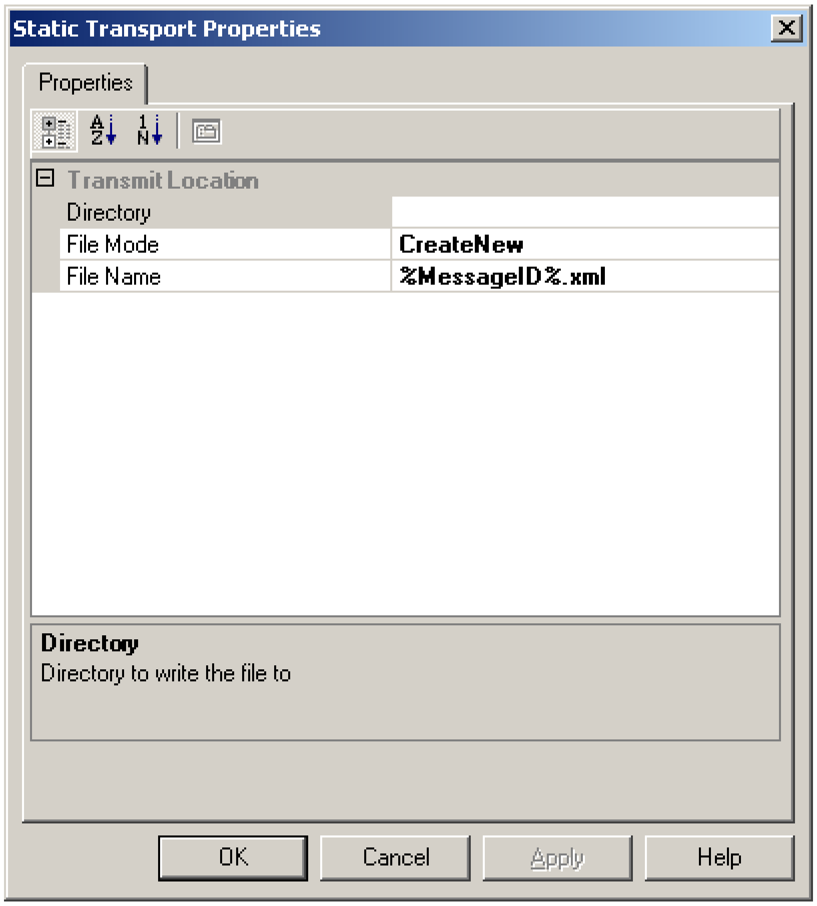
Modifying the Adapter Send Port and Receive Location Configuration Schemas

If you need to set port-specific properties for your adapter, you need to modify the receive location and send port configuration schemas. The TransmitLocation.xsd and ReceiveLocation.xsd schema files, which configure the send port and receive location respectively, are located in the <*drive*>:\Program Files\Microsoft BizTalk Server 2006\SDK\Samples\Adapters\File Adapter\Design Time\Adapter Management directory. These XSD files control the property pages used to configure the receive location and send port in BizTalk Explorer.

The Adapter Framework provides schema extensions and advanced configuration options to support common adapter configuration requirements. For more information about the Adapter Framework schema extensions, see Adapter Framework Configuration Schema Extensions. For more information about advanced configuration options such as custom drop-down editors and custom type converters, see Advanced Configuration Components for Adapters.

The code that follows is from the TransmitLocation.xsd file, and produces the following property page.

Illustrates the send port property page for the sample file adapter



Transmit Location

Directory	
File Mode	CreateNew
File Name	%MessageID%.xml

Directory
Directory to write the file to

Note that the send port configuration contains the <baf:designer>, <baf:displayname>, and <baf:description> tags, just like the send handler, but it also uses the <baf:category> tag. The category tag enables you to group properties together. If you have more than one category, the category is expandable and collapsible, and appears in gray as a header above the properties in that category. For more information, see Adapter Framework Configuration Schema Extensions.

This schema also contains a URI field, which is populated on the page that appears after entering all of the field information on the send port property page during the validation processing by the adapter.

Testing the Adapter Configuration XSD Files

To test the changes made to the configuration schemas, open each of the property pages generated by the XSD file to ensure that it asks for and accepts the correct data.

To test the TransmitLocation.xsd file

1. In Visual Studio .NET, on the **View** menu, click **BizTalk Explorer**.
2. In BizTalk Explorer, right-click the **Send Ports** node, and then click **Add Send Port**.
3. In the **Create New Send Port** dialog box, select one of the following port types from the drop-down list, and then click **OK**:
   - Static One-Way Port
   - Static Request-Response Port
4. In the **Send Port Properties** dialog box, in the **Name** box, type the name of the send port.
5. In the **Send Port Properties** dialog box, expand the **Transport** node, and then select the **Primary** node.
6. In the right pane, in the **Transport Type value** box, select **Static**.
7. In the **Address (URI) value** box, click the ellipsis (**...**) button.

   The displayed screen should contain the fields you specified in the TransmitLocation.xsd file.

To test the ReceiveLocation.xsd file

1. If no receive ports are created, follow the instructions for Adding a Receive Port Using BizTalk Explorer, and then return to this procedure.
2. In BizTalk Explorer, expand the **Receive Ports** collection, expand the newly created receive port or the receive port to which you want to add a receive location, right-click the **Receive Locations** collection, and then click **Add Receive Location**.
3. In the **Receive Location Properties** dialog box, in the **Name** box, type a name for the receive location.
4. In the **Receive Location Properties** dialog box, in the left pane, select the **General** node.
5. In the **Receive Location Properties** dialog box, in the right pane, in the **Transport Type value** box, select **Static**.
6. In the **Receive Location Properties** dialog box, in the right pane, in the **Address (URI) value** box, click the ellipsis (**...**) button.

The displayed screen should contain the fields you specified in the ReceiveLocation.xsd file.

To test the ReceiveHandler.xsd file

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Administration**.
2. Double-click the **Microsoft BizTalk Server (local)** node, double-click **Adapters**, double-click **Static**, and then click **Receive Handlers**.
3. In the results pane, right-click the receive handler, and then click **Properties**.

   The displayed screen should contain the fields you specified in the ReceiveLocation.xsd file.

To test the ReceiveHandler.xsd file

1. In the BizTalk Server Administration console, double-click **Static**, and then click **Send Handlers**.
2. In the results pane, right-click the send handler, and then click **Properties**.

   The displayed screen should contain the fields you specified in the TransmitLocation.xsd file.

If you are creating a static design-time adapter, continue to Modifying the Schema Categories in the Add Adapter Wizard.

If you are creating a dynamic design-time adapter, continue to Modifying the

Modifying the Schema Categories in the Add Adapter Wizard

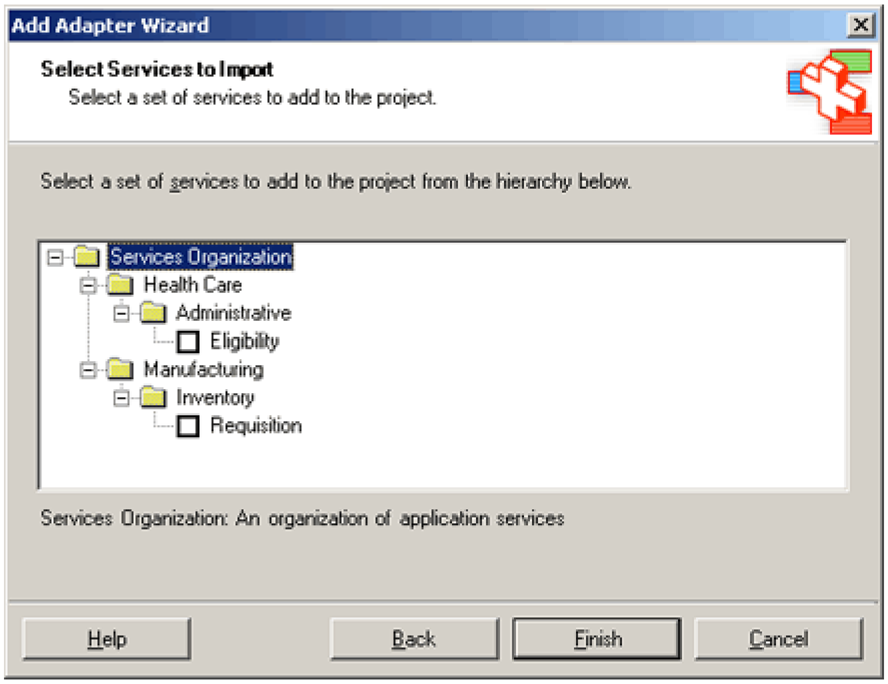
An adapter may use any one of thousands of schemas to transform data before passing it to BizTalk Server. When adding metadata to a BizTalk project, you must use the Add Adapter Wizard to select a schema from a list of all the schemas with which the adapter interacts.

In the sample file adapter, the CategorySchema.xml file populates a tree view organization of schemas. You should create the tree so that it organizes the schemas in a manner that is intuitive to your solution. The existing categories in CategorySchema.xml are just an example of what you can do in your own tree. The categories do not have any particular relevance to the data that is passed by the sample adapter. The organization of the schemas is particularly important with application-specific adapters, where thousands of different schemas may be available. For transport-specific adapters, this tree view organization is unnecessary.

The CategorySchema.xml file is located in the <*drive*>:\Program Files\Microsoft BizTalk Server 2006\SDK\Samples\Adapters\File Adapter\Design Time\Adapter Management folder.

The following figure shows the Select Services to Import page in the Add Adapter Wizard.

Illustrates the tree view of the schema categories in the Add Adapter Wizard


Add Adapter Wizard
Select Services to Import
Select a set of services to add to the project.
Help
Back
Finish
Cancel

Select a set of services to add to the project from the hierarchy below.

- Services Organization
  - Health Care
    - Administrative
      - ☐ Eligibility
  - Manufacturing
    - Inventory
      - ☐ Requisition

Services Organization: An organization of application services

Modifying the GetServiceDescription Method

The **GetServiceDescription** method of the **StaticAdapterManagement** class in the AdapterManagement.cs file returns schemas represented as Web Services Description Language (WSDL) files. The schemas that are represented in the WSDL file are added to your BizTalk project after you complete the Add Adapter Wizard.

The **GetServiceDescription** method should return the WSDL file that corresponds to the category that you selected in the tree in the Add Adapter Wizard. Again, this refers to the tree populated by the CategorySchema.xml file that you modified in Modifying the Schema Categories in the Add Adapter Wizard.

In the existing File Adapter sample code, Service1.wsdl is hard-coded. If you have more than one schema type available for selection in the tree, you will need more than one WSDL file. If you have many possible schema and WSDL choices, you may want to add a database lookup to return the correct WSDL file.

Modifying the WSDL Files

1. In the sample file adapter, the Service1.wsdl file contains the XSD files that BizTalk adds to the project. You may choose to modify the Service1.wsdl file or create your own WSDL file that contains the XSDs to add to your BizTalk project.

2. The following code is from the Service1.wsdl file.

Modifying the GetSchema Method

Modify the code in the **GetSchema** method of AdapterManagement.cs to return any external XSD files that are not included with the WSDL files.

You can embed small schemas in the WSDL for convenience. Or you can import large schemas using the WSDL for manageability.

Rebuilding and Testing the Adapter Management Project

To test all of the changes made to the AdapterManagement project, rebuild the project, then go through the Add Adapter Wizard, and ensure that all of the internal XSD and external XSD files are added to the AdapterManagement project.

For instructions on using the Add Adapter Wizard, see Adding Adapter Metadata to a BizTalk Project.

Optionally, you may want to remove the unused dynamic adapter code from the AdapterManagement.cs file if you created a static adapter, or remove the static adapter code if you created a dynamic adapter.

Developing a Dynamic Design-Time Configuration for a Custom Adapter

This section explains how to modify the sample adapter to create a dynamic design-time configuration for your custom adapter. This procedure only calls out the places in the sample file adapter where modifications need to occur; you will need to define your own requirements and determine what modifications you need to make. Links to sections of Microsoft® BizTalk® Server Help that describe these steps in more detail or provide additional background are provided when available.

In This Section

- Installing the Sample File Adapter

- Registering the Dynamic Sample Adapter

- Installing the Dynamic Sample Adapter

- Modifying the DisplayUI Method

Installing the Sample File Adapter

Introduction here.

Overview of the custom adapter development process

1. Install, build, and run the sample file adapter.

   For information about installing and building the sample file adapter, see File Adapter (BizTalk Server Sample).

2. Create a list of adapter requirements.

   Review the list of adapter variables in About Adapter Variables to determine what kind of adapter you want to develop.

3. Create a list of adapter configuration requirements.

   Determine the configuration parameters that will need to be set. Determine if the parameters are global (for all receive locations and send ports) or port specific.

4. Run the dynamic adapter registry file.

   For more information about this step, see Registering the Dynamic Sample Adapter.

5. Install the dynamic adapter.

   For more information about this step, see Installing the Dynamic Sample Adapter.

6. Modify the registry keys of the adapter.

   For more information about this step, see Modifying the Adapter Registration File.

7. Modify the adapter property pages to account for any new configuration parameters.

   To modify the handler configuration files, see Modifying the Adapter Handler Configuration Schemas. To modify the receive location or send port configuration files, see Modifying the Adapter Send Port and Receive Location Configuration Schemas.

8. Test the changes made to the adapter property pages.

For more information about this step, see Testing the Adapter Configuration XSD Files.

9. Create a custom user interface (UI) for the Add Adapter Wizard to select the schema to add to the project.

   For more information about this step, see Modifying the DisplayUI Method.

10. Modify sample code to return schemas as WSDL files.

    For more information about this step, see Modifying the GetServiceDescription Method.

11. Modify the existing WSDL file or create new WSDL files.

    For more information about this step, see Modifying the WSDL Files.

12. Modify the sample code to return any additional XSD files needed by the adapter that are not included in the WSDL.

    For more information about this step, see Modifying the GetSchema Method.

13. Rebuild and test the Add Adapter Wizard.

    For more information about this step, see Rebuilding and Testing the Adapter Management Project.

Registering the Dynamic Sample Adapter

Before modifying the dynamic sample adapter, register it with BizTalk Server.

To register the dynamic sample adapter

1. Complete the procedure to run the File Adapter sample in the SDK. For more information, see File Adapter (BizTalk Server Sample).
2. Click **Start**, point to **All Programs**, point to **Accessories**, and then click **Windows Explorer**.
3. Navigate to the installation drive for BizTalk Server 2006, and then navigate to **<drive>:\Program Files\Microsoft BizTalk Server 2006\SDK\Samples\Adapters\File Adapter**.
4. To add the sample adapter to the registry, double-click **DynamicAdapterManagement.reg**.

5. In the **Registry Editor** dialog box, click **Yes** to add the sample adapter to the registry, and then click **OK** to close the dialog box, verifying that the information was added to the registry.

6. To close Windows Explorer, on the **File** menu, click **Close**.

   The sample dynamic adapter is now registered with BizTalk Server.

   Continue to Installing the Dynamic Sample Adapter.

Installing the Dynamic Sample Adapter

After the sample dynamic adapter is registered with BizTalk Server, it needs to be added to the BizTalk Management database by using the BizTalk Administration console. After installing the adapter, restart the host instance.

To install the dynamic sample adapter

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.

2. In the BizTalk Administration console, double-click the **Microsoft BizTalk Server 2006 (Local)** node.

3. Right-click **Adapters**, click **New**, and then click **Adapter**.

4. In the **Add Adapter** dialog box, do the following.

Use this	To do this
Name	Type **Dynamic**.
Adapter	Select **Dynamic DotNetFile** from the drop-down list.
Comment	Type **Sample Dynamic Adapter**.

5. Click **OK**.

   The dynamic adapter now appears in the list of adapters in the right window of the BizTalk Administration console.

To stop and restart the host instance

1. Click **Start**, point to **Programs**, point to **Microsoft BizTalk Server 2006**, and then click **BizTalk Server Administration**.

2. In the BizTalk Administration console, double-click the **Microsoft BizTalk Server 2006 (local)** node, expand **Hosts**, and then select **BizTalkServerApplication** in the left pane.

3. In the results pane, right-click the host instance (typically, the computer name), and then click **Stop**.

   The status of the host instance changes to **Stopped**.

4. In the results pane, right-click the host instance, and then click **Start**.

   The status of the host instance changes to **Start pending**. You must click **Refresh**, or right-click the host instance and then click **Refresh**, to change the status to **Running**.

Modifying the DisplayUI Method

Modify the code within the **displayUI** method of the AdapterManagement.cs file to create a custom UI for selecting the type of schema to accept, and choose the appropriate WSDL file.

Creating Custom Pipeline Components

This section describes how to develop a pipeline component. You can create three types of pipeline components: general, assembling, and disassembling. Each of the three types can additionally implement probing functionality. Each type of pipeline component has an associated interface that must be implemented for the component to be plugged into the BizTalk Messaging Engine; the pipeline interfaces that distinguish the types of components are **IComponent**, **IAssemblerComponent**, and **IDisassemblerComponent**. For probing components, you must implement the **IProbeMessage** interface.

Microsoft® BizTalk® Server 2006 contains a sample pipeline component that you can reference when creating your own component. The sample component demonstrates how to append data to the end of a message and add data at the beginning of the message. For more information about the sample pipeline component, see CustomComponent (BizTalk Server Sample).

In This Section

- Using Pipeline Interfaces
- Developing a General Pipeline Component
- Developing an Assembling Pipeline Component
- Developing a Disassembling Pipeline Component

- Developing a Probing Pipeline Component
- Implementing a Seek Method in a Managed Streaming Pipeline Component
- Using the Parsing and Serializing Engines
- Reporting Errors from Pipeline Components
- Deploying Pipeline Components

Using Pipeline Interfaces

A pipeline component is a .NET or COM component that implements a set of predefined interfaces for interaction with the BizTalk Messaging Engine. Depending on the functionality of the component, different interfaces must be implemented. This topic discusses these interfaces and some of their methods.

IPipelineContext

All pipeline components can use **IPipelineContext** methods to access all document processing-specific interfaces. The **IPipelineContext** interface provides the following functionalities:

- Allows components to retrieve the ambient pipeline and stage settings.
- Allows components to retrieve message and message factories. With these factories, components can create various objects required for the execution of the component.
- Allows components to retrieve the document specifications. A document specification is an XSD schema plus additional annotations.

For more information about the **IPipelineContext** interface, see **IPipelineContext Interface**.

IBaseComponent

All pipeline components need to implement this interface to provide basic information about the component.

For more information about the **IBaseComponent** interface, see **IBaseComponent Interface**.

IComponent

All pipeline components except assemblers and disassemblers implement this interface to get messages from the BizTalk Server engine for processing and to pass processed messages back to the engine.

Execute. Method called by the engine to pass the input message to the component and retrieve the processed message from the component.

For more information about the **IComponent** interface, see **IComponent Interface**.

IPropertyBag, IPersistPropertyBag

Pipeline components need to implement **IPersistPropertyBag** to receive its configuration information. This interface and **IPropertyBag** are the standard interfaces. For more information about these interfaces, refer to the Microsoft .NET Framework Software Development Kit (SDK) documentation.

IDisassemblerComponent

A disassembling component is a pipeline component that receives one message on input and produces zero or more messages on output. Disassembling components are used to split interchanges of messages into individual documents. A disassembler component must implement the methods of the **IDisassemblerComponent** interface to get messages from BizTalk Server for processing and to pass disassembled documents back to BizTalk Server.

Method	Description
Disassemble	Performs the disassembling of the incoming document **pInMsg**.
GetNext	Gets the next message from the message set that resulted from disassembler execution. Returns **NULL** if there are no more messages.

For more information about **IDisassemblerComponent**, see **IDisassemblerComponent Interface**.

IAssemblerComponent

An assembling component is a pipeline component that receives several messages on input and produces one message on output. Assembling components are used to collect individual documents into the message interchange batch.

An assembler component implements the **IAssemblerComponent** methods that are called by the BizTalk Server engine at run time.

Method	Description
AddDocument	Adds the document **pInMsg** to the list of messages that will be included in the interchange.
Assemble	Builds the interchange from the messages that were added by the previous method. Returns a pointer to the assembled message.

For more information about **IAssemblerComponent**, see **IAssemblerComponent Interface**.

IProbeMessage

Any pipeline component (general, assembling, or disassembling) can implement **IProbeMessage** if it requires message probing functionality. A probing component is used in the pipeline stages that have **FirstMatch** execution mode. In such stages, BizTalk Server gives the message to the component, and the **Probe** method examines the beginning of the message to determine if the component recognizes the format of the message.

Method	Description
Probe	This method takes **pInMsg** message, and returns **True** if the format is recognized or **False** otherwise.

For more information about **IProbeMessage**, see **IProbeMessage Interface**.

INamedItem

This is a helper interface for accessing document schemas from managed and unmanaged code.

For more information about **INamedItem**, see **INamedItem**.

INamedItemList

This is a helper interface for accessing document schemas from managed and unmanaged code.

For more information about **INamedItemList**, see **INamedItemList Interface**.

IDocumentSpec

Pipeline components can use methods of the **IDocumentSpec** interface to perform document-specific actions, such as moving content properties to context and back, accessing document schemas, and so on.

Method	Description
DocType	Returns the type of the current document.
DocSpecName	Returns the specification name of the current document.
GetSchemaCollection	Returns the list of document schemas for the current

	document.
GetBodyPath	Returns the XPath to the node in the document where the body part begins.
GetDistinguishedPropertyAnnotationEnumerator	Returns a dictionary enumerator of all distinguished field property annotations.
GetPropertyAnnotationEnumerator	Returns an enumerator of all property annotations.

For more information about **IDocumentSpec**, see **IDocumentSpec Interface**.

IComponentUI

Pipeline components must implement this interface to be used within the Pipeline Designer environment.

Method	**Description**
Icon	Provides the icon that is associated with this component.
Validate	Pipeline Designer calls this method before pipeline compilation to verify that all the configuration properties are set correctly.

The **Icon** property returns an **IntPtr**. The following C# example shows how to return an **IntPtr**.

Developing a General Pipeline Component

A general pipeline component is a .NET or COM component that implements the following interfaces:

- **IBaseComponent Interface**
- **IComponent Interface**
- **IComponentUI Interface**
- **IPersistPropertyBag.** Refer to the .NET Framework SDK documentation for this interface.

A general pipeline component gets one message from the BizTalk Messaging Engine, processes it, and returns it to the BizTalk Server engine. General components can also be implemented so that they do not return messages to the server. Such components are called consuming components because the component receives messages but does not produce any result messages.

Developing an Assembling Pipeline Component

An assembling pipeline component is a .NET or COM component that receives several messages on input and produces one message on output. Assembling components are used to collect individual documents into the message interchange batch.

An assembling component must implement the following interfaces:

- **IBaseComponent**
- **IAssemblerComponent**
- **IComponentUI**
- **IPersistPropertyBag.** Refer to the .NET Framework SDK documentation for this interface.

Developing a Disassembling Pipeline Component

disassembling pipeline component receives one message on input and produces zero or more messages on output. Disassembling components are used to split interchanges of messages into individual documents. Disassembler components must implement the following interfaces:

- **IBaseComponent**
- **IDisassemblerComponent**
- **IComponentUI**
- **IPersistPropertyBag.** Refer to the .NET Framework SDK documentation for this interface.

You can create your own disassembling component by extending the **FFDasmComp** or **XMLDasmComp** class.

In This Section

- Handling Encoding in a Disassembler Pipeline Component
- Extending the Flat File Disassembler Pipeline Component

Handling Encoding in a Disassembler Pipeline Component

Ensure that your custom disassembler component encodes outbound documents in one of the following formats:

- UTF-8
- UTF-16
- UTF-32
- UTF-16LE
- UTF-16BE

The orchestration engine may not be able to process documents with other encoding formats.

UTF-32LE and UTF-32BE are not supported by the .NET Framework; to use these formats, you must create a custom encoding implementation.

Extending the Flat File Disassembler Pipeline Component

The following sample illustrates how to create a custom disassembler to parse flat file documents that are UTF-7 encoded. To process UTF-7 documents, the component inherits from the **FFDasmComp** class and then overrides its **GetDataReader** method.

Developing a Probing Pipeline Component

Any pipeline component (general, assembling, or disassembling) can implement the **IProbeMessage** interface if it must support message probing functionality. A probing component is used in the pipeline stages that have **FirstMatch** execution mode. In such stages, the BizTalk Messaging Engine gives the beginning part of the message to the component to determine if the component recognizes the format of the message. If the component recognizes the format, the entire message is given to the component for processing.

The **IProbeMessage** interface exposes a single method, **Probe**, which enables the component to check the beginning part of the message. The return value determines whether this component is run. The following steps outline how the BizTalk Messaging Engine runs a stage that requires recognition:

1. If the stage does not contain any components, the stage is not run and the message is given to the subsequent stages for processing.

2. Check if the component implements the **IProbeMessage** interface. If not, the Messaging Engine invokes the component. The stage processing is done and the message is given to the next stage.

3. The **Probe** method is invoked. If the return value is **True**, the component is run. Then the stage processing is done and the message is given to a next stage.

4. The Messaging Engine gets the next component in the stage. If there are no more components and none of the components have been run, it generates an error that pipeline processing has failed. If there are no more components and at least one component has been run, the processing is done.

If a stage does not require recognition (for example, the execution mode is **All**), the Messaging Engine invokes the component without first querying for the **IProbeMessage** interface and calling the **Probe** method.

Implementing a Seek Method in a Managed Streaming Pipeline Component

The native **IStream** interface does not provide a method to check the current stream position, so the messaging engine uses the following **Seek** method.

This method does not move the stream pointer; instead it queries the current position. So if you implement a pipeline component that works with a nonseekable stream.

Using the Parsing and Serializing Engines

Microsoft BizTalk Server 2006 includes a parsing and serializing engine to simplify the process of parsing and serializing flat file documents.

The parsing engine is a schema-driven framework that can parse many different document formats into XML. In addition, the engine has a well-defined extensibility model to make parsing custom formats even easier.

For complex parsing, disassemblers are used. In addition to converting the native format to XML, the disassembler can also separate a single document into multiple documents.

The serializing engine is similar to the parsing engine, except that it works in the opposite direction. Where the parsing engine converts native formats to XML, the serializing engine converts XML to native formats. And just as the parsing engine uses disassemblers, the serializing engine uses assemblers.

This section discusses how to perform character encoding, parse and serialize based on XML Schema definition language (XSD) schemas, and perform other common tasks using the parsing engine and serializing engine APIs.

In This Section

- Implementing Character Encoding in a Pipeline Component
- Using Schemas
- Using the Flat File Parsing Engine
- Using the Flat File Serializing Engine

Implementing Character Encoding in a Pipeline Component

To support custom character encoding, you must implement a custom encoding class by deriving from the Microsoft .NET Framework **Encoding** class, then create a custom flat file pipeline component by inheriting from the standard Flat File Disassembler or Flat File Assembler component. You can supply a new encoding instance to the parsing engine by overriding the protected virtual method **FFDasmComp.GetDataReader** as shown in the following example.

Using predefined encoding classes

The following encoding types are predefined by the Microsoft .NET Framework and can be used to construct the parser:

- ASCII
- UTF7
- UTF8
- Unicode (UTF16)

Using supported code pages

Use the following code to support Shift-JIS (codepage 932).

Using a private encoding class

You can create your own encoding class that derives from the **System.Text.Encoding** abstract class and perform your own encoding and decoding.

Using a private DataReader class

You can create your own **DataReader** class that implements the **IDataReader** interface and performs reading without creating any encoding classes.

Using Schemas

This section contains code examples for common tasks associated with using schemas.

Using XSD schemas

The **IDocumentSpec Interface** interface represents a document shape defined by an XML Schema definition language (XSD) schema; the shape is rooted by a top-level element of the XSD. After the schema is installed, it can be retrieved by calling the **IPipelineContext.GetDocumentSpecByType Method** or **IPipelineContext.GetDocumentSpecByName Method** methods in the **IPipelineContext** interface.

Using XSD flat file schemas

Both the **GetDocumentSpecByType** and **GetDocumentSpecByName** methods return the **IDocumentSpec** interface. If the schema is actually a flat file schema (one that has additional flat file-specific annotations), you can typecast the **IDocumentSpec** into **IFFDocumentSpec** and initiate parsing and serializing sequences from there.

Using the Flat File Parsing Engine

This section contains code examples for common tasks associated with the flat file parsing engine.

In This Section

- Invoking the Flat File Parsing Engine
- Customizing Parsing Engine Errors
- Wrapping the Parsing Engine
- Parsing Modes
- Parsing Escape Characters
- Parsing Flat File Schemas with Positional Records

Invoking the Flat File Parsing Engine

The flat file parsing engine can be invoked by calling the **Parse** method of the **IFFDocumentSpec** interface, which in turn is typecast from the **IDocumentSpec Interface** retrieved from the **PipelineContext**. Because the **XmlReader** returned from the **Parse** method allows clients to retrieve one node at a time, this is a good opportunity to customize the parser.

Customizing Parsing Engine Errors

You can register your own error-handling callback with the parsing engine to handle errors.

Wrapping the Parsing Engine

You can create your own **XmlReader** that embeds the reader returned by the **Parse** method and provides customization**.**

Parsing Modes

The parsing mode is an attribute on the schemaInfo record, with two modes: speed and complexity.

Note These properties are not available to edit within the BizTalk Schema Editor. You must edit the schema directly using any text editor.

In speed mode, the parser tries to fit data as they appear in the stream. For example, given the following schema.

In complexity mode, the flat file parsing engine uses both top-down and bottom-up parsing, and tries to fit data more accurately. In speed mode, the parser tries to fit data as they appear in the stream.

If you have optional elements with required elements

Parsing Escape Characters

When the parser encounters an escape character that prefixes a regular character (that is, one that is not a delimiter or other special character), the escape character is ignored. For example, given a string "abc\d" where "\" is the escape character, the output is "abcd".

If the parser encounters a double escape character (for example, "abc\\d"), the output includes a single escape character ("abc\d").

If the parser encounters three escape characters (for example, abc\\\d), the output is "abc\d" because the first two escape characters are parsed to "\" and the third escape character is ignored.

The parser treats misplaced delimiters as regular characters. For example, if "Record, Field1, Field,2" is received, the output XML is <Field1> <Field,2>.

Parsing Flat File Schemas with Positional Records

When parsing a flat file schema with positional records of unequal size, you must include a tag within each schema record or the trailer to indicate the size of each positional record. Otherwise, the parsing engine returns the longest record size.

Using the Flat File Serializing Engine

The flat file serializing engine is invoked by calling the **Serialize** method of the **IFFDocumentSpec** interface. By providing a customized **XmlReader** as the argument to the method, you can control the final data that gets serialized.

Reporting Errors from Pipeline Components

Pipeline components report errors in two ways:

- For .NET-based components, by throwing an exception.
- For COM-based components, by setting the **ErrorInfo** object and returning a failure HRESULT.

Reporting errors from .NET pipeline components

To report an error, a .NET-based pipeline component needs to throw an exception where it reports the error description. To report the name of the component that throws an error, set the **Source** property of the **Exception** object.

The Messaging Engine uses the **Message** and **Source** properties of the **Exception** object to report an error. The following message is written to the event log:

"There was a failure executing the [receive|send] pipeline: <pipeline name> Source: <Source> [Receive Location|Send Port:] <location|port name> Reason: <Message>."

Reporting errors from COM pipeline components

To report an error, COM-based pipeline components perform the following actions:

1. The pipeline component sets the **IErrorInfo** object by calling the **SetErrorInfo** method.

2. The pipeline component returns a failed HRESULT to the Messaging Engine.

The Messaging Engine uses the **GetSource** and **GetDescription** properties of the **IErrorInfo** object to report an error. If the source is not set, the name of the component is used. If the description is not set or the whole **ErrorInfo** object is not set, the returned HRESULT is reported instead of the description. The following message is written to the event log:

"There was a failure executing the [receive|send] pipeline: <pipeline name> Source: <GetSource> [Receive Location|Send Port:] <location|port name> Reason: <GetDescription or HRESULT>."

Deploying Pipeline Components

All the .NET pipeline component assemblies (native and custom) must be located in the <*installation directory*>\Pipeline Components folder to be executed by the server. If the pipeline with a custom component will be deployed across several servers, the component's binaries must be present in the specified folder on every server.

You must add a custom pipeline component to be used by the BizTalk Runtime to the Global Assembly Cache (GAC). Note that this is not necessary for a custom pipeline component to be used by BizTalk design-time components.

Custom COM components in the pipeline will also appear in the Toolbox, provided they are registered on the computer as a COM component. Custom .NET pipeline components must be placed into the <*installation directory*>\Pipeline Components folder.

After the binary files are in the correct location, you need to add the component to the Toolbox. For instructions on adding the pipeline component to the Toolbox.

Creating Custom Functoids

Although BizTalk Server provides many functoids to support a range of diverse operations, you will likely encounter a situation that requires a different approach. Custom functoids provide a way for you to extend the range of operations available within the BizTalk Server mapping environment. Each custom functoid is deployed as a .NET assembly using classes derived from **Microsoft.BizTalk.BaseFunctoid**. An assembly can contain more than one custom functoid.

You should consider using a custom functoid in the following scenarios:

- You have special validation and conversion rules for a character code field using data that is only accessible through a proprietary legacy API.
- You need to encrypt or decrypt fields using custom business logic and key management.

- You need to generate a hash code from part of the message for use in another application.
- Accounting requests that messages transmitted to their department include summary information about total sales by each product type.
- You want to reduce the complexity of a map by combining several related steps, by using a different approach, or by using new class libraries.
- More than one map is using the same script code in a script functoid.
- You need to write to the event log when an operation fails.

Custom functoids can be integrated into a solution directly by using inline code or indirectly by reference to a method in a class library deployed into the global assembly cache. Both types of integration rely on the **BizTalk.BaseFunctoids** class and follow the same set of general steps:

1. Create a new class library project using the .NET language of your choice.
2. Using the strong-naming utility sn.exe, create a keyfile and assign it to the project.
3. Add a reference to Microsoft.BizTalk.BaseFunctoids.dll. This assembly contains the **BaseFunctoids** base class.
4. Create a resource file and add it to the project. Add string resources for the functoid name, tooltip, and description. Add a 16x16-pixel image resource to represent the functoid on the map designer palette.
5. Implement the factoid class by deriving from **BaseFunctoid**, establishing basic parameters in the constructor, and then writing the functoid method and any supporting methods. The assembly can contain multiple custom functoids.
6. Deploy the assembly and ensure the new functoid is available from the Toolbox palette. See Adding and Removing Custom Functoids from the Visual Studio 2005 Toolbox.

In This Section

This section contains:

- Using BaseFunctoid
- Developing a Custom Referenced Functoid
- Developing a Custom Inline Functoid

- Developing a Custom Cumulative Functoid
- Adding and Removing Custom Functoids from the Visual Studio 2005 Toolbox

Using BaseFunctoid

All custom functoids must derive from the **BaseFunctoid** class. You must first override the constructor and make a set of calls that tell BizTalk Mapper about your custom functoid. Then you need to write the functoid logic.

Overriding the Constructor

You must perform a number of tasks in the class constructor override method to characterize your functoid. These tasks are in addition to any functoid-specific code your solution requires. The following table describes the primary tasks.

Task	**Use these methods or properties**	**Comments**
Assign a unique ID to the functoid	**ID**	Use a value greater than 6000 that has not been used. Values less than 6000 are reserved for use by internal functoids.
Indicate whether the functoid has side effects	**HasSideEffect**	Used by the mapper to optimize the XSLT code that is generated. This property is true by default.
Point to the resource assembly	**SetupResourceAssembly**	Include a resource file with your project. If building with Visual Studio, the resource assembly must be **ProjectName.ResourceName**.
Enable the custom functoid to appear in the BizTalk Mapper palette	**SetName** **SetTooltip** **SetDescription** **SetBitmap**	Use a resource ID pointing to a string for the name, tooltip and description; use a 16x16-pixel bitmap.
Assign the functoid to one or more categories	**Category**	Categorize the functoid by using one or more FunctoidCategory values.
Specify the number of	**SetMinParams**	Use the **SetMinParams** method to set the number of required

parameters accepted	**SetMaxParams** **HasVariableInputs**	parameters and the **SetMaxParams** method to set the number of optional parameters. Use the following guidelines to set these values: • If you have no optional parameters, set min = max. • If you have some optional parameters, set max = (number of optional parameters - min number of parameters). • If you want to allow unlimited optional parameters, do not set max. • If you have a variable number of inputs, do not set min or max, and set **HasVariableInputs** = **true**.
Declare what can connect to your functoid	**AddInputConnectionType**	Call **AddInputConnectionType** once for each ConnectionType that the functoid supports.
Declare what your functoid can connect to	**OutputConnectionType**	Use values from **ConnectionType** to tell BizTalk Mapper the types of objects that can receive output from your functoid. Use **OR** to specify multiple connection types.
Tell BizTalk Server which methods to invoke for your functoid	**SetExternalFunctionName** **SetExternalFunctionName2** **SetExternalFunctionName3**	For cumulative functoids, use **SetExternalFunctionName** to set the initialization function, **SetExternalFunctionName2** to set the accumulation function, and **SetExternalFunctionName3** to specify the function that returns the accumulated value. For noncumulative functoids use **SetExternalFunctionName** to set the functoid method.
Have BizTalk	**AddScriptTypeSupport**	Call **AddScriptTypeSupport** with ScriptType to enable inline code.

Server use inline code to invoke your functoid	**SetScriptBuffer**	Invoke **SetScriptBuffer** to pass in the code for the functoid. This code will be copied into the map.
Declare global variables for an inline functoid	**SetScriptGlobalBuffer**	Any declarations made will be visible to other inline scripts included in the map.
Indicate which helper functions your inline functoid requires	**RequiredGlobalHelperFunctions**	Use values from the **InlineGlobalHelperFunction** enumeration to specify which helper functions are required. Use **OR** to specify multiple helper functions.
Validate parameters passed to your functoid	**IsDate** **IsNumeric**	These functions provide a true/false answer without throwing an exception.

Implementing Functoid Logic

To make the functoid useful you must implement one or more methods depending upon the functoid category. If the functoid is cumulative, then you need to supply three methods—one for initialization, one to do the accumulation, and one to return the accumulated value. If the functoid is not cumulative, then you need to supply one method that returns a value.

You also have to decide if the functoid implementation code should be copied inline into the map or kept within a compiled .NET assembly and used through a reference.

Consider using an inline functoid when:

- It is okay for others to read and potentially modify your business logic.
- Your functoid depends only upon .NET Framework namespaces that the map supports. For available namespaces, see Scripting Using Inline C#, JScript .NET, and Visual Basic .NET .
- You do not want to deploy and maintain another assembly with your BizTalk solution.

- You are writing a series of functoids that share variables.

Consider using a referenced functoid when:

- You do not want your business logic copied into the map where it might be seen or modified by others.
- Your functoid depends upon .NET Framework classes that the map does not support.
- The added functionality provided by the .NET Framework justifies deploying and maintaining another assembly with your BizTalk solution.

Developing a Custom Referenced Functoid

Custom referenced functoids do not copy implementation code inline into the map. Instead, a reference to the assembly, class, and method is placed in the extension object file associated with the generated style sheet and called at run time.

Developing a Custom Inline Functoid

Custom inline functoids provide functionality by copying implementation code directly into a map and not by referencing an assembly, class, and method name like a custom referenced functoid.

Building Inline Script

There are two ways to provide script for inclusion into the map. Choose from the following methods, based on whether your custom functoid supports a variable number of parameters:

- Override **GetInlineScriptBuffer** when your custom functoid accepts a variable number of input parameters and you have set the **HasVariableInputs** property to **true**. For example, use this method if you want to concatenate a variable number of strings or find the largest value in a set of values.
- Use **SetScriptBuffer** when you do not need to support a variable number of input parameters. You can still use optional parameters, but the total number of parameters is fixed.

These two methods require different implementations.

Providing Inline Code with SetScriptBuffer

To configure your custom functoid to use inline script:

1. Call **AddScriptTypeSupport** with ScriptType to enable inline code and set the supported script type.

2. Invoke **SetScriptBuffer** to set the code to use for the custom functoid. You will call this function three times with the *functionNumber* parameter for custom cumulative functoids and once for custom noncumulative functoids.

3. Use **SetScriptGlobalBuffer** to declare any global variables that your inline code uses.

4. Use **RequiredGlobalHelperFunctions** to indicate the helper functions that your custom inline functoid requires.

You can build your script by using StringBuilder or constants. One approach to writing script code is to write a custom referenced functoid first and, when all bugs are eliminated, convert it to inline by copying your functions into string constants.

Providing Inline Code with GetInlineScriptBuffer

If your custom inline functoid supports a variable number of parameters, you will override **GetInlineScriptBuffer**. To configure your custom functoid to use inline script:

1. In the constructor, declare that your custom functoid has variable inputs by setting **HasVariableInputs** to **true**.

2. In the constructor, call **AddScriptTypeSupport** with **ScriptType** to enable inline code and set the supported script type.

3. Override **GetInlineScriptBuffer** to construct and return the code to use in the map for your custom functoid. Use the parameters to build the correct code by checking the *scriptType* and *numParams*. The final parameter, *functionNumber*, should be 0. This is because cumulative functions have a fixed number of inputs and do not use this mechanism.

4. Use **SetScriptGlobalBuffer** to declare global variables that your inline code uses.

5. Use **RequiredGlobalHelperFunctions** to indicate the helper functions that your custom inline functoid requires.

Testing an Inline Script

Testing is an important consideration in any development effort. Custom inline functoids can be challenging to test. To simplify the process, use one or both of the following techniques:

- Examine the XSLT of a map that uses the custom inline functoid.
- Verify the input and output of a map that uses the custom inline functoid.

Examine the XSLT of a Map That Uses the Custom Inline Functoid

This technique often reveals problems with logic or subtle syntax issues. It also helps you to understand what happens in the map.

To view the XSLT for a map:

1. From a Visual Studio BizTalk project, click the **Solution Explorer** tab, right-click a map that uses your custom inline functoid, and then click **Validate Map**.

2. Scroll the Output window to find the URL for the XSLT file. Press CTRL and click the URL to view the file.

Test a Map That Uses the Custom Inline Functoid

This tests whether the map and custom inline functoid work as expected.

To test a map:

1. From a Visual Studio BizTalk project, click the **Solution Explorer** tab, right-click a map that uses your custom inline functoid, and then click **Test Map**.

2. Scroll the Output window to find the URL for the output file. Press CTRL and click the URL to view the file.

You can check input and output values to verify that the map behaved as expected

Developing a Custom Cumulative Functoid

Use a custom cumulative functoid to perform accumulation operations for values that occur multiple times within an instance message.

You must implement three functions when developing a cumulative functoid. The three functions correspond to the initialize, cumulate, and get actions that the map needs to perform the accumulation. Before discussing these functions it is important to discuss thread safety.

Writing a Thread-Safe Functoid

The functoid code must be thread-safe because under stress conditions multiple instances of a map may be running concurrently. Some of the points to remember include:

- Static state must be thread-safe.

- Instance state does not always need to be thread-safe.

- Design with consideration for running under high-stress conditions. Avoid taking locks whenever possible.
- Avoid the need for synchronization if possible.

BizTalk Server provides a simple mechanism to reduce the complexity of writing a thread-safe cumulative functoid. All three functions have the same first parameter, an integer index value. BizTalk Server assigns a unique number to the index value when it calls your initialization function. You can use this value as an index into an array that holds accumulating values, as shown in the following code:

This example uses a HashTable instead of an ArrayList. This is because the initialization function might not be called with in-order index values.

Implementing the Three Cumulative Functions

You will need to implement three functions for each custom cumulative functoid you develop. The functions and the methods you must call in the constructor to set them are summarized in the following table. All of the functions return string values.

Function Purpose	Arguments	To set a reference	To set inline script
Initialization	**int index**	**SetExternalFunctionName**	**SetScriptBuffer** with *functionNumber* = 0
Cumulation	**int index, string val, string scope**	**SetExternalFunctionName2**	**SetScriptBuffer** with *functionNumber* = 1
Get	**int index**	**SetExternalFunctionName3**	**SetScriptBuffer** with *functionNumber* = 2

Initialization

Initialization enables you to prepare the mechanisms you will use to perform accumulation. You can initialize an array, reset one or more values, or load other resources as needed. The string return value is not used.

Cumulation

This is where you perform the accumulation operation appropriate for your functoid. BizTalk Server passes in the following three parameters:

- **Index.** An integer value representing a map instance. There may be multiple map instances running concurrently.
- **Val.** A string containing the value that should be accumulated. Unless you are writing a string Cumulate functoid it is a numeric value.

- **Scope.** A string containing a number indicating which element or attribute value should be accumulated. Actual values are determined by an implementation.

You decide which values to accumulate and which values to ignore. For example, you may ignore values that are not below 0 but throw an exception when a value is not numeric. **BaseFunctoid** supplies two functions—**IsDate** and **IsNumeric**—to assist with validation.

The string return value is not used.

Get

When BizTalk Server finishes iterating through all of the values determined by the functoid settings in the map, it requests the accumulated value. The get function has one argument, *Index*, which is an integer value representing a map instance. Your function should use the index value to find and return the accumulated value as a string.

Adding and Removing Custom Functoids from the Visual Studio 2005 Toolbox

This topic describes how to add custom functoids to and remove custom functoids from the Visual Studio 2005 Toolbox.

Adding Custom Functoids to Visual Studio

Custom functoids must be added to the Visual Studio Toolbox before they can be used in a map. Use the following procedure to add custom functoids.

To add a custom functoid

1. Add the functoid to the Visual Studio Toolbox.
    a. Using Windows Explorer, find the assembly that implements your custom functoids.
    b. Copy the assembly to the *<BizTalk Server installation folder>***\Developer Tools\Mapper Extensions** directory. This is where BizTalk Mapper looks for custom functoids.
    c. From a Visual Studio BizTalk project, on the **Tools** menu, click **Choose Toolbox Items**.
    d. In the **Choose Toolbox items** dialog box, click the **Functoids** tab.
    e. Click **Reset**, and then click **OK**. This process may take a few moments.

Your custom functoids should now appear in the Toolbox under tabs matching their category.

- OR -

f. From a Visual Studio BizTalk project, on the **Tools** menu, click **Choose Toolbox Items**.

g. In the **Choose Toolbox items** dialog box, click the **Functoids** tab.

h. Click **Reset**, and then click **OK**.

i. On the **File** menu, click **Exit** to close Visual Studio.

j. Click **Start**, point to **Programs**, point to **Microsoft Visual Studio 2005**, point to **Visual Studio Tools**, and then click **Visual Studio 2005 Command Prompt**.

k. At the command prompt, type **devenv /setup**.

l. Click **Start**, point to **Programs**, point to **Microsoft Visual Studio 2005**, and then click **Microsoft Visual Studio 2005**.

The custom functoid(s) should appear in the appropriate tab.

2. Add the assembly to the global assembly cache. If your assembly contains only inline functoids, then you can skip this step.

a. Click **Start**, point to **Programs**, point to **Microsoft Visual Studio 2005**, point to **Visual Studio Tools**, and then click **Visual Studio 2005 Command Prompt**.

b. Switch to the folder containing your assembly.

c. At the command prompt, type **gacutil /if <assembly_path >**. For example, if your assembly name is FunctoidLibrary.dll, then type **gacutil /if FunctoidLibrary.dll**.

d. When you are finished, type **exit**.

Removing Custom Functoids from Visual Studio

Use the following procedure to remove custom functoids.

To remove a custom functoid

1. Remove the functoid from the Visual Studio Toolbox.

a. From a Visual Studio BizTalk project, on the **Tools** menu, click **Choose Toolbox Items**.

b. In the **Choose Toolbox items** dialog box, click the **Functoids** tab.

c. Find the custom functoid in the list, select the **Remove** check box, and then click **OK**.

- OR -

d. While editing a map in a Visual Studio BizTalk project, click the **Toolbox** tab to bring up the Toolbox Palette.

e. Click the functoid group containing your custom functoid.

f. Right-click the functoid you want to remove, and then click **Delete** or press the delete key.

2. Remove the functoid assembly from the **Developer Tools\Mapper Extensions** directory.

   a. Start Windows Explorer and navigate to the **Developer Tools\Mapper Extensions** directory of BizTalk Server.

   b. Right-click the assembly containing the removed functoid, and then click **Delete** to remove the file.

3. Remove the functoid assembly from the global assembly cache. If your assembly contains only inline functoids, then you can skip this step.

   a. Click **Start**, point to **Programs**, point to **Microsoft Visual Studio 2005**, point to **Visual Studio Tools**, and then click **Visual Studio 2005 Command Prompt**.

   b. At the command prompt, type **gacutil /u <assembly_display_name>**. For example, if your assembly name is FunctoidLibrary.dll, then type **gacutil /if FunctoidLibrary**.

   c. When you are finished, type **exit**.

Utilities in the SDK

This section provides instructions for using several useful utilities included in the Microsoft® BizTalk® Server 2006 Software Development Kit (SDK).

In This Section

- Adapter Registration Wizard
- BizTalk Message Queuing Large Message Extension
- **Pipeline Toolsc**
- How to Extend the Schema Generator Wizard
- Subscription Viewer
- Replace Public Key Token Utility

Adapter Registration Wizard

You use the Adapter Registration Wizard to create the registry files needed to configure and register a custom adapter.

Location in SDK

<Installation Path>\SDK\Utilities\AdapterWizard\

⊟To Run This Utility

Start the wizard by running the AdapterWizard.exe executable. The pages that follow prompt you for information about your adapter and the properties that it supports. The individual Adapter Registration Wizard pages are described in the sections that follow.

Generic Adapter Properties page

Use the Generic Adapter Properties page to specify adapter properties.

Use this	To do this
Transport adapter type	Specify the adapter type.
Property namespace	Specify the adapter namespace. BizTalk Server 2006 stores adapter-specific properties on the message context in this namespace.
Adapter can receive messages and submit them to BizTalk Server	Identify communication patterns supported by the adapter. Used to calculate the Constraints registry entry for the adapter.
Adapter can send messages	Identify communication patterns supported by the

from BizTalk Server	adapter. Used to calculate the Constraints registry entry for the adapter.

Inbound Part page

Use the Inbound Part page to specify inbound receiving logic for your adapter.

Use this	To do this
Browse	Select the assembly that implements the inbound receiving logic for your adapter. A list of the public types exposed by this assembly appear in the drop-down list. Select the type within the specified assembly that implements the inbound logic for this adapter from the drop-down list.
Adapter supports request-response protocol	Specify the receive capabilities of your adapter. Used to calculate the Constraints registry entry for your adapter.
Adapter is isolated (that is hosted in a different process)	Specify the receive capabilities of your adapter. Used to calculate the Constraints registry entry for your adapter.

Outbound Part page

Use the Outbound Part page to specify outbound receiving logic for your adapter.

Use this	To do this
Browse	Select the assembly that implements the outbound receiving logic for your adapter. A list of the public types exposed by this assembly appears in the drop-down list . Select the type for the specified assembly that implements the outbound logic for this adapter from the drop-down list. You must also enter a comma-separated list of aliases used to identify the protocol used by this adapter (for example, submit://).
Adapter supports solicit-response protocol	Identify the transmitting capabilities of your adapter. These values are used to calculate the Constraints registry entry for your adapter.

Adapter Management page

Use the Adapter Management page to specify design-time management logic for your adapter.

Use this	To do this
Browse	Select the assembly that implements the design-time adapter management logic for your adapter. A list of the public types exposed by this assembly appears in the drop-down list. Select the type for the specified assembly that implements the adapter management logic for this adapter from the drop-down list.

Output File name

Use the Output file name page to specify an output file name and location.

Use this	To do this
Browse	Choose an output file name and location. The adapter registry is written to this file.

Remarks

The Adapter Registration Wizard utility populates the Enterprise Single Sign-On (SSO) configuration store properties with default values. If your adapter does not use the standard property pages provided by the Adapter Framework for handler and location adapter properties, you must edit the registry file manually to modify these values. For more information about these settings, see "Registration of adapter properties for SSO configuration store" in the topic Adapter Registration.

BizTalk Message Queuing Large Message Extension

Native message queuing cannot process a message with a body larger than 4 megabytes (MB). However, Microsoft® BizTalk® Server 2006 includes an add-on for native message queuing that permits processing messages larger than 4 MB. This add-on is delivered as the Mqrtlarge.dll file, and exposes the **MQSendLargeMessage** and **MQReceiveLargeMessage** application programming interfaces (APIs), and the analogous COM model. These functions are implemented as standard message queuing APIs, **MQSendMessage** and **MQReceiveMessage** respectively.

To participate in large message exchanges, the message queuing computer must have the Mqrtlarge.dll file installed, and the message queuing application should use the add-on APIs. Otherwise, complete messages will be fragmented.

Location in SDK

<*Installation Path*>\SDK\Utilities\Mqrtlarge.dll

File Inventory

The following table shows the file this utility uses and describes its purpose.

File(s)	Description
Mqrtlarge.dll	A Win32® dynamic-link library that exposes **MQSendLargeMessage** and **MQReceiveLargeMessage**. The header files are located in the <*Installation Path*>\SDK\Include directory.

Using This Utility

Use the following procedure to run the Mqrtlarge.dll file.

To use the Mqrtlarge.dll file

1. Add the file Mqrtlarge.dll to the computer that does not contain an installation of BizTalk Server. Message Queuing uses Mqrtlarge.dll to send or receive messages to or from BizTalk Server.

2. To access the APIs in the Mqrtlarge.dll file, add a reference to the Mqrtlarge.dll file in the applications that send or receive messages to or from other computers to the BizTalk Message Queuing endpoint.

Remarks

Large messages are sent to standard Message Queuing (also known as MSMQ) queues. You should use only the large message APIs on such queues, although this is not enforced.

You can still use **MQSendMessage** to send messages to a queue to which you sent large messages. Likewise, you can use **MQReceiveMessage** to receive messages from such queues, although it is not recommended because it may impact the performance of the **MQReceiveLargeMessage** API.

For recovery purposes, it is recommended that you use **DEAD_LETTER** journaling.

Fragmentation

In general, a large message will be fragmented into numerous messages, each smaller than 4 MB. It is important to understand that only the body is fragmented. The rest of the properties are sent with each fragment.

The number of fragments is defined by the size of the message and the size of the fragments. The fragment size may be automatically determined by the file

Mqrtlarge.dll or the appropriate part of BizTalk Message Queuing. Applications can also explicitly specify the fragment size.

Note that the large message extension requires extending the message by adding additional internal data of a constant size.

PROPID_M_EXTENSION

You can use the **PROPID_M_EXTENSION/PROPID_M_EXTENSION_LEN** properties to sign the message. The signature comprises 40 bytes (B). The following table shows signature fragments.

Description	**Size**
Globally unique identifier (GUID) denoting that this message was sent using **MQSendLargeMessage**	16 B
GUID for the message: unique per-large message ID that is common to all parts of the message	16 B
Total size of the large message	4 B
Part number	2 B
Number of message parts	2 B

The decision to use **PROPID_M_EXTENSION** has an additional implication. The Microsoft Host Integration Server MSMQ-MQSeries Bridge uses this property to pass a structure that contains properties that are not mapped directly between MQSeries and Message Queuing messages. Applications that send messages to and from MQSeries explicitly or implicitly use this structure. Message Queuing can generate acknowledgments when a message has been received from a queue by a receiving application. The acknowledgments are sent to a queue specified by the sending application. In the case of large messages, you send this acknowledgment only for the last part of the large message.

Pipeline Tools

The pipeline tools supplied with the Microsoft® BizTalk® Server 2006 Software Development Kit (SDK) enable you to verify that a pipeline is functioning correctly without having to configure the BizTalk Server environment, such as send/receive ports. You can also use the pipeline tools to:

- Debug third-party pipeline components outside of the server environment.
- Diagnose parsing engine error messages.

- Experiment with different schemas without the burden of compiling, deploying, undeploying, and recompiling.
- Explore flat file and XML assembler/disassembler behavior.
- View disassembler output and discover which message context properties are promoted and their values.
- Run send/receive pipelines without disassembler and assembler components (for example, you can view the output of the S/MIME decoder).
- Make fine-grained performance measurements of the pipeline alone (rather than the entire messaging subsystem).

Location in SDK

<*Installation Path*>\SDK\Utilities\PipelineTools

You use pipeline tools for executing, debugging, and profiling pipelines, and pipeline components (that is, flat file and XML assembler/disassembler components).

File Inventory

The following table lists the pipeline tools files and describes their purpose.

File(s)	**Description**
DSDump.exe	Enables you to dump the document schema structure, which is an in-memory lightweight representation of the one or more XSD schemas, with or without flat file annotations. This tool can be helpful when you get parsing engine errors such as $Root$0$3$2 and you need to decode them. Numbers after $ mean 0-based index or records as they appear in the document schema.
FFAsm.exe	Runs the flat file assembler component, directly invoking it by emulating a send pipeline to enable you to see how it serializes or assembles a user's XML document(s) into a flat file document.
FFDasm.exe	Runs the flat file disassembler component, directly invoking it by emulating a receive pipeline to enable you to see how it parses or disassembles a user's flat file document into one or more XML documents.
Pipeline.exe	Runs a send or receive pipeline; accepts one or more input documents and their parts, XSD schemas and related information; and produces an output document after the pipeline runs. Pipeline.exe does not access BizTalk Server databases, so pipelines containing BizTalk Framework assembler and disassembler components that access BizTalk Server 2006 databases during execution may not be

	supported.
XMLAsm.exe	Runs the XML assembler component, directly invoking it by emulating a send pipeline to enable you to see how it serializes, assembles, or envelopes a user's XML document(s) into an output XML document.
XMLDasm.exe	Runs the XML disassembler component, directly invoking it by emulating a receive pipeline to enable you to see how it parses, disassembles, or un-envelopes a user's XML document into one or more XML documents.

Usage

A more detailed description of each file is provided in the sections that follow.

DSDump.exe

DSDump.exe enables you to dump the document schema structure, which is an in-memory lightweight representation of one or more XSD schemas, with or without flat file annotations.

DSDump.exe (document schema dump tool) supports the following command-line parameter:

In case of success, DSDump prints the document schema to the console.

FFAsm.exe

FFAsm.exe (flat file assembler) supports the following command-line parameters:

For example, if you want to assemble three input XML documents into a single flat file document with a header and property demotion, use the following command:

FFDasm.exe

FFDasm.exe (flat file disassembler) supports the following command-line parameters:

For example, if you want to disassemble a flat file document that has a header, body, and trailer, and display the results to the console, use the following command:

Pipeline.exe

Pipeline.exe (the Pipeline tool) supports the following command-line parameters:

For example, if you want to run a receive pipeline from the file ReceivePipeline.btp (which has XML disassembler and XML validator components) using mySchema.xsd and display the results to the console, use the following command:

- Or -

In the first example, a qualified type name (**MyProject.MySchema**) for the schema is included as a command-line argument. In the second example, the schema is obtained from the specified BizTalk project file.

You can also run pipelines from the compiled BizTalk Server 2006 project files, as in the following example (the pipeline is referenced by its assembly-qualified type name):

In the following example, a pipeline is referenced by its type name and assembly file name separately:

The following example shows a pipeline referenced by its assembly name:

Pipeline.exe runs with whatever credentials you have used to launch it. It does not use the account that normal BizTalk Server 2006 host instances run under, so you may not be able to run pipelines that contain components that require database access. Make sure you run Pipeline.exe under an account that has all the required privileges.

XMLAsm.exe

XMLAsm.exe (XML Assembler tool) supports the following command-line parameters:

For example, if you want to assemble two input XML documents into a single XML document with an envelope and display the results to the console, use the following command:

XMLDasm.exe

XMLDasm.exe supports the following command-line parameters:

For example, if you want to disassemble an XML document with two nested envelopes and display the results to the console, use the following command:

How to Extend the Schema Generator Wizard

This topic explains how to extend the existing Schema Generator Wizard and how to create a new wizard for schema generation.

To extend the existing schema wizard

1. Implement the **ISchemaGenerator** interface to create a new schema generator module that you can integrate into the existing Schema Generator Wizard.

2. Drop the resulting assembly in the following Microsoft® BizTalk® Server 2006 installation folder:

   <*Installation folder*>\Developer Tools\Schema Editor Extensions

   The next time you run the Schema Generator Wizard, it will automatically pick up your new schema generator module.

Use the following procedure to create a new schema wizard.

Location in SDK

<*Installation Path*>\SDK\Utilities\Schema Generator

To create a new schema wizard

1. Run InstallDTD.vbs to install Microsoft.BizTalk.DTDToXSDGenerator.dll to <*Installation Path*>\Developer Tools\Schema Editor Extensions. DTDToXSDGenerator.dll exposes classes you can use to convert DTD files to XSD.

2. Run InstallWFX.vbs to install Microsoft.BizTalk.WFXToXSDGenerator.dll to <*Installation Path*>\Developer Tools\Schema Editor Extensions. WFXToXSDGenerator.dll exposes classes you can use to convert WFX files to XSD.

Subscription Viewer

The Subscription Viewer utility shows the types of messages and the destinations to which they are routed. This is useful for determining the activation subscription for an orchestration.

Location in SDK

<*Installation Path*>\SDK\Utilities\BTSSubscriptionViewer.exe

To run this utility

1. To run the subscription viewer, double-click BTSSubscriptionViewer.exe.

2. For each subscription, the following information is displayed:

- Name
- Subscription ID
- Service Type
- Service ID
- Port ID
- Instance ID
- Application Name
- Group ID
- Group
- Enabled
- Req/Resp

Replace Public Key Token Utility

This utility replaces a public key token in a file with a new public key token. This utility comprises three files:

- **ReplacePKT.bat.** A batch file that you can run to replace all existing occurrences of a public key token in a file with a new public key token, or replace the key value in a public key token (.snk) file with the environment variable **NEWTOKEN**.

- **ReplacePKT.vbs.** A script file that ...

- **ReplacePKT.wsf.** A script file that ...

Location in SDK

%ProgramFiles%\Microsoft BizTalk Server 2006\SDK\Utilities\ReplacePublicKeyToken\

To Run This Utility

To replace all instances of a public key token in a specified file, do this:

1. Open ReplacePKT.bat in a text editor and specify the following variables in the file:

- <.snk file>: Replace this text with the file name of the .snk file containing the public key token that you want substitute for the existing public key token.
- [<old public key token>]: Replace this text with the public key token that you want to replace.
- [<file to replace>]: Replace this text with

To replace a public key token in an .snk file with the environment variable **NEWTOKEN**, do this:

1. It can be pointed to a public key token (.snk) file only. In this case, it will extract the key value and set it in the environment variable. This functionality can be used to replace public keys in the invoking script file.

Deployment

This section provides information about deploying the BizTalk Server 2006 infrastructure. For information about deploying BizTalk applications, see Deploying BizTalk Assemblies from Visual Studio into a BizTalk Application and Deploying BizTalk Applications.

In This Section

- Best Practices: BizTalk Server 2006 Deployment
- Installing and Configuring BizTalk Server 2006
- Upgrading to BizTalk Server 2006
- Migrating Artifacts from a Previous Version of BizTalk Server
- Interoperating with Previous Versions of BizTalk Server
- Deploying the Biztalk Server 2006 Management Pack for MOM
- Configuring BizTalk Server 2006 in a Cluster
- Configuring Network Load Balancing
- Securing Your Deployment of BizTalk Server 2006
- **Setting up a Development and Test Environment**

Best Practices: BizTalk Server 2006 Deployment

This section describes best practices to follow when deploying BizTalk Server 2006. For information on installing and configuring BizTalk Server 2006, see BizTalk Server 2006 Installation Guide.

For information about planning and securing your deployment of BizTalk Server 2006, see Planning and Architecture .

Deploy Internet Information Services 6.0 to run Web services that use unicode file names

Internet Information Services (IIS) 6.0 fully supports unicode characters, while IIS 5.0 does not. We recommend that you deploy IIS 6.0 to run any Web services that require unicode file names.

Installing and Configuring BizTalk Server 2006

This section provides information about installing and configuring Microsoft BizTalk Server 2006.

In This Section

- Installing BizTalk Server 2006
- Configuring BizTalk Server 2006

Installing BizTalk Server 2006

This section provides information about installing Microsoft BizTalk Server 2006.

In This Section

- Installing BizTalk Server on a Single Server
- Installing BizTalk Server on Multiserver

Installing BizTalk Server on a Single Server

The following table lists the documents that describe how to install BizTalk Server 2006 on a single server.

Content	Description
Quick Start Guide to Installing and Configuring BizTalk Server 2006	• This document provides scaled down instructions for installing and configuring a complete installation of Microsoft BizTalk Server 2006 on a single computer running Microsoft Windows Server 2003. The information in this quick start guide assumes certain prerequisite knowledge

	on behalf of the reader.
Installing and Configuring BizTalk Server 2006 on Windows Server 2003	• This document provides hardware and software requirements and detailed instructions for installing and configuring Microsoft BizTalk Server 2006 on a single computer running Microsoft Windows Server 2003.
Installing and Configuring BizTalk Server 2006 on Windows 2000 Server	• This document provides hardware and software requirements and detailed instructions for installing and configuring Microsoft BizTalk Server 2006 on a single computer running Microsoft Windows 2000 Server.
Installing and Configuring BizTalk Server 2006 on Windows XP	• This document provides hardware and software requirements and detailed instructions for installing and configuring Microsoft BizTalk Server 2006 on a single computer running Microsoft Windows XP.

Installing BizTalk Server on Multiserver

The following table lists the document that describes how to install and configure BizTalk Server 2006 in multiple computer environments.

Content	Description
Installing and Configuring BizTalk Server 2006 in Multiple Computer EnvironmentsInstalling and Configuring BizTalk Server 2006 in Multiple Computer Environments	• This document provides the detailed guidelines and instructions for installing and configuring BizTalk Server 2006 in multiple computer environments including the steps to cluster Enterprise Single Sign-On service on Master Secret Server and cluster BizTalk Server In-Process Host on Windows Server 2003 cluster services.

Configuring BizTalk Server 2006

BizTalk Server 2006 gives you the ability to configure your server in one of two modes, Basic or Custom. Basic Configuration is targeted for developers setting up single server installations for development. Custom configuration allows you to configure the server using advanced configuration options. With custom configuration, you have the ability to selectively configure or unconfigure each feature.

In This Section

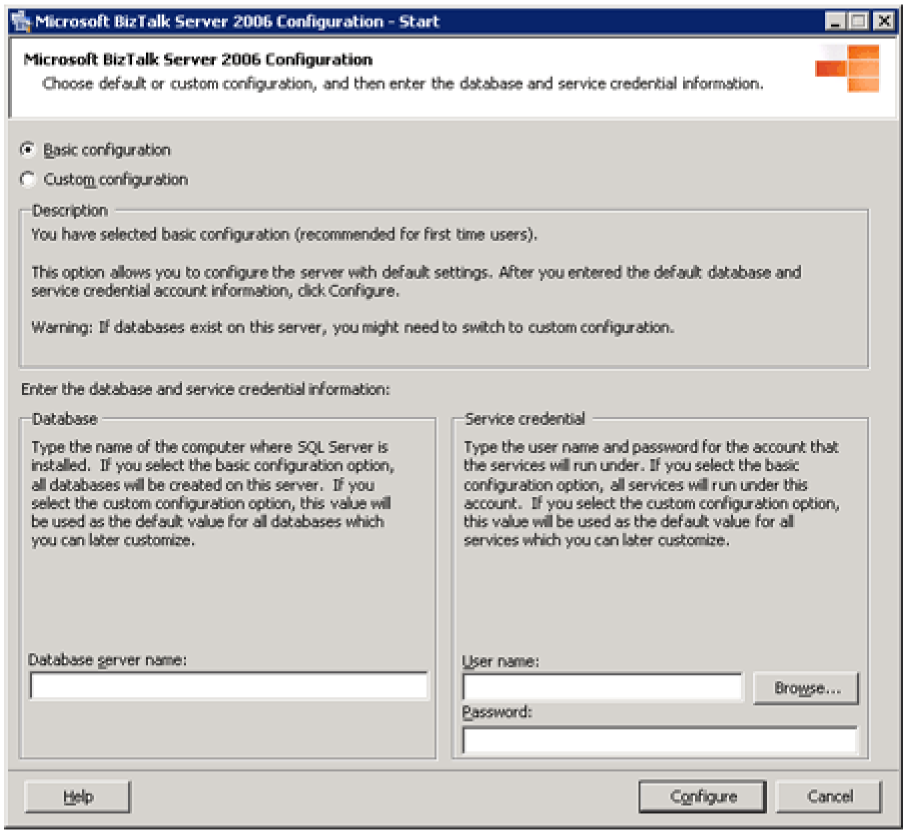
- Basic Configuration
- Custom Configuration

Basic Configuration

BizTalk Server 2006 provides the ability to configure the server using default settings. The default settings for the server are configured using the database server name, username, and password that you enter into the configuration wizard.

When using the basic configuration option, the following occurs:

- All database names will be generated by BizTalk Server.
- All applicable database logon information will run under the account provided.
- All BizTalk Server services will be generated by BizTalk Server.
- All BizTalk Server services will run under the account provided. The configuration process grants this account the necessary security permissions on the server and objects in SQL Server
- All features will be configured based on the pre-requisite software you have installed on the machine.
- The Default Web Site in Internet Information Services (IIS) is used for any feature that requires IIS.
- The user that is running the configuration tool will be the BAS site owner.
- The logged on user must be part of "OLAP Administrators" on the OLAP box.


Microsoft BizTalk Server 2006 Configuration - Start

Microsoft BizTalk Server 2006 Configuration
Choose default or custom configuration, and then enter the database and service credential information.

(●) Basic configuration
( ) Custom configuration

Description
You have selected basic configuration (recommended for first time users).

This option allows you to configure the server with default settings. After you entered the default database and service credential account information, click Configure.

Warning: If databases exist on this server, you might need to switch to custom configuration.

Enter the database and service credential information:

Database
Type the name of the computer where SQL Server is installed. If you select the basic configuration option, all databases will be created on this server. If you select the custom configuration option, this value will be used as the default value for all databases which you can later customize.

Database server name:

Service credential
Type the user name and password for the account that the services will run under. If you select the basic configuration option, all services will run under this account. If you select the custom configuration option, this value will be used as the default value for all services which you can later customize.

User name:
Password:

To configure your server using basic configuration

1. In **Microsoft BizTalk Server 2006 Configuration**, select Basic configuration

2. Under **Database**, type the name of the SQL server.

1. Under **Service Credential**, type the username and password that the services will run under.

2. Click **Configure**.

3. On the **Summary** screen, review the configuration, and then click **Configure**.

4. On the **Completion** screen, click **Finish**.

Considerations for Basic Configuration

When you configure BizTalk Server 2006 using basic configuration, consider the following:

- Configuring a remote SQL Server is not supported.

- The account you are logged on as must be part of the local administrators group and have System Administrator rights on SQL Server.
- The account you are logged on must be part of "OLAP Administrators" on the OLAP box if configuring.
- You cannot configure BAM Analysis on a SQL Server named instance using basic configuration. If you are using named instances and want to configure BAM Analysis, you should use the custom configuration manager.
- If you are using BizTalk Server Administration console to manage BAS artifacts, you must enable Transaction Internet Protocol (TIP) transactions for Microsoft Distributed Transaction Coordinator (DTC) on both the computer running the BAS Web site, and the computer running the BizTalk Server Administration console. By default, TIP functionality is disabled on Windows XP SP2 and Windows Server 2003 where in Windows 2000 Server, TIP functionality is disabled after you install security update 902400. To enable TIP functionality, follow the steps described in http://go.microsoft.com/fwlink/?LinkId=58318.

Custom Configuration

This topic discusses custom configuration in Microsoft BizTalk Server 2006.

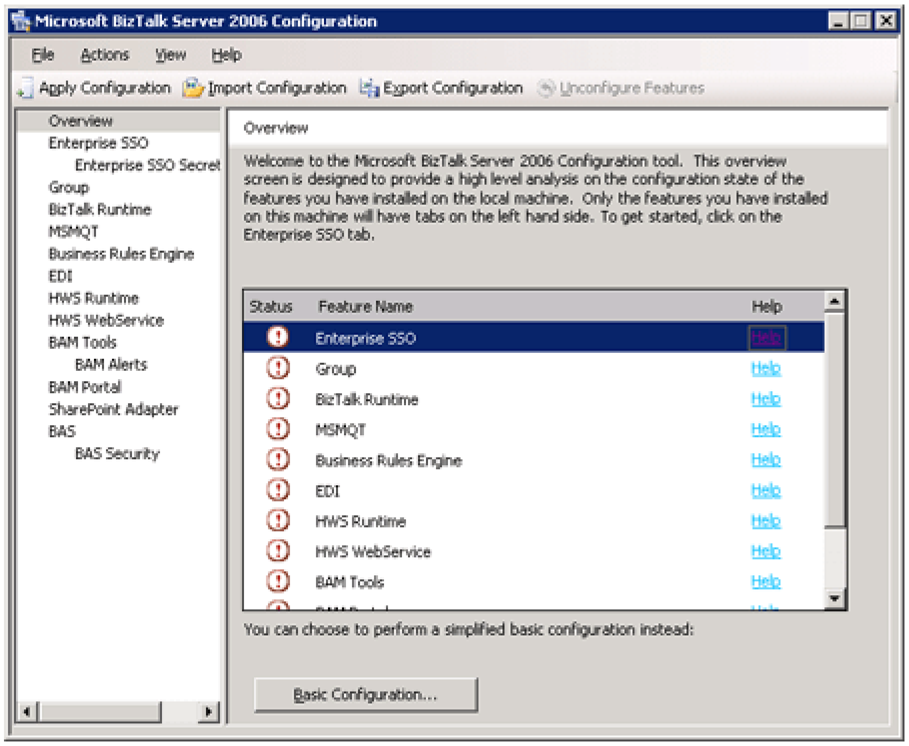
In This Section

- Working with the Custom Configuration Manager
- Configuring Groups Using the Configuration Manager
- Configuring Enterprise SSO Using the Configuration Manager
- Configuring BizTalk Server Runtime Using the Configuration Manager
- Configuring MSMQT Using the Configuration Manager
- Configuring Business Rules Engine Using the Configuration Manager
- Configuring the EDI Service Using the Configuration Manager
- Configuring HWS Runtime Using the Configuration Manager
- Configuring HWS Web Service Using the Configuration Manager
- Configuring BAM Tools Using the Configuration Manager
- Configuring BAM Alerts Using the Configuration Manager
- Configuring BAM Portal Using the Configuration Manager

- Configuring Windows SharePoint Services Adapter Using the Configuration Manager
- Configuring BAS Using the Configuration Manager
- Configuring BAS Security Using the Configuration Manager
- Working with the Configuration Framework

Working with the Custom Configuration Manager

The custom configuration manager provides a high-level analysis on the configuration state of the features you have installed on the local machine. The tool allows you to configure and unconfigure features, configure security settings, and import and export configurations.



Overview

Welcome to the Microsoft BizTalk Server 2006 Configuration tool. This overview screen is designed to provide a high level analysis on the configuration state of the features you have installed on the local machine. Only the features you have installed on this machine will have tabs on the left hand side. To get started, click on the Enterprise SSO tab.

Status	Feature Name	Help
!	Enterprise SSO	Help
!	Group	Help
!	BizTalk Runtime	Help
!	MSMQT	Help
!	Business Rules Engine	Help
!	EDI	Help
!	HWS Runtime	Help
!	HWS WebService	Help
!	BAM Tools	Help

You can choose to perform a simplified basic configuration instead:

Prerequisites for Using Custom Configuration Manager

When you configure BizTalk Server 2006 using custom configuration, consider the following:

- The account you are logged on as must be part of the local administrators group and have System Administrator rights on SQL Server.

- The default accounts generated by BizTalk Server and listed in the custom configuration manager are local groups. In a multiserver environment you must substitute the local groups with domain groups.
- The account you are logged on must be part of "OLAP Administrators" on the OLAP computer if configuring.

Applying Configurations

You can apply BizTalk Server configurations using the custom configuration manager.

To apply configurations

1. In the **Custom Configuration Manager**, click **Apply Configuration**.
2. On the **Summary** screen, review the configuration, and then click **Configure**.
3. On the **Completion** screen, click **Finish**.

Importing Configurations

You can import BizTalk Server configuration files using the custom configuration manager. For more information on working with these files, see Working with the Configuration Framework.

To import configurations

1. In the **Custom Configuration Manager**, click **Import Configuration**.
2. In the **Open** dialog box, select the configuration file you want to import, and then click **Open**.
3. In the **Custom Configuration Manager**, click **Apply Configuration**.
4. On the **Summary** screen, review the configuration, and then click **Configure**.
5. On the **Completion** screen, click **Finish**.

Exporting Configurations

You can export BizTalk Server configurations using the custom configuration manager. For more information on working with these files, see Working with the Configuration Framework

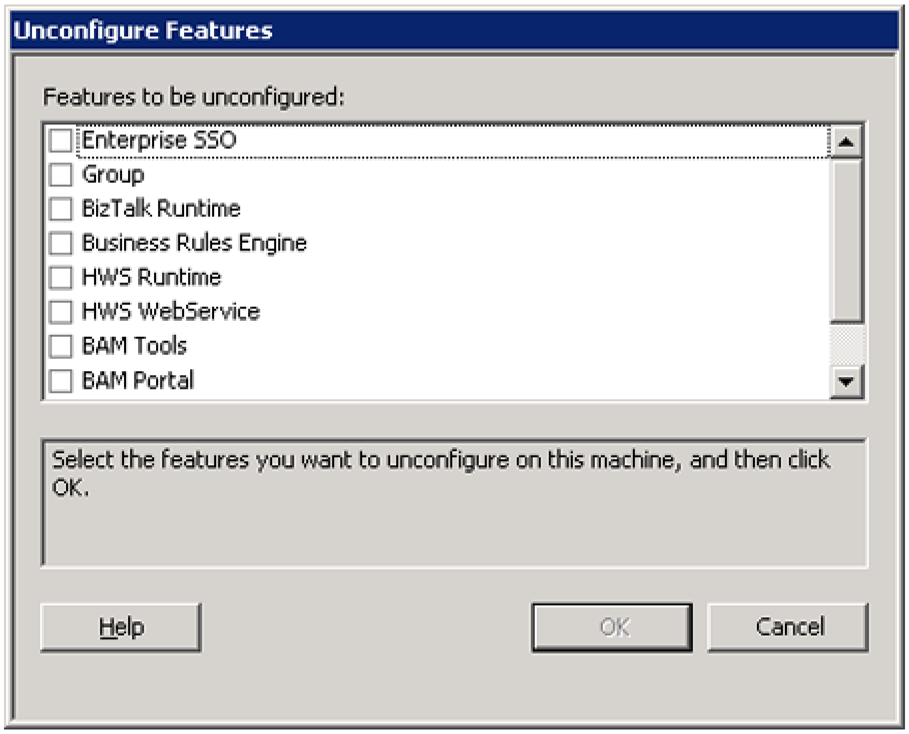
To export configurations

1. In the **Custom Configuration Manager**, click **Export Configuration**.

2. In the **Save As** dialog box, Type the file name you want to save, and then click **Save**.

Unconfiguring Features

You can unconfigure BizTalk Server features on the machine by removing them in the custom configuration manager.


Unconfigure Features

Features to be unconfigured:

- [ ] Enterprise SSO
- [ ] Group
- [ ] BizTalk Runtime
- [ ] Business Rules Engine
- [ ] HWS Runtime
- [ ] HWS WebService
- [ ] BAM Tools
- [ ] BAM Portal

Select the features you want to unconfigure on this machine, and then click OK.

To unconfigure features

1. In the **Custom Configuration Manager**, click **Unconfigure Features**.

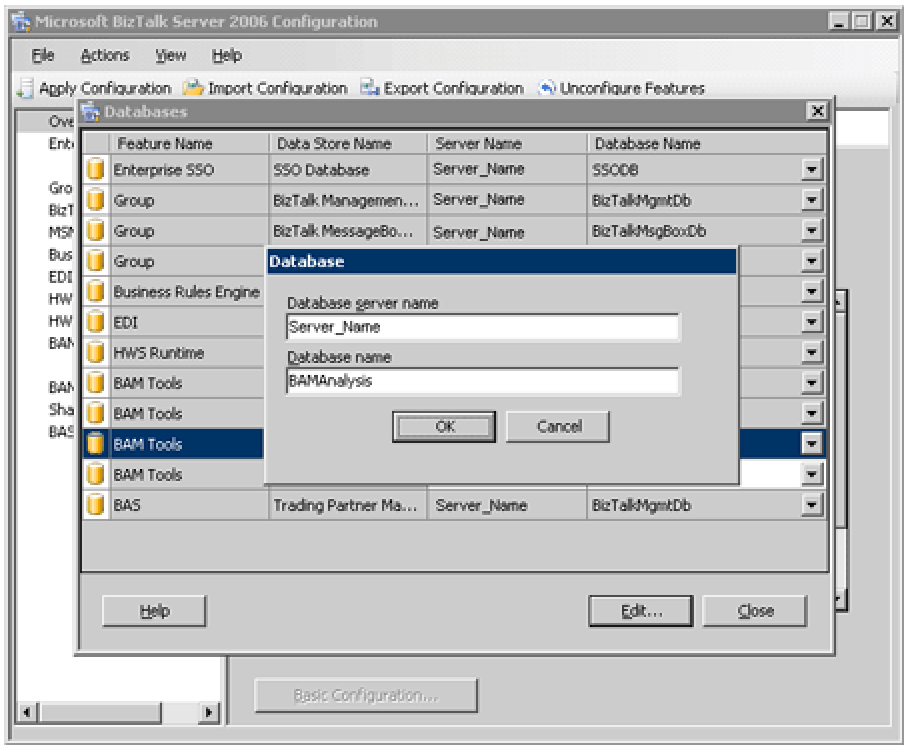
2. In the **Unconfigure Features** dialog box, select the features you want to remove, and then click **OK**.

1. On the **Summary** screen, review the configuration, and then click **Unconfigure**.

2. On the **Completion** screen, click **Finish**.

Editing Databases

You can change the database servers and databases associated with BizTalk Server features by editing them in the custom configuration manager.


Microsoft BizTalk Server 2006 Configuration
File Actions View Help
Apply Configuration Import Configuration Export Configuration Unconfigure Features
Databases
Database
OK Cancel
Help Edit... Close
Basic Configuration...

Feature Name	Data Store Name	Server Name	Database Name
Enterprise SSO	SSO Database	Server_Name	SSODB
Group	BizTalk Managemen...	Server_Name	BizTalkMgmtDb
Group	BizTalk MessageBo...	Server_Name	BizTalkMsgBoxDb
Group			
Business Rules Engine			
EDI			
HWS Runtime			
BAM Tools			
BAM Tools			
BAM Tools			
BAM Tools			
BAS	Trading Partner Ma...	Server_Name	BizTalkMgmtDb

Database server name: Server_Name
Database name: BAMAnalysis

To edit databases

1. In the **Custom Configuration Manager**, click **View**, and then click **Databases.**

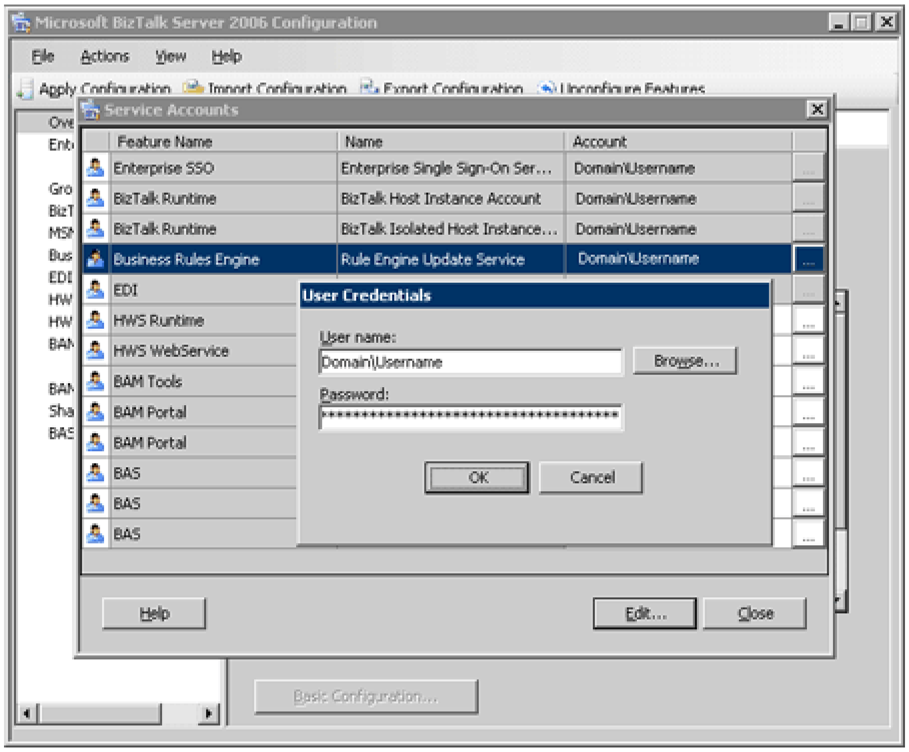
2. In the **Databases** dialog box, select the feature you want to edit, and then click **Edit**.

3. In the **Databases** dialog box, type the database server name and database name you want to use, and then click **OK**.

4. In the **Databases** dialog box, click **Close**.

Editing Service Accounts

You can change the service accounts associated with BizTalk Server features by editing them in the custom configuration manager.



Feature Name	Name	Account
Enterprise SSO	Enterprise Single Sign-On Ser...	Domain\Username
BizTalk Runtime	BizTalk Host Instance Account	Domain\Username
BizTalk Runtime	BizTalk Isolated Host Instance...	Domain\Username
Business Rules Engine	Rule Engine Update Service	Domain\Username
EDI		
HWS Runtime		
HWS WebService		
BAM Tools		
BAM Portal		
BAM Portal		
BAS		
BAS		
BAS		

User name: Domain\Username

Password: ************************************

To edit service accounts

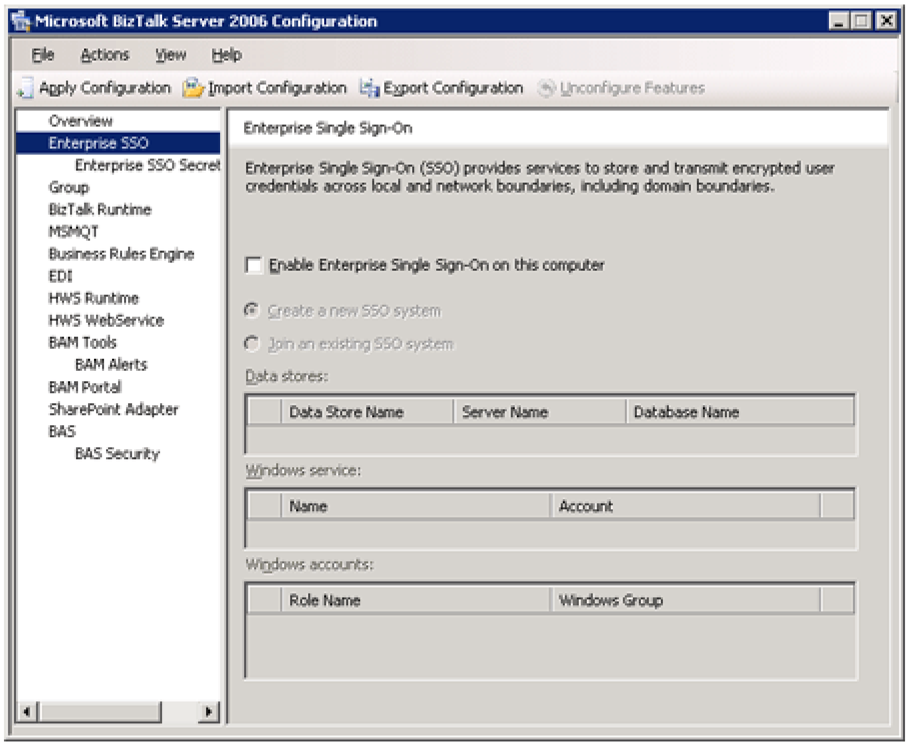
1. In the **Custom Configuration Manager**, click **View**, and then click **Service Accounts…**

2. In the **Service Accounts** dialog box, select the feature you want to edit, and then click **Edit**.

3. In the **User Credentials** dialog box, type the username and password you want to use, and then click **OK**.

4. In the **Service Accounts** dialog box, click **Close**.

Configuring Enterprise SSO Using the Configuration Manager



Enterprise Single Sign-On

Enterprise Single Sign-On (SSO) provides services to store and transmit encrypted user credentials across local and network boundaries, including domain boundaries.

Enable Enterprise Single Sign-On on this computer

Create a new SSO system

Join an existing SSO system

Data stores:

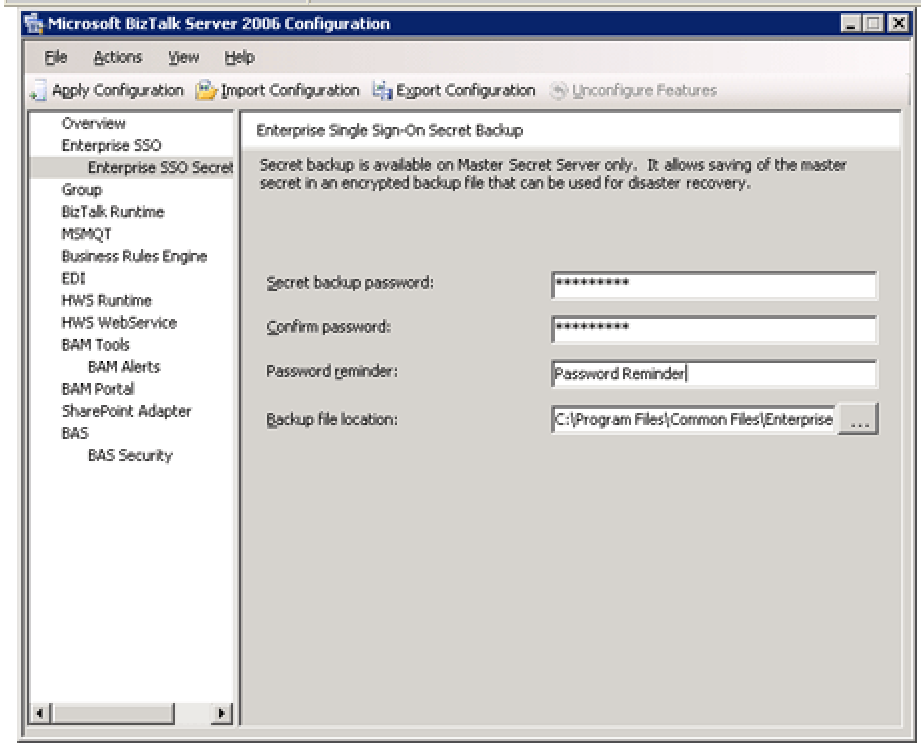
Windows service:

Windows accounts:

Use the **Enterprise Single Sign-On (SSO)** page to configure SSO settings for your BizTalk Server environment.

Use this	To do this
Enable Enterprise Single Sign-On on this computer	Select **Enable Enterprise Single Sign-On on this computer** to configure this server with SSO settings.
Create a new SSO system	Select **Create a new SSO system** if this is the first SSO server you are configuring in your SSO system. This also creates and configures the SSO Credential database. You must also back up the secret on this secret server.
Join an existing SSO system	Select **Join an existing SSO system** to connect to an existing SSO system in the BizTalk Server environment.
Data stores	The **Data stores** list provides an editable view of the data stores used for the SSO database.
Windows service	The **Windows service** list provides an editable view of the account used to run the Enterprise Single Sign-On service.
Windows accounts	The **Windows accounts** list provides an editable view of the SSO Administrators and SSO Affiliate Administrators Windows

	groups.



Enterprise Single Sign-On Secret Backup

Secret backup is available on Master Secret Server only. It allows saving of the master secret in an encrypted backup file that can be used for disaster recovery.

Secret backup password: **********

Confirm password: **********

Password reminder: Password Reminder

Backup file location: C:\Program Files\Common Files\Enterprise

Use the **Enterprise Single Sign-On Secret Backup** page to save the master secret to an encrypted backup file in the event that disaster recovery is needed.

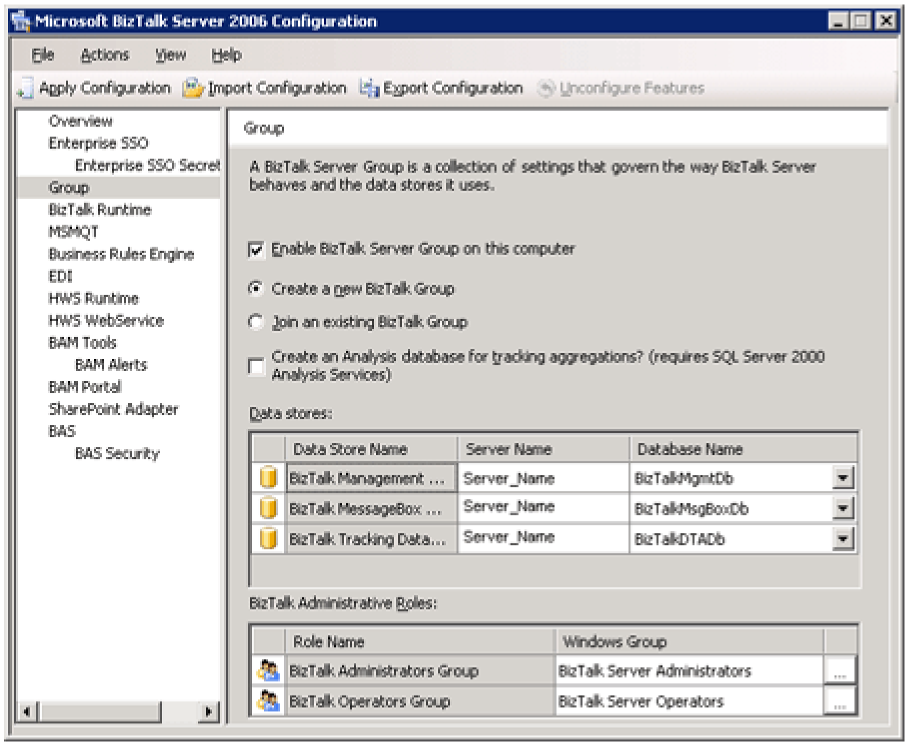
Use this	To do this
Secret backup password	Type the password for the backup file.
Confirm secret backup password	Confirm the password for the backup file.
Secret backup reminder	Type a reminder for the password you enter.
Secret backup file	Provide a file name and file path to the secret backup file. By default it is stored at <drive>:\Program Files\Common Files\Enterprise Single Sign-On\.

Considerations for configuring Enterprise SSO

When you configure Enterprise SSO in BizTalk Server 2006, consider the following:

- When configuring the SSO Windows accounts using local accounts, you must specify the account name without the computer name.

Configuring Groups Using the Configuration Manager



Group

A BizTalk Server Group is a collection of settings that govern the way BizTalk Server behaves and the data stores it uses.

[✓] Enable BizTalk Server Group on this computer

(●) Create a new BizTalk Group

( ) Join an existing BizTalk Group

[ ] Create an Analysis database for tracking aggregations? (requires SQL Server 2000 Analysis Services)

Data stores:

Data Store Name	Server Name	Database Name
BizTalk Management ...	Server_Name	BizTalkMgmtDb
BizTalk MessageBox ...	Server_Name	BizTalkMsgBoxDb
BizTalk Tracking Data...	Server_Name	BizTalkDTADb

BizTalk Administrative Roles:

Role Name	Windows Group
BizTalk Administrators Group	BizTalk Server Administrators
BizTalk Operators Group	BizTalk Server Operators

Use the **Groups** page to configure the group settings for the BizTalk Server you are configuring.

Use this	To do this
Enable the BizTalk Server Group on this computer	Select **Enable the BizTalk Server Group on this computer** to configure this server as a BizTalk Group.
Create a new BizTalk Group	Select **Create a new BizTalk Group** to create the BizTalk Server group that you will use to administer all the BizTalk Server databases.
Join an existing BizTalk Group	Select **Join an Existing BizTalk Group** to join a BizTalk group that already exists.
Create an Analysis database for tracking aggregations	Select **Create an Analysis database for tracking aggregations** if you have Microsoft SQL Server Analysis Services installed, and you want to track and store health monitoring OLAP cubes.

Data Stores	The **Data stores** list provides an editable view of the BizTalk Server Management, MessageBox, and Tracking databases. Type the name of the server and database that you will use for each one.
BizTalk Administrative Roles	The **BizTalk Administrative Roles** list provides an editable view of the BizTalk Server Administrators and Operators Windows groups.

Configuring BizTalk Server Runtime Using the Configuration Manager

Use the **Runtime** page to configure the BizTalk Server runtime on this computer.

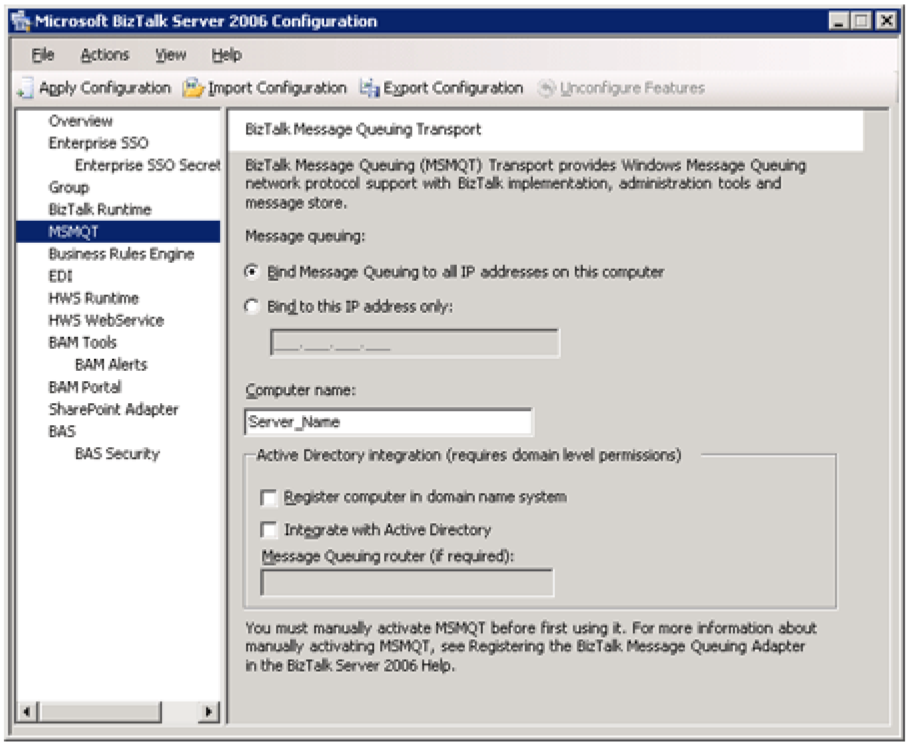
Use this	**To do this**
Enable BizTalk Host creation on this computer	Select **Enable BizTalk Host creation on this computer** to enable host creation on this computer.
Create BizTalk Host application	Select **Create BizTalk Host application** to create a Host application on this computer. **Trusted**: Select this if you want to pass the credentials (SSID and/or Party ID) of the sender when submitting messages to the MessageBox database. This is equivalent to creating a trust relationship between the servers.
Create BizTalk Isolated Host application	Select **Create BizTalk Isolated Host application** to create an Isolated Host application on this computer. **Trusted**: Select this if you want to pass the credentials (SSID and/or Party ID) of the sender when submitting messages to the MessageBox database. This is equivalent to creating a trust relationship between the servers.
Windows Service	The **Windows Service** list provides an editable view of the account used to run the BizTalk Host and Isolated Host Windows Services.
Windows Groups	The **Windows Groups** list provides an editable view of the BizTalk Host and Isolated Hosts windows groups.

Considerations for Configuring BizTalk Server Runtime

When you configure the BizTalk Server runtime in BizTalk Server 2006, consider the following:

- The first host you create in a group must be an In-Process host and host instance.
- In a multiserver environment in which you have two or more computers configured with BizTalk Server runtime and belonging to the same group, you cannot have the same service account used for both a trusted and untrusted host application.

Configuring MSMQT Using the Configuration Manage



BizTalk Message Queuing Transport

BizTalk Message Queuing (MSMQT) Transport provides Windows Message Queuing network protocol support with BizTalk implementation, administration tools and message store.

Message queuing:

(●) Bind Message Queuing to all IP addresses on this computer

( ) Bind to this IP address only:

Computer name: Server_Name

Active Directory integration (requires domain level permissions)

[ ] Register computer in domain name system

[ ] Integrate with Active Directory

Message Queuing router (if required):

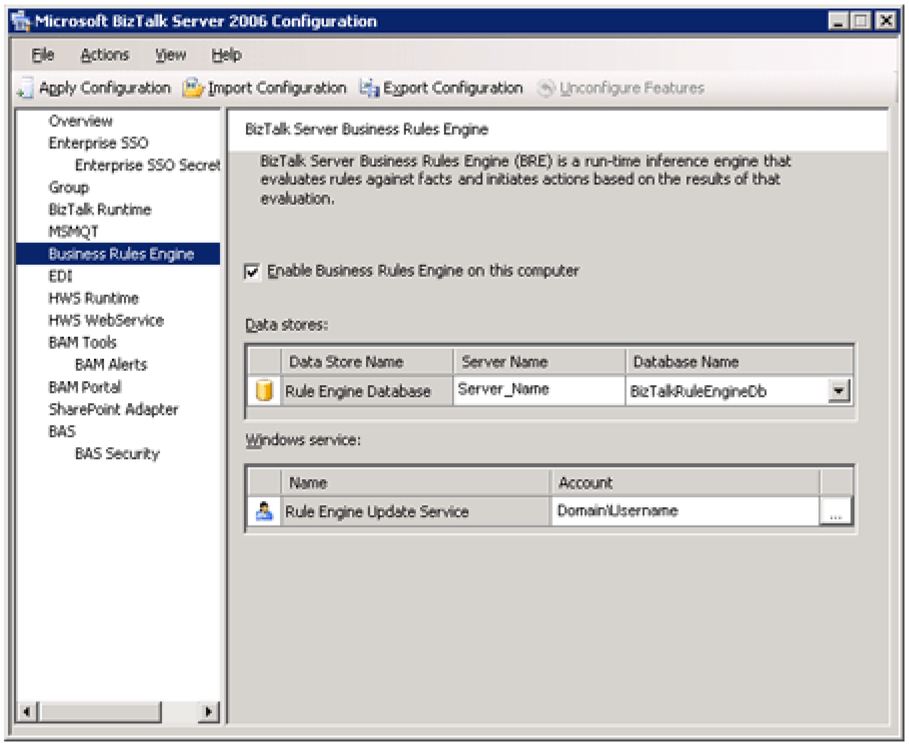
You must manually activate MSMQT before first using it. For more information about manually activating MSMQT, see Registering the BizTalk Message Queuing Adapter in the BizTalk Server 2006 Help.

The BizTalk Message Queuing adapter is installed on BizTalk Server 2006 with a default configuration. The BizTalk Message Queuing adapter default configuration is applied even if the adapter is not specifically configured with the BizTalk Configuration program. Therefore, the BizTalk Message Queuing adapter will provide full functionality if it is added to the list of adapters in the BizTalk Administration console, even if it has not been configured with the BizTalk Configuration program

Use the **MSMQT** page to configure the BizTalk Message Queuing Transport (MSMQT) on this computer.

Use this	To do this
Message Queuing	Select **Bind Message Queuing to all IP addresses on this computer** to bind the transport to all IP addresses. Select **Bind to this IP address only** to enter a specific IP address. Specify an IP address if you have a side-by-side installation with standard Message Queuing or if BizTalk Message Queuing must work with a specific IP address for NLB. Choose a side-by-side installation only after you examine all the other options - (for example, run MSMQ on a different computer or run MSMQ on a virtual computer from the same computer). Specify the name of the computer account used by MSMQT. All messages sent by MSMQT have the sender identity set to the Global Universal Identification (GUID) of this computer. If messages are sent to MSMQT and use a different computer name in the address (DIRECT=OS:<computer>\private$\queue), they are rejected. In most cases, it is acceptable to use the default value (local computer name). For side-by-side installations with standard Message Queuing, or if you choose Active Directory, you cannot use the local computer name. If you are setting up a BizTalk group, you must use the same name on all computers in the group. There are no restrictions on the name as long as it is the same on all computers and you correctly set up the Domain Name System (DNS).
Active Directory integration	Select **Register computer in domain name system** to register this server with your DNS. Select **Integrate with Active Directory** to integrate/register the server in Active Directory in the current domain. By default, this option is OFF. This option requires that the server belong to a domain with an Active Directory controller. Type the name of the Message Queuing router if your environment requires this. The router name is used in the Active Directory configuration to send messages with PRIVATE= and PUBLIC= addresses.

Configuring Business Rules Engine Using the Configuration Manager



BizTalk Server Business Rules Engine

BizTalk Server Business Rules Engine (BRE) is a run-time inference engine that evaluates rules against facts and initiates actions based on the results of that evaluation.

☑ Enable Business Rules Engine on this computer

Data stores:

Data Store Name	Server Name	Database Name
Rule Engine Database	Server_Name	BizTalkRuleEngineDb

Windows service:

Name	Account
Rule Engine Update Service	Domain\Username

Use the **Business Rules Engine** page to configure the Business Rules Engine for this computer. For more information about the Business Rules Engine, see Understanding BizTalk Server 2006.

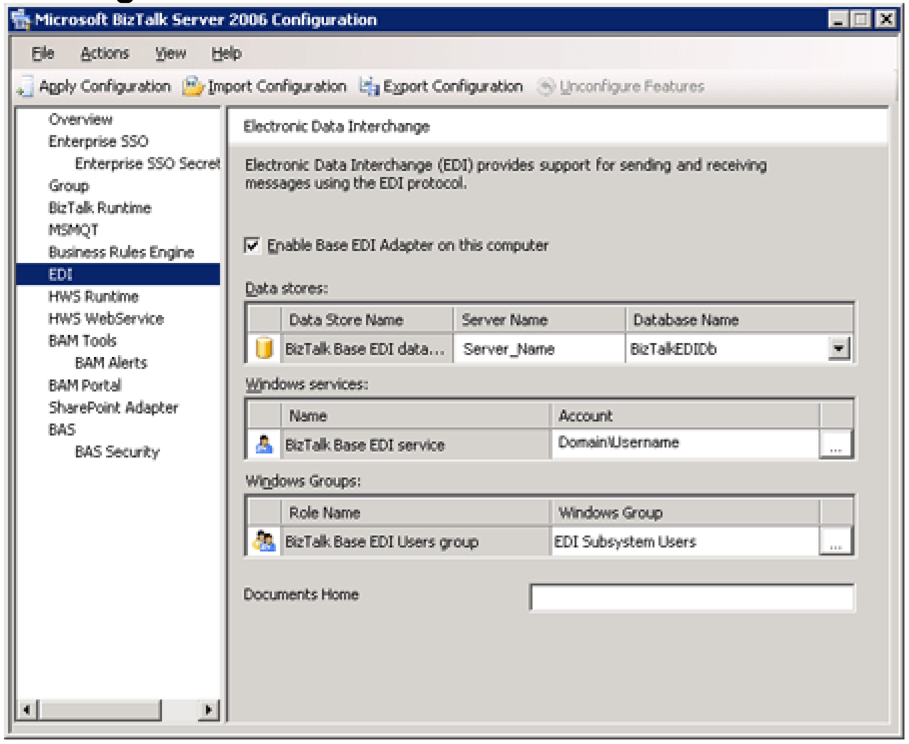
Use this	**To do this**
Enable the Business Rules Engine on this computer	Select **Enable the Business Rules Engine on this computer** to configure this server with the Business Rules Engine.
Data stores	The **Data stores** list provides an editable view of the Rules Engine database server name and database name.
Windows service	The **Windows service** list provides and editable view of the account used to run the Rule Engine Update Service.

Considerations for Configuring the Business Rule Engine

When you configure the Business Rule Engine in BizTalk Server 2006, consider the following:

- We recommend that you configure a BizTalk Server group before you configure the Business Rule Engine. If you configure the Business Rule Engine before configuring a BizTalk Server group, the BizTalk Server configuration does not add group-related administrative roles to the Rule Engine database.

Configuring the EDI Service Using the Configuration Manager



Electronic Data Interchange (EDI) provides support for sending and receiving messages using the EDI protocol.

☑ Enable Base EDI Adapter on this computer

Data stores:

Data Store Name	Server Name	Database Name
BizTalk Base EDI data...	Server_Name	BizTalkEDIDb

Windows services:

Name	Account
BizTalk Base EDI service	Domain\Username

Windows Groups:

Role Name	Windows Group
BizTalk Base EDI Users group	EDI Subsystem Users

Documents Home

Use the **EDI** page to configure the EDI adapter on this computer. The Microsoft BizTalk Server 2006 Base EDI adapter provides comprehensive EDI messaging functionality to complement the XML messaging capabilities of BizTalk Server 2006. The Base EDI adapter facilitates:

- Sending and receiving documents in an EDI format
- Creating partner-specific XML schemas
- Tracking the processing history of an individual document
- Automatically acknowledging the receipt of an EDI message

For more information on the Base EDI Adapter, see What Is the Base EDI Adapter?.

Use this	**To do this**
Enable the Base EDI Adapter on this computer	Select **Enable the Base EDI Adapter on this computer** to enable the Base EDI adapter.
Data stores	The **Data stores** list provides an editable view of the server name and database used for the adapter.
Windows services	The **Windows services** list provides an editable view of the account used to run the BizTalk Base EDI adapter service.
Windows Groups	The **Windows Groups** list provides an editable view of the BizTalk Based EDI windows group.
Documents Home	Indicates the name of the file share that serves as the working directory for the EDI Subsystem service. By default the EDIDocsHome file share that is created maps to the <drive>:\Documents and Settings\All Users\Application Data\Microsoft\Biztalk Server 2006\EDI\Subsystem directory of the BizTalk server that the configuration program is running on. If the BizTalk group that you are configuring will contain multiple BizTalk servers that are running the EDI Subsystem service then this file share should be configured to be highly available. See High Availability for the BizTalk Base EDI Adapter for more information. **Note** If you create a new file share for the Documents Home directory you must grant the <Domain>\EDI Subsystem Users group full control to the specified file share. If the <Domain>\EDI Subsystem Users group does not have "Full Control" access to the file share at the file system and share level then configuration of the EDI feature will fail.

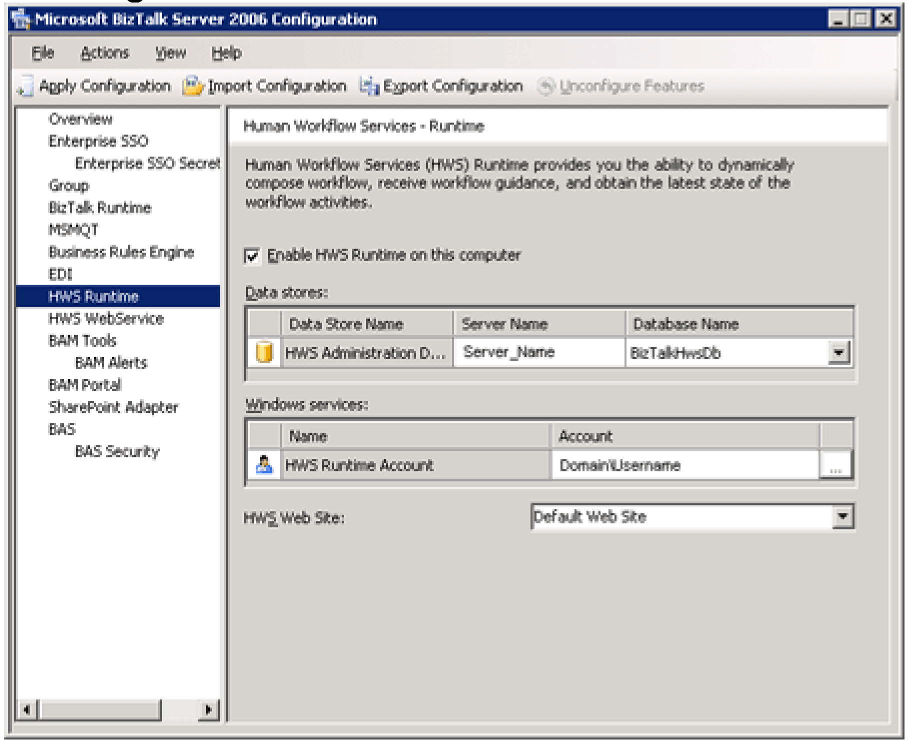
Considerations for Configuring the EDI Service

When you configure the EDI service in BizTalk Server 2006, consider the following:

- The BizTalk Base EDI database must be different than the BizTalk Management database.
- Using built-in accounts such as NT AUTHORITY\LOCAL SERVICE, NT AUTHORITY\NETWORK SERVICE, NT AUTHORITY\SERVICE or NT AUTHORITY\SYSTEM and local system account are not supported.
- The Client Connectivity component for SQL Client Tools is required.

- In an administration only installation of BizTalk Server 2006, you must add the following EDI connection strings to the registry.

Configuring HWS Runtime Using the Configuration Manager



Human Workflow Services - Runtime

Human Workflow Services (HWS) Runtime provides you the ability to dynamically compose workflow, receive workflow guidance, and obtain the latest state of the workflow activities.

☑ Enable HWS Runtime on this computer

Data stores:

Data Store Name	Server Name	Database Name
HWS Administration D...	Server_Name	BizTalkHwsDb

Windows services:

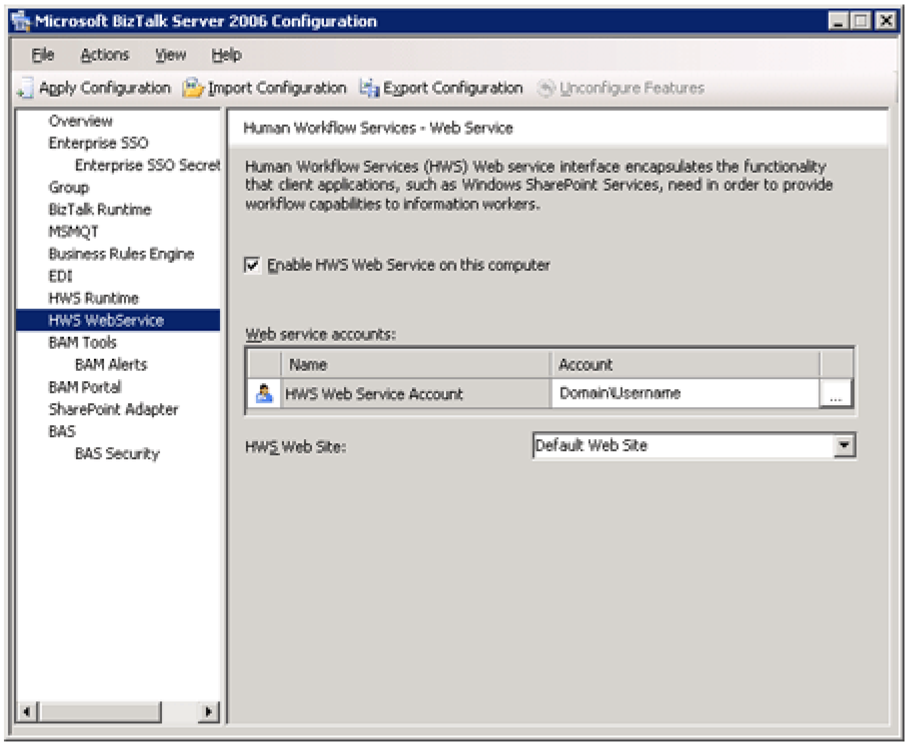
Name	Account
HWS Runtime Account	Domain\Username

HWS Web Site: Default Web Site

Use the **Human Workflow Services - Runtime** page to configure HWS on this computer.

Use this	To do this
Enable the HWS Runtime on this computer	Select **Enable the HWS Runtime on this computer** to enable the HWS Runtime.
Data stores	The **Data stores** list provides an editable view of the server name and database used for this feature.
Windows services	The **Windows services** list provides an editable view of the account used to run the HWS Runtime service.
HWS Web Site	Select the web site that will be used to host the application.

Configuring HWS Web Service Using the Configuration Manager



Human Workflow Services - Web Service

Human Workflow Services (HWS) Web service interface encapsulates the functionality that client applications, such as Windows SharePoint Services, need in order to provide workflow capabilities to information workers.

☑ Enable HWS Web Service on this computer

Web service accounts:

Name	Account
HWS Web Service Account	Domain\Username

HWS Web Site: Default Web Site

Use the **Human Workflow Services - Web Service** page to configure the HWS Web service on this computer.

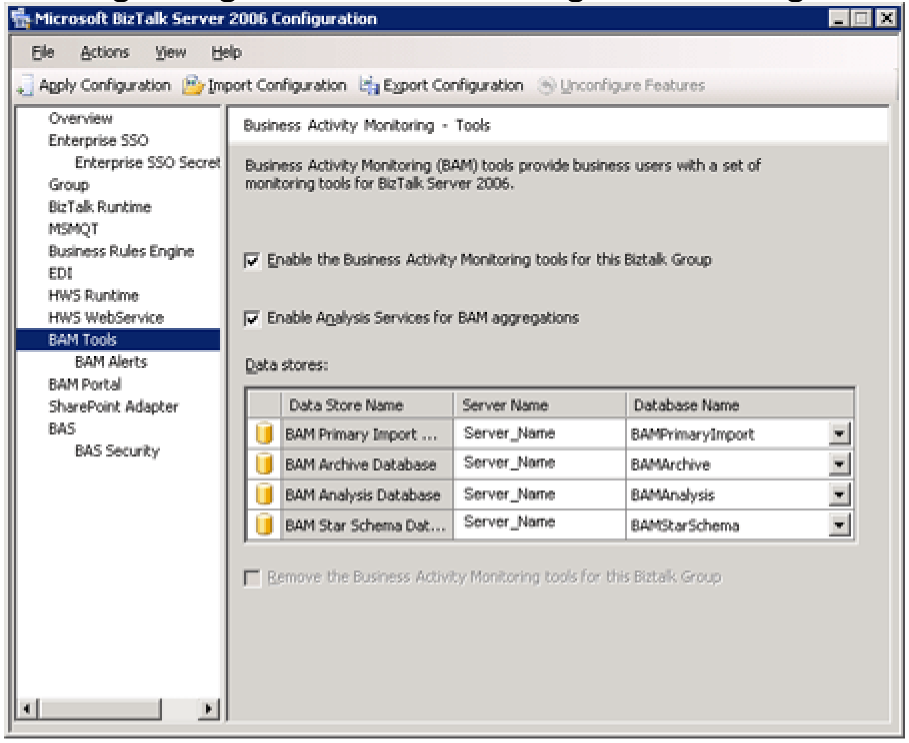
Use this	To do this
Enable the HWS Web Service on this computer	Select **Enable the HWS Web Service on this computer** to enable the HWS Web service.
Data stores	The **Data stores** list provides an editable view of the server name and database used for this feature.
Web service accounts	The **Web service accounts** list provides an editable view of the account used to run the HWS Web service.
HWS Web Site	Select the Web site that will be used to host the Web service.

Considerations for Configuring HWS Web Service

When you configure HWS Web service in BizTalk Server 2006, consider the following:

- The HWS Web service requires the HWS Runtime in the group you are configuring. For information on configuring HWS Runtime, see Configuring HWS Runtime Using the Configuration Manager .

Configuring BAM Tools Using the Configuration Manager



Business Activity Monitoring - Tools

Business Activity Monitoring (BAM) tools provide business users with a set of monitoring tools for BizTalk Server 2006.

[✓] Enable the Business Activity Monitoring tools for this Biztalk Group

[✓] Enable Analysis Services for BAM aggregations

Data stores:

Data Store Name	Server Name	Database Name
BAM Primary Import ...	Server_Name	BAMPrimaryImport
BAM Archive Database	Server_Name	BAMArchive
BAM Analysis Database	Server_Name	BAMAnalysis
BAM Star Schema Dat...	Server_Name	BAMStarSchema

[ ] Remove the Business Activity Monitoring tools for this Biztalk Group

Use the **Business Activity Monitoring - Tools** page to configure the server with monitoring tools for business users. The tools consist of the following:

- BAM add-in for Excel
- BAM Manager
- BAM Portal

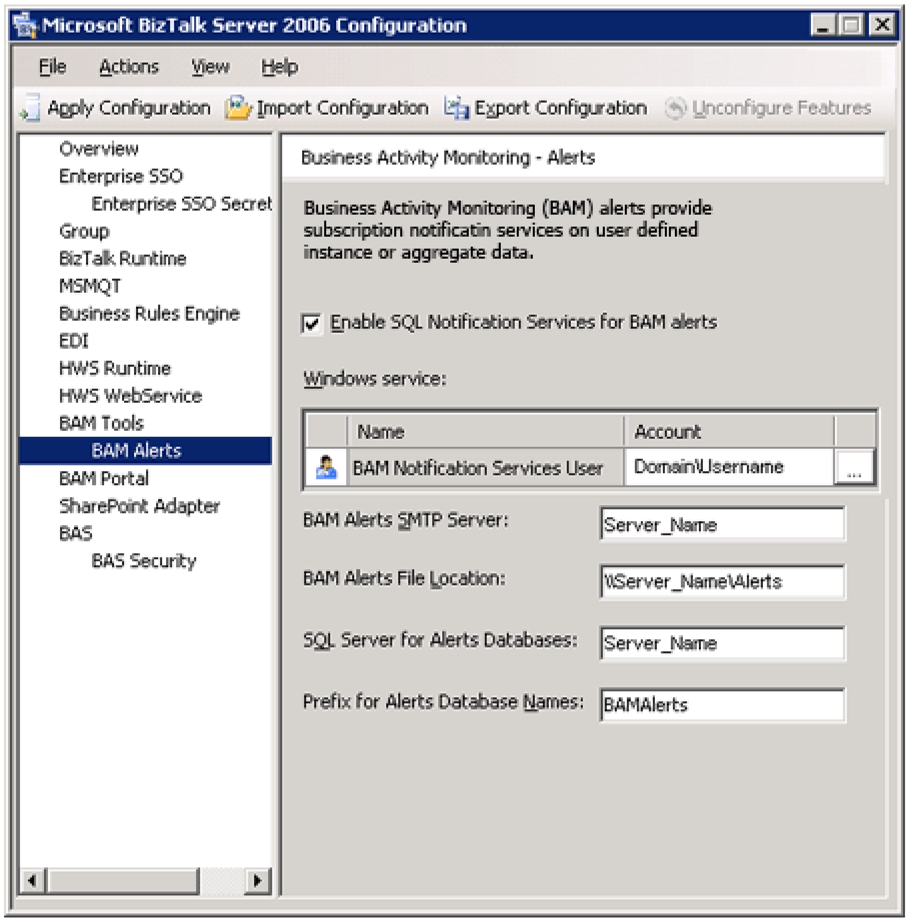
Use this	To do this
Enable Business Activity Monitoring tools	Select **Enable Business Activity Monitoring tools** to enable the tools.
Enable Analysis Services for BAM aggregations	Select **Enable Analysis Services for BAM aggregations** to provide tracking information for BAM alerts.
Data stores	The **Data stores** list provides an editable view of the server name and databases used for BAM tools.

Considerations for Configuring BAM Tools

When you configure BAM tools in BizTalk Server 2006, consider the following:

- Configuration of BAM tools will require certain SQL Server administrative functionality and must be performed from a machine that has SQL Server Client Tools with Data Transformation Service (DTS) or Integration Services installed depending on which version of SQL Server you are using.
- The BAM tools may be used by multiple BizTalk groups. When you unconfigure the BAM tools, the connection to the BizTalk group is removed; however the BAM SQL Server infrastructure will continue to work for other BizTalk groups pointing at the BAM Primary Import Tables.

Configuring BAM Alerts Using the Configuration Manager



Business Activity Monitoring - Alerts

Business Activity Monitoring (BAM) alerts provide subscription notificatin services on user defined instance or aggregate data.

☑ Enable SQL Notification Services for BAM alerts

Windows service:

	Name	Account
	BAM Notification Services User	Domain\Username

BAM Alerts SMTP Server: Server_Name

BAM Alerts File Location: \\Server_Name\Alerts

SQL Server for Alerts Databases: Server_Name

Prefix for Alerts Database Names: BAMAlerts

Use the **Business Activity Monitoring - Alerts** page to configure subscription based notification services.

Use this	To do this
Enable SQL Notification Services for BAM alerts	Select **Enable SQL Notification Services for BAM alerts** to enable BAM alerts.
Windows service	The **windows service** list provides an editable view of the account used to run the BAM Notification Service.
BAM Alerts SMTP Server	Type the name of the SMTP server that will be used to send the BAM alerts.
BAM Alerts File Location	Type the name of the network share that will be used to store the BAM alerts.
SQL Server for Alerts Databases	Type the name of the SQL Server that will be used for the Alerts database.

Prefix for Alerts Database Names	Type a prefix that will be used for the Alerts databases.

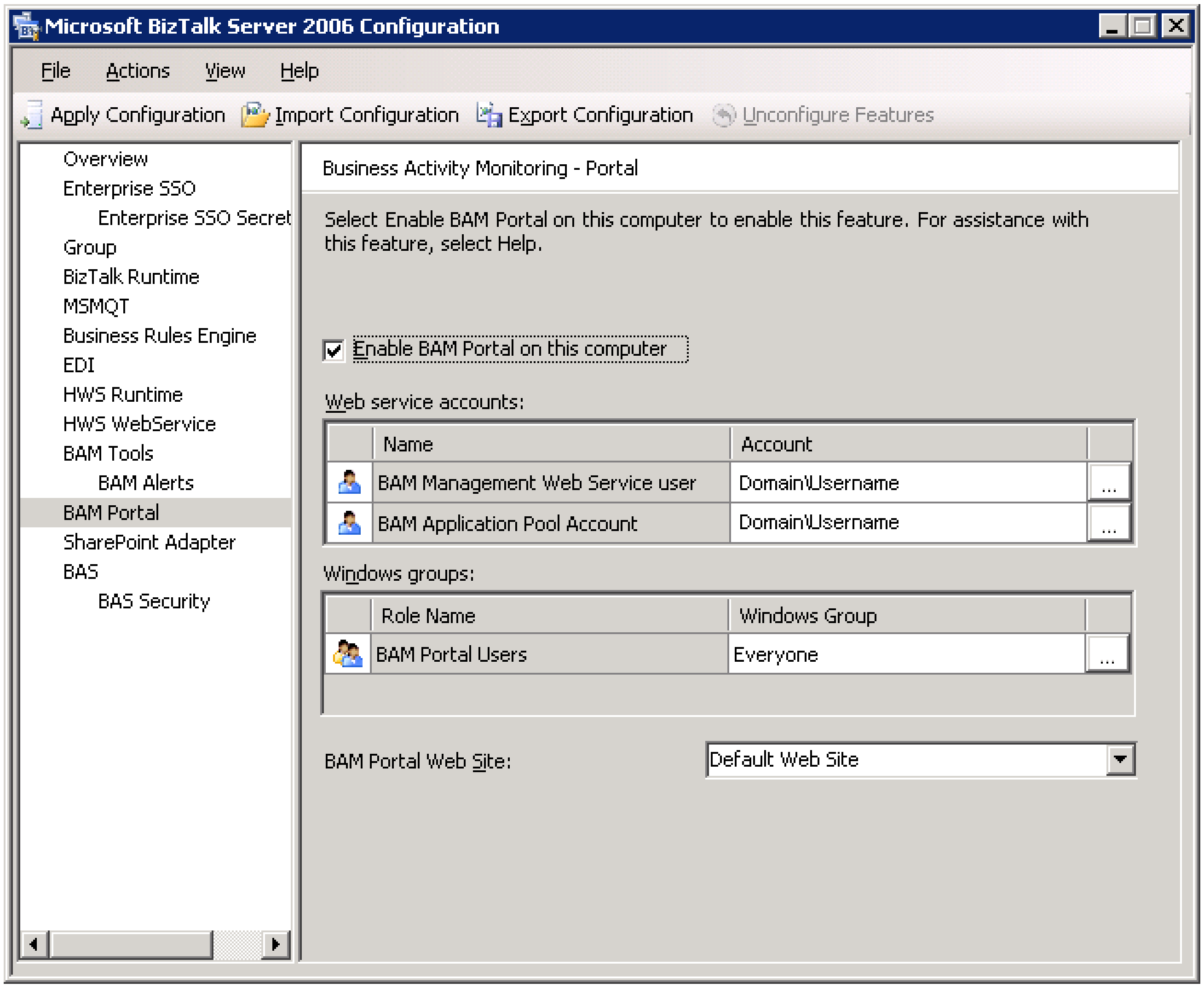
Considerations for Configuring BAM Alerts

When you configure BAM alerts in BizTalk Server 2006, consider the following:

- BAM alerts may fail during configuration when configuring in a distributed environment using binary collation if you log onto the computer with the domain in the username in the form of *domain\username*. To work around this problem, log onto the computer with the domain name in uppercase or type in the domain in uppercase.

- When you unconfigure BAM Alerts, the NS$BAMAlerts Windows Service does not get removed and you may not be able to reconfigure BAM Alerts. To remove this service, do the following:

Configuring BAM Portal Using the Configuration Manager



Business Activity Monitoring - Portal

Select Enable BAM Portal on this computer to enable this feature. For assistance with this feature, select Help.

☑ Enable BAM Portal on this computer

Web service accounts:

Name	Account
BAM Management Web Service user	Domain\Username
BAM Application Pool Account	Domain\Username

Windows groups:

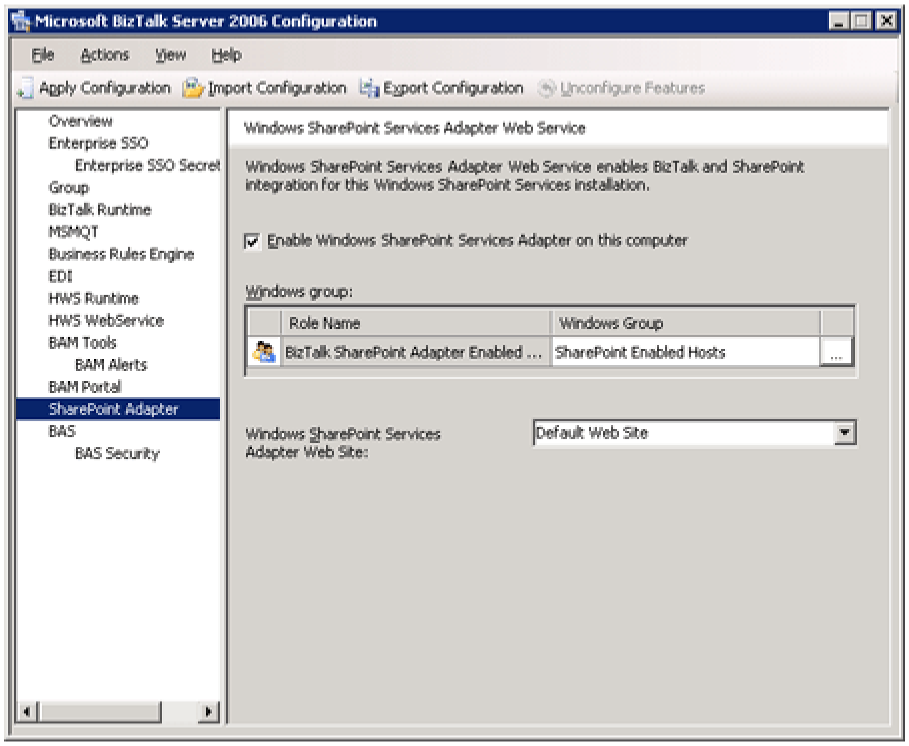
Role Name	Windows Group
BAM Portal Users	Everyone

BAM Portal Web Site: Default Web Site

Use the **Business Activity Monitoring - Portal** page to enable the BAM portal on this computer.

Use this	To do this
Enable the BAM Portal on this computer	Select **Enable the BAM Portal on this computer** to enable this feature.
Web service accounts	The **Web service accounts** list provides an editable view of the account used to run the BAM Management Web service.
Windows groups	The **Windows groups** list provides an editable view of the BAM Portal Windows group.
BAM Portal Web Site	Select the Web site that will be used to host the BAM Portal

Configuring Windows SharePoint Services Adapter Using the Configuration Manager



Windows SharePoint Services Adapter Web Service

Windows SharePoint Services Adapter Web Service enables BizTalk and SharePoint integration for this Windows SharePoint Services installation.

☑ Enable Windows SharePoint Services Adapter on this computer

Windows group:

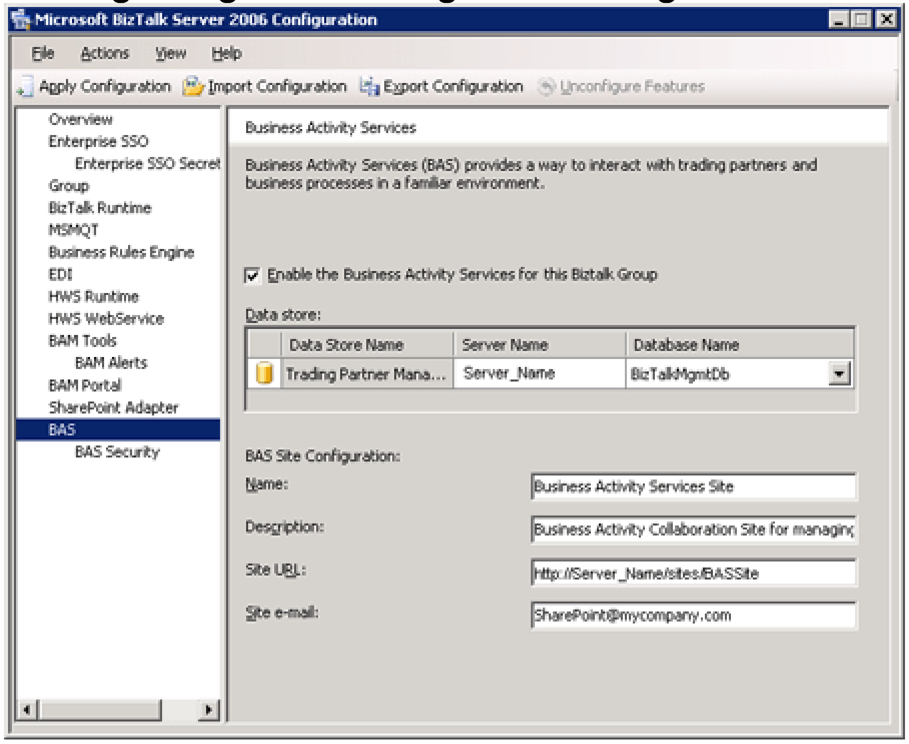
Role Name	Windows Group
BizTalk SharePoint Adapter Enabled ...	SharePoint Enabled Hosts

Windows SharePoint Services Adapter Web Site: Default Web Site

Use the **Windows SharePoint Services** page to configure the Windows SharePoint Services adapter on this computer. The BizTalk Server 2006 adapter for Windows SharePoint Services provides a tighter integration with Windows SharePoint Services and Microsoft Office InfoPath. For more information on setting up and deploying the Windows SharePoint Services Adapter, see What Is the Windows SharePoint Services Adapter? .

Use this	To do this
Enable Windows SharePoint Services Adapter on this computer	Select **Enable Windows SharePoint Services Adapter on this computer** to enable the adapter on this computer.
Windows group	The **Windows group** list provides an editable view of the BizTalk SharePoint Adapter Enabled Hosts windows group.
Windows SharePoint Services Adapter Web site	Select the Web site that will host the Windows SharePoint Service Adapter Web service.

Configuring BAS Using the Configuration Manager



Business Activity Services

Business Activity Services (BAS) provides a way to interact with trading partners and business processes in a familiar environment.

[✓] Enable the Business Activity Services for this Biztalk Group

Data store:

Data Store Name	Server Name	Database Name
Trading Partner Mana...	Server_Name	BizTalkMgmtDb

BAS Site Configuration:

Name:	Business Activity Services Site
Description:	Business Activity Collaboration Site for managing
Site URL:	http://Server_Name/sites/BASSite
Site e-mail:	SharePoint@mycompany.com

Use the **Business Activity Services (BAS)** page to configure this computer to interact with trading partners and business process. For more information on BAS, see Managing Partner Relationships with BAS.

Use this	To do this
Enable the Business Activity Trading Partner Management Web Services for this BizTalk Group	Select **Enable the Business Activity Trading Partner Management Web Services for this BizTalk Group** to enable BAS.
Data store	The **Data store** list provides an editable view of the server name and database name used for this feature.
BAS Site Configuration	To configure the BAS site, enter the following information: **Name**: Type the friendly name of the BAS site.

Considerations for configuring BAS

When you configure BAS in BizTalk Server 2006, consider the following:

- If you specify a web site that requires SSL for the **Site URL** option then you must also modify the web.config file for the TpPubWS and the TpMgmtWs web services after BAS is configured. To modify the web.config file for these web services follow these steps:
  - Open the web.config files located in the *%biztalk root%*\Business Activity Services\TPM\Management\ and *%biztalk root%*\Business Activity Services\TPM\Publishing\ directories of your BizTalk Server using a text editor such as Notepad.exe.
  - Locate the following entry in these files:
  - Change the entry to read as follows:
  - Save and close the modified web.config files.
- If you are using BizTalk Server Administration console to manage BAS artifacts, you must enable Transaction Internet Protocol (TIP) transactions for Microsoft Distributed Transaction Coordinator (DTC) on both the computer running the BAS Web site, and the computer running the BizTalk Server Administration console. By default, TIP functionality is disabled on Windows XP SP2 and Windows Server 2003 where in Windows 2000 Server, TIP functionality is disabled after you install security update 902400. To enable TIP functionality, follow the steps described in http://go.microsoft.com/fwlink/?LinkId=58318.
- Before configuring BAS on a stand-alone computer, you must configure BizTalk Server (runtime) on another computer. When you configure BAS, BizTalk Server prompts you for the server and database name for the Configuration database.
- If you configure BAS in a BizTalk group that contains a different BAS Site URL stored in the BizTalk management database, you will receive a warning during configuration. If you ignore the warning, the new BAS Site URL will replace the original BAS Site URL in the BizTalk Server Management database.
- If you configure BAS on a computer that was previously configured with BAS and you are using the same Trading Partner Management (TPM) database, you must synchronize the TPM database with Windows SharePoint Services after configuration is complete.
- When using BAS and Microsoft Internet Security and Acceleration (ISA), you must change the HTTP security filter setting. The HTTP filter screens all incoming Web requests to the ISA Server computer, and only allows requests that comply

with the restrictions configured in ISA Server. The Verify Normalization feature (enabled by default) specifies that requests with URLs that contain escape characters after normalization will be blocked. Escaped characters include, but are not limited to, the percent sign (%) and space character ( ). If this feature is enabled, Windows SharePoint Services document libraries will fail. URLs for document libraries and files uploaded and downloaded include non-standard characters such as the percent sign (%).

- You will receive invalid character error if you use space in the URL for BAS site during the configuration. If you must have a space in the URL, you will need to make sure that the parent portion of the URL is explicitly included in the SharePoint Central Administration web site under the virtual server being used

Configuring BAS in Multiserver Environment

To provide failover support for a Business Activity Services (BAS) configuration, configure BAS on multiple computers in a Network Load Balanced (NLB) Web server environment. In this environment, the computers share Windows SharePoint Services content and Trading Partner Manager (TPM) data. BizTalk Server uses the NLB cluster computer name to send messages to BAS to remove any dependency on a particular computer in a multiple Web server environment.

Overview

The following is an overview of the process for configuring the BAS on NLB cluster:

1. Set up a hardware or software NLB cluster.

2. Set the Default Web Site host header to the NLB cluster name in IIS 6.0 before extending the SharePoint sites on all NLB cluster nodes.

3. Extend SharePoint site on the first node by selecting Extend and create a content database option under Provisioning Options.

4. Extend SharePoint site on the second node by selecting Extend and map to another virtual server option under Provisioning Options. Map the virtual server that you configured on first node. You will need to use the same configuration and content databases.

5. In BizTalk Server Configuration Wizard, when configuring BAS on each node, use the virtual site URL instead of the node URL. For example, http://NLBCluster/sites/BASSite.

Example of Configuring BAS on NLB Cluster

The following is an example on how to install BAS on software NLB cluster on a Windows Server 2003 with two network interface card (NIC) presented. You must do this prior to extend any Windows SharePoint Services 2.0 virtual server. To install and configure BAS on NLB cluster, do the following:

1. On the first machine, click **Start**, click **Control Panel**, and then double-click **Network Connections**.

2. Right-click the second **Local Area Connection**, and then click **Properties**.

3. In the **Local Area Connection Properties** dialog box, click to select the **Network Load Balancing** check box, and then click **Properties**.

4. On **Cluster Parameters** tab, do the following

   - Enter the following for **Cluster IP configuration**:

b. **Name**	c. **Value**
d. IP address	e. 192.168.1.10
f. Subnet mask	g. 255.255.255.0
h. Full Internet name	i. BAS.Fabrikam.com

- Select the **Unicast** for the **Cluster operation mode**.

1. On **Host Parameters** tab, do the following:

   - Select **1** for the **Priority**.

   - Enter the following for **Dedicated IP configuration**:

c. **Name**	d. **Value**
e. IP address	f. 192.168.201.1
g. Subnet mask	h. 255.255.255.252

   - Select **Started** from the **Default state** drop down selection box.

2. Click on **OK** to close **Network Load Balancing Properties** dialog box.

3. Select the **Internet Protocol (TCP/IP)** and then click **Properties**.

4. Select **Use the following IP address** and enter the following:

1.	**Name**	1.	**Value**
1.	IP address	1.	192.168.1.10
1.	Subnet mask	1.	255.255.255.0

1. On second machine, repeat the Step 1 to Step 8 with the exception of the below,

- On **Host Parameters** tab, do the following:
- Select **2** for the **Priority**.
- Enter the following for **Dedicated IP configuration**:

a.	**Name**	**b.**	**Value**
c.	IP address	d.	192.168.201.2
e.	Subnet mask	f.	255.255.255.252

2. On first machine, click **Start** and select **Administrative Tools**. Select **Internet Information Services (IIS) Manager** and click **Internet Information Services (IIS) Manager**.

3. Expand **(local computer)** and expand **Web Sites**.

4. Right click on **Default Web Site** and select **Properties**. Click **Properties**.

5. On the **Web Site** tab, click **Advanced**. Select **Default** and click **Add**.

6. Enter **BAS.Fabriakm.com** for the **Host Header value** and click **OK**. Click **OK** on **Advanced Web Site Identification** page. Click **OK** on **Default Web Site Properties** page.

7. Repeat the Step 10 to Step 14 on second machine.

8. You will need to add a host (A) record to a zone in DNS on the domain controller in you network. To do so, do the following:

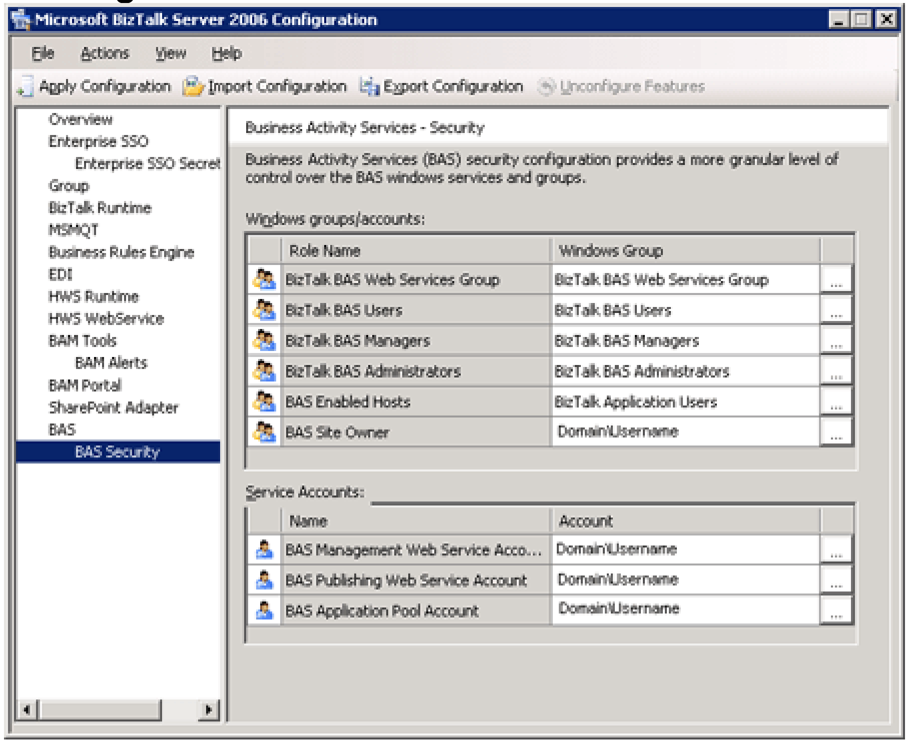
- Click **Start** and select **Administrative Tools**. Select **DNS** and click **DNS**.
- Expand **Forward Lookup Zones** and expand **fabrikam.com**.
- Right click on **fabrikam.com** and select **New Host (A)** and click **New Host (A)**.

- Enter **BAS.Fabrikam.com** for **Name** and **192.168.1.10** for **IP address**. Click **Add Host**. Click **OK** and then click **Done**.

9. You will now extend the Windows SharePoint Services 2.0 virtual server. When extending the virtual server on first machine, make sure that you select **Extend and create a content database** under **Provisioning Options**. When extending the virtual server on second machine, select **Extend and map to another virtual server** option to extend the virtual server and choose http://BAS.Fabrikam.com. For more information on configuring Windows SharePoint Services 2.0 for BizTalk Server 2006, see the BizTalk Server 2006 installation guide, at http://go.microsoft.com/fwlink/?LinkId=46922.

1. You are now ready to configure the BAS on NLB cluster in the multiple computer environments. When configuring the BizTalk Server 2006, use the below URL instead of the URL showing as default on the configuration page.

Configuring BAS Security Using the Configuration Manager



Business Activity Services - Security

Business Activity Services (BAS) security configuration provides a more granular level of control over the BAS windows services and groups.

Windows groups/accounts:

Role Name	Windows Group
BizTalk BAS Web Services Group	BizTalk BAS Web Services Group
BizTalk BAS Users	BizTalk BAS Users
BizTalk BAS Managers	BizTalk BAS Managers
BizTalk BAS Administrators	BizTalk BAS Administrators
BAS Enabled Hosts	BizTalk Application Users
BAS Site Owner	Domain\Username

Service Accounts:

Name	Account
BAS Management Web Service Acco...	Domain\Username
BAS Publishing Web Service Account	Domain\Username
BAS Application Pool Account	Domain\Username

Use the **Business Activity Services - Security** page to configure the windows groups and service accounts used for BAS.

Use this	To do this
Windows groups/accounts	The **Windows groups/accounts** list provides an editable view of windows groups used for the BAS roles.

Service Accounts	The **Service Accounts** list provides an editable view of the accounts used to run the BAS windows services.

BAS Roles

The following is a list of roles and descriptions for BAS:

Role	**Default Windows Group**	**Description**
BizTalk BAS Web Services Group	BizTalk BAS Web Services Group	The BAS Web Services group is an internal group used by the BAS components.
BizTalk BAS Users	BizTalk BAS Users	Business users perform publishing operations that do not modify BizTalk Server settings. They have the fewest privileges.
BizTalk BAS Managers	BizTalk BAS Managers	Business managers perform management operations that make changes in BizTalk Server, such as deploying partners and activating agreements.
BizTalk BAS Administrators	BizTalk BAS Administrators	BAS administrators perform operations that directly affect the operations of BizTalk Server, such as creating a BizTalk Server registration and synchronizing and repairing databases with the Trading Partner Management database.
BAS Enabled Hosts	BizTalk Application Users	This group has rights to call some of the BAS Web services for handling BAS parameters. Not all BizTalk Host Instance accounts can make calls to the BAS Web services. The default Windows group, "BizTalk Application Users", is the same group used for the default application.
BAS Site Owner	BAS Site Owner	This user is the site owner and Administrator of the SharePoint Services Web site used by BAS.

Considerations for Configuring BAS Security

When you configure BAS security in BizTalk Server 2006, consider the following:

- The BAS Site Owner cannot be configured with a Windows group account.

- When configuring a BAS only installation in a multiserver environment, you must either use domain groups or manually create the BizTalk BAS Web Services Group on the machine that hosts the TPM SQL databases.

Working with the Configuration Framework

The Configuration Framework is a generic method that enables you to quickly and easily change configuration at setup. In conjunction with the Microsoft Windows Installer (MSI), the Configuration Framework determines the state of your computer and the configuration tasks requiring action.

When you configure BizTalk Server for the first time using the Configuration Wizard, the Configuration Framework generates an XML file (a configuration snapshot) that you can then modify (that is, change user names, passwords, and so on) and export to other computers. You save your configuration snapshot on the Configuration Summary page, part of the Configuration Wizard. You can use this snapshot to replicate your configuration as part of a scripted installation. This file is located at <BizTalk Installation Path>\ConfigMain.xml

Configuration Framework Command Line Parameters

The following table describes the parameters you can run on the command line within the Configuration Framework.

Command Line Parameter	**Description**
/U	Unconfigures all features.
/S	Silent configuration. You must also pass the full path to the configuration XML that contains the features to install and configure. If /s is not passed, the tool runs in User Interface (UI) mode.
/L	• Sets the full path to the log file (optional).
/H	• Shows the valid command line parameters.

Upgrading to BizTalk Server 2006

BizTalk Server 2006 was designed to allow for an easy upgrade experience from its previous version. For information on upgrading, see the document list in below.

Content	Description
Upgrading to BizTalk Server 2006	• This document provides an overview of the upgrade process and provides information on some of the basic steps you will take during the process.

Migrating Artifacts from a Previous Version of BizTalk Server

This section describes how to migrate BizTalk artifacts from previous versions of BizTalk Server. Artifacts are the items comprising a BizTalk business solution, such as orchestrations, schemas, maps, and pipelines. This section describes the steps involved in migrating artifacts from BizTalk Server 2002 to BizTalk Server 2006. In addition, while migration is automatic when you perform an in-place upgrade to BizTalk Server 2006 from BizTalk Server 2004, this section covers some additional steps that you may want to take to use these artifacts. It also describes how to import artifacts from an .msi file that was created in BizTalk Server 2006 into a BizTalk Server 2006 application.

In This Section

- Migrating Artifacts from BizTalk Server 2002
- Migrating Artifacts from BizTalk Server 2006
- How to Import Artifacts from a BizTalk Server 2006 .msi File

Migrating Artifacts from BizTalk Server 2002

With careful planning and execution, migrating your business solutions from Microsoft BizTalk Server 2002 to Microsoft BizTalk Server 2006 can be a straightforward exercise. After your solutions are migrated, they can take advantage of the powerful new features and enhanced performance of BizTalk Server 2006. This section provides prescriptive guidance in using the BizTalk Server 2006 Migration Wizard to migrate a BizTalk Server 2002-based solution that includes orchestrations and messaging items.

In This Section

- Planning for Migration
- The Migration Wizard
- Migrating Messages
- Migrating Orchestrations

Planning for Migration

When planning your migration of a BizTalk Server 2002 solution to BizTalk Server 2006 solution, you must assess the technical architecture requirements, and plan the migration from a feature perspective. This section provides a detailed outlook of the planning considerations and recommendations from both perspectives.

The answers to the following questions help you plan your migration from BizTalk Server 2002 in your BizTalk Server 2006 environment.

Question	**Recommendation**
What is a promoted property?	Property promotion enables you to flag schema nodes at design time for promotion to the MessageBox at run time. In addition to any custom metadata you explicitly define by promoting properties, BizTalk Server 2006 has several pre-existing properties that it tracks by default. These pre-existing properties are not defined by instance schemas. For more information about promoting properties, see Promoting Properties .
What is a binding file?	Binding files are shortcuts so that you can bind once using the BizTalk Explorer, export that binding, and then import it when you need to assign the binding subsequent times. For example, if you install and deploy your development environment and save a binding file, when it is time to go into production, you can specify the binding file to use to create the bindings for the ports. For messaging migration, the Migration Wizard only generates one binding file for the send or receive locations in a solution. For more information about binding files, see Binding Files and Application Deployment.
Can I have multiple schemas with the same root node?	No. The messaging engine relies on the root node to resolve an XML instance to a schema at run time. (This is possible in BizTalk Server 2002.) For more information about schemas, see About Schemas .
Can I add an orchestration within my migration project?	Yes, it is possible, but not recommended because you have to manually create a send/receive port to/from an orchestration, thereby complicating the migration of your messaging solution. For more information about adding orchestrations, see Creating and Modifying Orchestrations .
Is it possible to create an orchestration that receives a non-XML message (a binary large object (BLOB)	You can pass type-less messages through orchestrations by declaring them as an XmlDocument. If you want to retrieve their data, you need to pass them to a .NET object as XLANGPart/XLANGMessage and call

for instance) through a pipeline with a custom disassembler and promote properties in the blob?	part.RetrieveAs(typeof(Stream)); by using part.LoadFrom(myStream), you can also construct non-XML messages in an orchestration. For more information about adding orchestrations, see Creating and Modifying Orchestrations .
How do I create a dynamic send port?	You can create a dynamic send port either using BizTalk Explorer or within an orchestration: **BizTalk Explorer** – Create the port using BizTalk Explorer, and then assign values to the transport properties in BizTalk Editor using the global property schema. You can then select the appropriate schema from references. By default, BizTalk Server 2006 references Microsoft.BizTalk.GlobalPropertySchema. Values to the transport properties come from your document. You need to promote elements from your schema using the global property schema. **Orchestration** – You can create dynamic ports in an orchestration, and, after you deploy it, the port is accessible from BizTalk Explorer. In the orchestration, you need to assign the address (transport) using an Expression shape. For more information about dynamic send ports in BizTalk Explorer, see How to Add a Dynamic Send Port Using BizTalk Explorer .
Will the Migration Wizard migrate custom script functoids?	No. Any maps that are migrated that contain custom script functoids need to be opened and functoids re-coded. For more information about dynamic send ports in BizTalk Explorer, see Migrating Functoids .
Will the Migration Wizard migrate my queue receive functions?	Yes. BizTalk Server supports all type of receive functions including BizTalk Server message queuing (MSMQ) file receive. BizTalk Server supports the conventional HTTP receive function, but not the HTTP file receive function because it has no meaningful use in the product. For more information about receive functions, see Managing Receive Locations.
Will the Migration Wizard migrate my pre-processor?	No. You must update your project by manually migrating any components such as pre-processors not migrated by the Migration Wizard. For more information about component migration, see Updating a Message Migration Project.
Will the Migration Wizard	No, with one exception. BizTalk Server 2006

migrate my channel routing and filter configurations?	automatically migrates routing and filtering configurations for channels that use dynamic routing or filtering. Custom routing done in a channel requires that you promote the properties and create the filters. For more information about migrating channels, see The Migration Wizard.
Will the Migration Wizard migrate orchestrations?	No. The Migration Wizard only migrates the following items. For more information about the Migration Wizard, see The Migration Wizard.

BizTalk Server 2002 Messaging Item	**BizTalk Server 2006 Migration Path**
document definitions	schemas
maps	maps
receive functions	receive locations
ports and channels	ports and pipelines
port groups (distribution lists)	send port groups

In This Section

- Feature Migration Planning
- Technical Architecture Requirements

Feature Migration Planning

BizTalk Server 2006 is a significantly different product than BizTalk Server 2002. It not only provides a rich set of new features, but offers new approaches to completing tasks.

For example, assume you have a business rule that must be invoked from various business processes. With BizTalk Server 2002, you probably solved this challenge by either implementing a decision shape in each orchestration, or by implementing the business rule in a custom object called from each orchestration. In BizTalk Server 2006, you can choose to implement the business rule within the Business Rules Library and call out to this rule from each orchestration. The question then becomes: "When I migrate this area of my BizTalk Server 2002 solution to BizTalk Server 2006, how should I establish a set of design principles to best take advantage of the new Business Rules Library?)"

Your design criteria could include a set of outstanding requirements for enhancing your BizTalk Server 2002 solution. The next question could be: "Do I implement functionality enhancements to the solution as it is being migrated?" If you plan to add functionality to your solutions as part of your migration project, you should complete a "horizontal" migration of the functionality first, and perform regression testing before adding or changing functionality.

Feature planning involves looking in detail at your BizTalk Server 2002 solution and deciding the most efficient way to migrate its functionality to BizTalk Server 2006. A new concept with BizTalk Server 2006 is that solutions are based on solution projects that must be built and deployed to a BizTalk Server 2006 environment.

As part of planning your feature migration, you should first segment your BizTalk Server 2002 solution, and take an inventory of your components.

In This Section

- Segment a BizTalk Server 2002 Solution
- BizTalk Server 2002 Component Inventory

Segment a BizTalk Server 2002 Solution

The first step in planning your feature migration is to segment your BizTalk Server 2002 solution by major functional areas. Create a logical grouping of BizTalk Server 2002 items to facilitate conversion to BizTalk Server 2006. Segment your BizTalk Server 2002 solution to partially migrate your solution and subsequently minimize your risk and effort.

To reduce configuration dependencies, you should further segment the functional areas into orchestration and messaging projects. To redeploy a new or changed migration project, all ports must not reference any objects in the assemblies within your migration project. By placing the orchestrations into a separate project, you reduce the complexity of configuration dependencies. In addition, the Migration Wizard creates a binding file that causes the migration to fail when you add an orchestration to your migration project.

BizTalk Server 2002 Component Inventory

After you segment your solution, you must understand all of the components that comprise each solution. Create an inventory all of the components in your orchestration and messaging solution by functional area. This inventory is helpful when you eventually convert the components to BizTalk Server 2006 items. For information about converting components, see Migrating Orchestration Implementation Shapes. A sample inventory of components follows.

Assembly: PlantMessagingObjects

Messaging Object	Name	Purpose
Queue Receive	qrfPlantActualReceive	Consumes power plant generation data in XML format delivered directly from

Function		external sources via HTTP posts.
	qrfPlantAlarmReceive	Consumes power plant alarm data in XML format delivered directly from external sources via HTTP posts.
	qrfPlantScheduleReceive	Consumes power plant schedule data in XML format delivered directly from external sources via HTTP posts.
File Receive Function	frfPlantActualReceive	Consumes power plant generation data in XML format delivered manually for re-submission by a system administrator.
	frfPlantAlarmReceive	Consumes power plant alarm data in XML format delivered manually for re-submission by a system administrator.
	frfPlantScheduleReceive	Consumes power plant schedule data in XML format delivered manually for re-submission by a system administrator.
Channel	chlPlantActualToPowerBiz	Validates XML message against power plant generation schema.
	chlPlantAlarmToPowerBiz	Validates XML message against power plant schema.
	chlPlantScheduleToPowerBiz	Validates XML message against power plant schedule schema.
Port	prtPlantDataToPowerBiz	Delivers power plant XML message to application integration components (AICs) for entry into PowerBiz application.
AIC	clsPlantAIC	Class that implements IBTSAppIntegration interface for processing inbound power plant documents.

Assembly: PlantOrchestrations

Business Process Object	**Name**	**Purpose**
Orchestrations	orchPlantProcess	Overarching business process flow control.

	orchPlantSchedule	Controls sending and receiving of schedule information among several systems.
Dependencies	PlantMessagingObjects	Assembly containing the schemas used by the plant orchestrations

Technical Architecture Requirements

Implementing BizTalk Server 2006 requires significant architecture planning. Performance and scalability metrics have changed between BizTalk Server 2002 and 2006, and require you to rethink your infrastructure requirements. In addition, you should define your requirements around security, availability, scalability, performance, flexibility, manageability, and usability. After defining your architecture requirements, you should create a plan for meeting them.

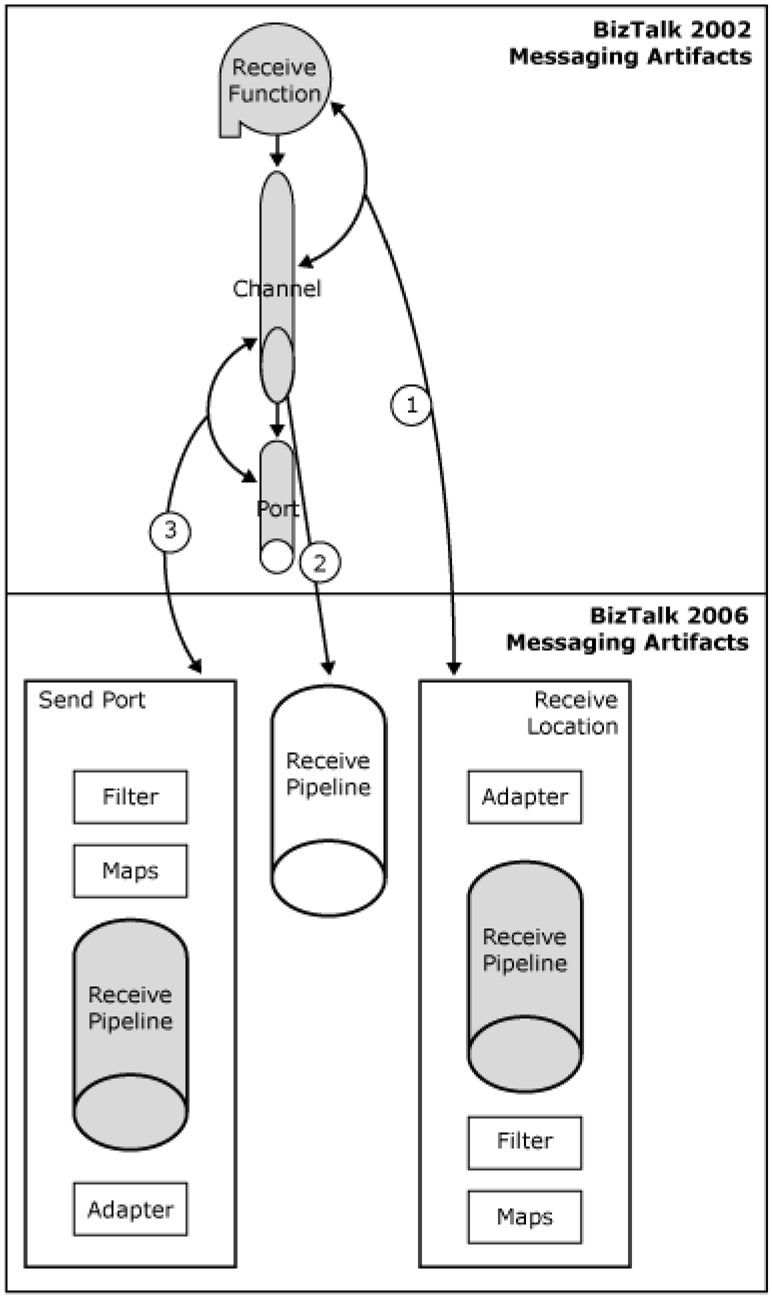
Defining your technical architecture requirements is outside of the scope this topic, but you can peruse the BizTalk Server 2006 Web site for additional information to create an architecture plan before implementing your migration. For more information about BizTalk Server architecture requirements, see http://go.microsoft.com/fwlink/?LinkId=25487.

The Migration Wizard

The topics in this section provide background information about the Migration Wizard. After planning your migration activities and setting up your BizTalk Server 2006 environment based on the technical architecture plan, you can use the Migration Wizard to migrate your BizTalk Server 2002 components to the BizTalk Server 2006 environment.

The following diagram depicts how the Migration Wizard migrates messaging items from BizTalk Server 2002 to BizTalk Server 2006.

The Migration Wizard migrates items from BizTalk Server 2002 to BizTalk Server 2006



The Migration Wizard uses Microsoft Windows integrated security to connect to the specified BizTalk MessageBox database. Therefore, the Windows user that is logged in must have the appropriate permissions to access the specified database.

If there is a custom pre-process configured on the receive function, the Migration Wizard does not automatically migrate it.

Channels that use dynamic routing do not automatically have their routing configurations migrated. You must manually configure them in the properties of the send port or send port group being migrated.

Channels that use filters do not automatically have their filtering configurations migrated. You must manually configure them in the properties of the migrated send port or send port group.

The Migration Wizard does not migrate scripting functoids used in BizTalk Server 2002 maps. They must be rewritten using one of the .NET Framework languages such as Microsoft Visual Basic .NET, Visual C#, and JScript .NET in BizTalk Server 2006, rather than legacy Visual Basic Scripting Edition (VBScript) and JScript scripting engines.

When you migrate BizTalk Server 2002 maps containing scripting functoids with custom VBScript code, the Migration Wizard copies the code to the Inline Script Buffer window in the BizTalk Server 2006 scripting functoid. You must then manually change and migrate that code to equivalent Visual Basic .NET or other managed code (C#, JScript .NET, XSLT). For simple VBScript code being migrated, you must add the System.VisualBasic namespace in front of specialized Visual Basic-only functions such as Randomize(), CInt(), and CStr().

For more complex or lengthy VBScript functoid code, Visual Studio 2005 includes a tool which can quickly migrate larger amounts of VBScript code to Visual Basic .NET. This tool, the Upgrade Visual Basic 6 Code, resides under the Tools menu. To see this Tools menu choice, you must add a new or existing Visual Basic .NET project (Windows Application, Class Library, Web Application or Web Services) to your solution and open the Code View of one of the *.vb files in the project.

You can then open the Visual Basic to Visual Basic .NET code conversion window by pointing to **Tools** and selecting **Upgrade Visual Basic 6 Code**. Copy your old VBScript code from the Inline Script Buffer window in the BizTalk Server 2006 scripting functoid to this code conversion window. Add any appropriate COM object references used in the code. Convert the code by clicking the **Convert** button. You can then paste the converted Visual Basic .NET code into the Inline Script Buffer window of the functoid. You can also move this code into a separate .NET class library or custom functoid project for optimal reusability.

The Migration Wizard converts BizTalk Server 2002 channels into receive pipelines and the corresponding ports into send pipelines. BizTalk Server 2006 uses a publish and subscribe (pub/sub) architecture making it unnecessary to have a send pipeline to publish the data into an orchestration. When the message arrives in the MessageBox database, BizTalk Server 2006 accesses the subscription, and invokes the orchestration which receives the message. You do not need to define a filter in your orchestration to receive a message, but you do need to configure the logical binding for the orchestration.

The filter on the send ports may be more complex than you need, as it specifies filters on Organization and Application names that do not apply to BizTalk

Server 2006. You may need to change or remove this filtering from the appropriate send ports.

After you select the Ports to migrate in the Migration Wizard and click Next, the following error message may appear:

Unable to copy <*xxx*> from the DAV store. Please check if the DAV store is correctly configured and the files exist.

where:

<*xxx*> is the URI for the item (for example, a schema or map) causing the error.

This error occurs when BizTalk Server 2006 does not have HTTP access to the BizTalk Server 2002 Web Distributed Authoring and Versioning (WebDAV) repository. The channels and file receive functions being migrated contain URL links to schemas (document definitions and envelopes) and maps stored in a WebDAV repository. BizTalk Server 2002 does not store these schemas and maps in the InterchangeBTM database. These error messages appear if the WebDAV Internet Information Services (IIS) server is not on the network and IIS is stopped on that computer, or you have insufficient security permissions to read documents using WebDAV.

Because the error provided by the Migration Wizard does not identify the specific schemas or maps it is unable to locate in WebDAV, you can determine these URLs by viewing all rows of data in the following InterchangeBTM table.

BTM Object	**Table**	**WebDAV Column/Other Important Columns**
Channels	bts_channel	Mapref (contains HTTP address of BizTalk Server 2002 maps)/inpdocid & outdocid (unique ID of document schema in the following rows)
Schemas	bts_xmlshares	Reference (contains http address of BizTalk Server 2002 schemas)
Document Definitions	bts_document	None but references preceding rows using foreign key shareid references bts_xmlshare.id ID & name (match with bts_channel.inpdocid and bts_channel.outdocid columns)
Envelopes	bts_envelope	None but references preceding tables via foreign key shareid references bts_xmlshare.id ID & name (match with bts_port.envid

		-or- adm_ReceiveService.DocumentName -or- adm_ReceiveService.EnvelopeName)

The following two options help resolve this error and enable you to proceed with the BizTalk Server 2006 migration:

- Get access to the WebDAV server being referenced in the preceding WebDAV columns.
- Obtain a copy of the BizTalk Server 2002 schema and map files. Create an IIS virtual directory named BizTalkServerRepository on your BizTalk Server 2006 computer. Create two subfolders in this directory named DocSpecs and Maps. Put the schemas files in the DocSpecs directory and the map files in Maps directory. Open the InterchangeBTM database and change the appropriate WebDAV map URL references (bts_channel.mapref) and schema URL references (bts_xmlshare.reference) to point to your local http://localhost/BizTalkServerRepository virtual directory. Make sure the URL paths to these schemas and maps are correct by testing some of the URLs in your browser.

In This Section

- Receive Function and Channel Migration
- Channel Migration
- Channel and Port Combination Migration

Receive Function and Channel Migration

The Migration Wizard migrates every BizTalk Server 2002 receive function to a BizTalk Server 2006 receive location. During this process, the Migration Wizard creates the following items:

- Binding file (defines receive location). When migrating messages, the Migration Wizard only generates one binding file for the receive location in a solution.
- BizTalk Server 2006 maps and schemas (XML Schema Definition language (XSD) files)
- Receive pipeline

If the BizTalk Server 2002 receive function is statically bound to a channel, the receive pipeline name becomes a concatenation of the receive function name and the channel name. The receive pipeline created during migration adds any migrated schemas in the original channel to the schema collection of the validation stage. The binding file creates the additional information necessary to generate the receive location. When you deploy the migration assembly with the binding file, the receive location appears with the following properties configured:

- Adapters are set to the transport type of the migrated port (for example, SMTP, HTTP, FILE, MSMQ)
- Any maps in the channel are added to the maps collection
- Receive pipelines created are selected as the receive locations pipeline

The Migration Wizard converts BizTalk Server 2002 channels into receive pipelines and the corresponding ports into send pipelines. It is unnecessary, however, to have a unique receive and send pipeline for each migrated channel and port combination. In fact, you can create a single receive and send pipeline and set several XML schemas into the XML Validators.

Instead of having a distinct receive or send location for each URI and map combination (created by default when you run the Migration Wizard), BizTalk Server 2006 enables you to select multiple maps into a collection within a send or receive port. At run time, the messaging engine resolves the incoming instance message to the schema type by using the root node of the instance and, based on the schema type, it dynamically selects a map to use for transformation. The same process is repeated in outbound ports and pipelines. Therefore, you can consolidate any send or receive ports with the same URI that differ by only maps and specifications.

Channel Migration

The Migration Wizard migrates every BizTalk Server 2002 channel to a BizTalk Server 2006 receive pipeline. The wizard creates the following items:

- BizTalk Server 2006 maps and schemas (XSDs)
- Receive pipeline

You must manually add any migrated schemas in the original channel to the schema collection during the validation stage. The channel migration does not add to or create a binding file. In a partial self-routing scenario, in which you select only some channels, you can use these pipelines that you migrate from the channels. You can manually add the pipelines into a solution that you migrate later.

Channel and Port Combination Migration

The Migration Wizard migrates every BizTalk Server 2002 channel and port combination to a BizTalk Server 2006 send port. The wizard creates the following items:

- Binding file (defines send port). The Migration Wizard only generates one binding file for the send port location in a solution.
- BizTalk Server 2006 maps and schemas (XSDs)
- Send pipelines

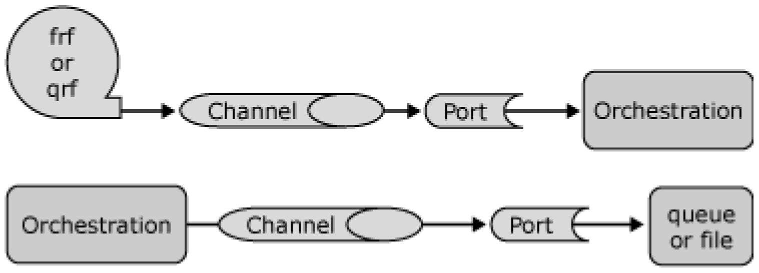
The binding file creates the additional configuration information necessary to generate the send port. When you deploy the migration assembly with the binding file, the send port appears with the following properties configured:

- The adapter is set to the transport type of the migrated port (for example, SMTP, HTTP, FILE, MSMQ).
- Any maps in the channel are added to the maps collection.
- The send pipeline created is selected as the send port pipeline.

Migrating Messages

The following migration scenario covers common message patterns in BizTalk Server 2002 solutions.

Receive handling / Send ports and Orchestration



A typical messaging scenario in BizTalk Server 2002 involves processing a message through a file receive function (frf) or a queue receive function (qrf) and submitting it to a messaging channel. This channel leads to a messaging port and an orchestration. The orchestration calls one or more channels that lead to ports bound to queue or file locations.

The topics in this section explain how to migrate messages in a typical messaging scenario.

In This Section

- Reviewing BizTalk Server 2002 Inventory
- How to Create a Message Migration Project
- Updating a Message Migration Project
- How to Build and Deploy a Message Migration Project

Reviewing BizTalk Server 2002 Inventory

Review the list of items that must be migrated.

Assembly Name: PlantMessaging

BizTalk Object	**Name**	**Purpose**
File Receive function	frfSCADAReceive	Processes energy meter usage data in XML format delivered directly from external sources via FTP.
Channel	chlSCADAReceive	Validates an XML message against raw SCADA schema, and maps the XML into a format more conducive for database inserts.
	chlSCADALoadReceivedInfo	Maps the raw SCADA schema to a trigger document.
Port	prtSCADAReceive	Delivers an energy meter usage XML message to the orchestration for further processing and entry into PowerBiz application.
	prtSCADALoadReceivedInfo	Delivers a trigger document to a queue that is monitored by a client application.
Map	mapSCADARawToSql.xml	Maps a raw meter document to a document format that is easier to parse and process.
	mapSCADASQLToLoadReceivedInfo.xml	Maps a raw meter document to a document that is placed in a

		queue for notifying a client application that new data has arrived.

How to Create a Message Migration Project

Create a migration project for each functional area and use the Migration Wizard to migrate the messaging items and schemas. Use the Migration Wizard provided with BizTalk Server 2006 to migrate the following messaging objects:

BizTalk Server 2002 Messaging Item	**BizTalk Server 2006 Migration Path**
Document definitions	Schemas
Maps	Maps
Receive functions (FILE, HTTP, MSMQ)	Receive locations
Ports (SMTP, HTTP, MSMQ, File) and channels	Ports and pipelines
Port groups (distribution lists)	Send port groups

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Microsoft Visual Studio 2005 must be installed.

To create a message migration project

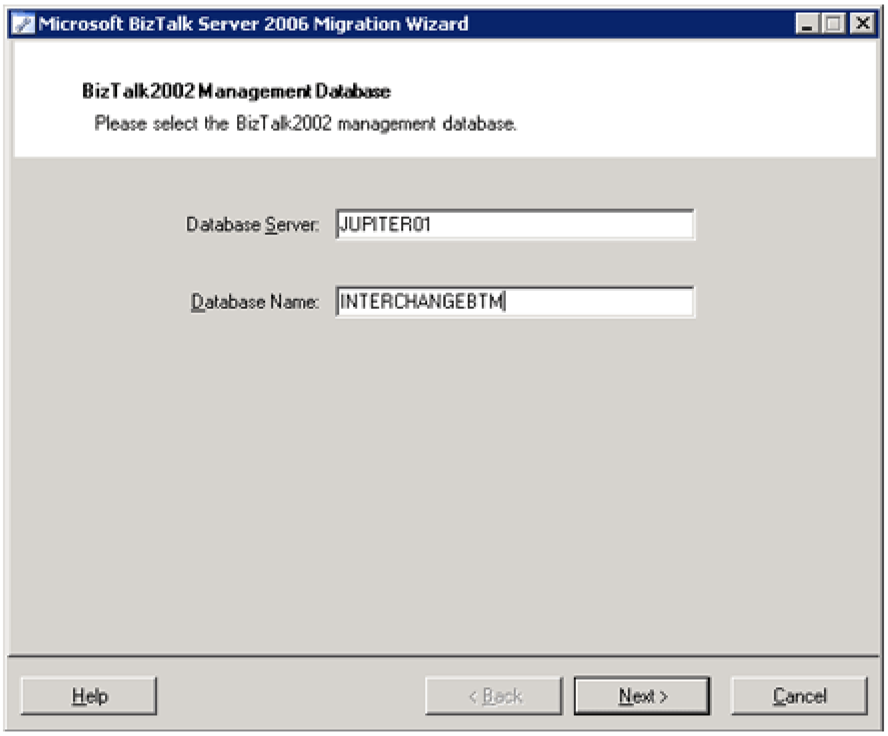
1. Create a new BizTalk Server 2006 migration project in Visual Studio 2005, as described in Creating BizTalk Projects. This starts the Migration Wizard.

2. On the **Welcome to the BizTalk Migration Wizard** page, click **Next**. The BizTalk 2002 Management Database page appears.

3. On the **BizTalk 2002 Management Database** page, enter a Database Server and Database Name where the migration objects are located, and then click **Next**.

4. The Migration Wizard attempts to connect to the specified messaging management data source.

5. **Enter a Database Server and Database Name**



BizTalk2002 Management Database

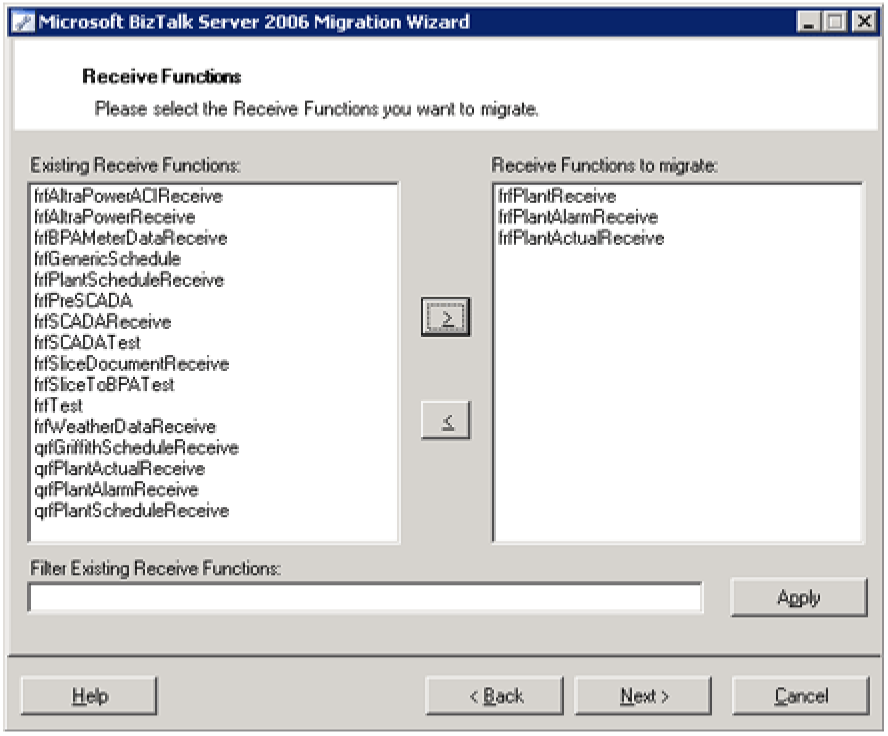
Please select the BizTalk2002 management database.

Database Server: JUPITER01

Database Name: INTERCHANGEBTM

7. On the **Receive Functions** page, specify the receive functions to include in the migration project, and then click **Next**.

Specifying receive functions


Microsoft BizTalk Server 2006 Migration Wizard

Receive Functions

Please select the Receive Functions you want to migrate.

Existing Receive Functions:

frfAltraPowerACIReceive
frfAltraPowerReceive
frfBPAMeterDataReceive
frfGenericSchedule
frfPlantScheduleReceive
frfPreSCADA
frfSCADAReceive
frfSCADATest
frfSliceDocumentReceive
frfSliceToBPATest
frfTest
frfWeatherDataReceive
qrfGriffithScheduleReceive
qrfPlantActualReceive
qrfPlantAlarmReceive
qrfPlantScheduleReceive

Receive Functions to migrate:

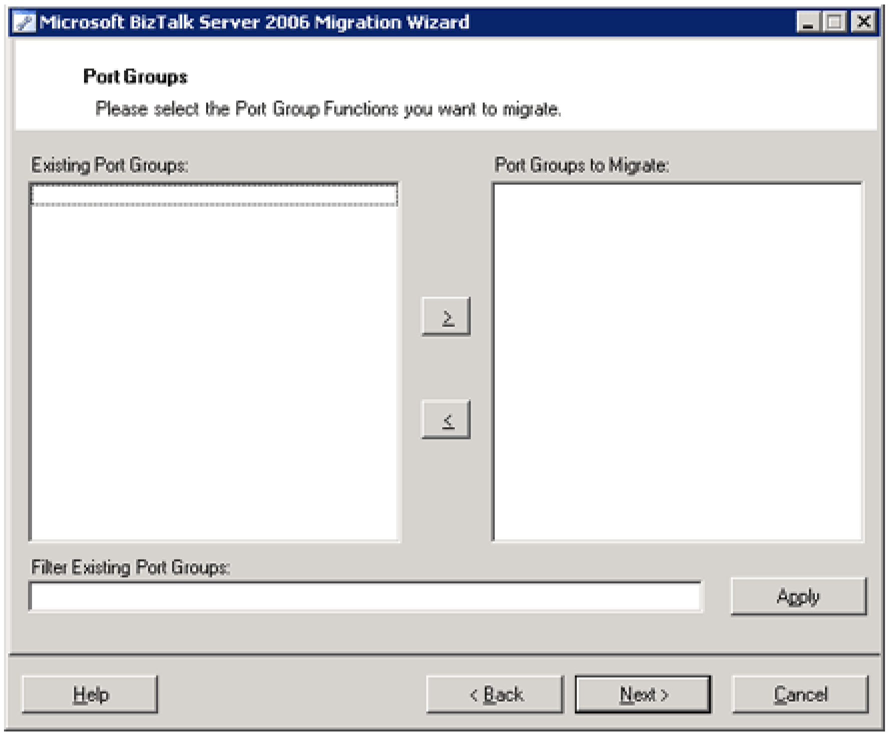
frfPlantReceive
frfPlantAlarmReceive
frfPlantActualReceive

Filter Existing Receive Functions:

If the BizTalk MessageBox database includes a large number of receive functions, you can apply a filter to the receive function name to more easily select specific functions. For example, if you only want to browse file receive functions, you can enter frf* in the Filter Existing Receive Functions field, and then click **Apply**.

8. On the **Port Groups** page, specify which send port groups to include in the migration project, and then click **Next**.

Select the Port Groups you want to migrate



Microsoft BizTalk Server 2006 Migration Wizard

Port Groups

Please select the Port Group Functions you want to migrate.

Existing Port Groups:

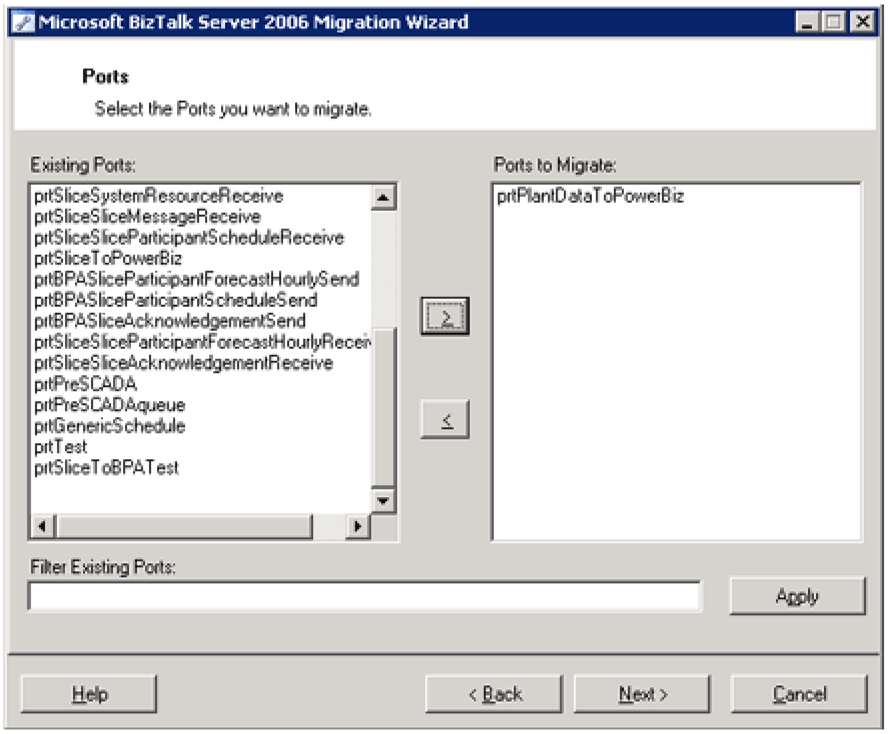
Port Groups to Migrate:

Filter Existing Port Groups:

You can filter send port groups based on the port name. Enter the wildcard character '*' as text into the **Filter Existing Port Groups** field, and then click **Apply**. The Migration Wizard only supports the wildcard character '*'. The text is the partial name of the port.

9. On the **Ports** page, specify the ports to include in your migration project. You can filter based on the port name. Enter text into the **Filter Existing Ports** field, and then click **Apply**. After specifying the ports, click **Next**.

Select the Ports you want to migrate



Ports

Select the Ports you want to migrate.

Existing Ports:

prtSliceSystemResourceReceive
prtSliceSliceMessageReceive
prtSliceSliceParticipantScheduleReceive
prtSliceToPowerBiz
prtBPASliceParticipantForecastHourlySend
prtBPASliceParticipantScheduleSend
prtBPASliceAcknowledgementSend
prtSliceSliceParticipantForecastHourlyRecei
prtSliceSliceAcknowledgementReceive
prtPreSCADA
prtPreSCADAqueue
prtGenericSchedule
prtTest
prtSliceToBPATest

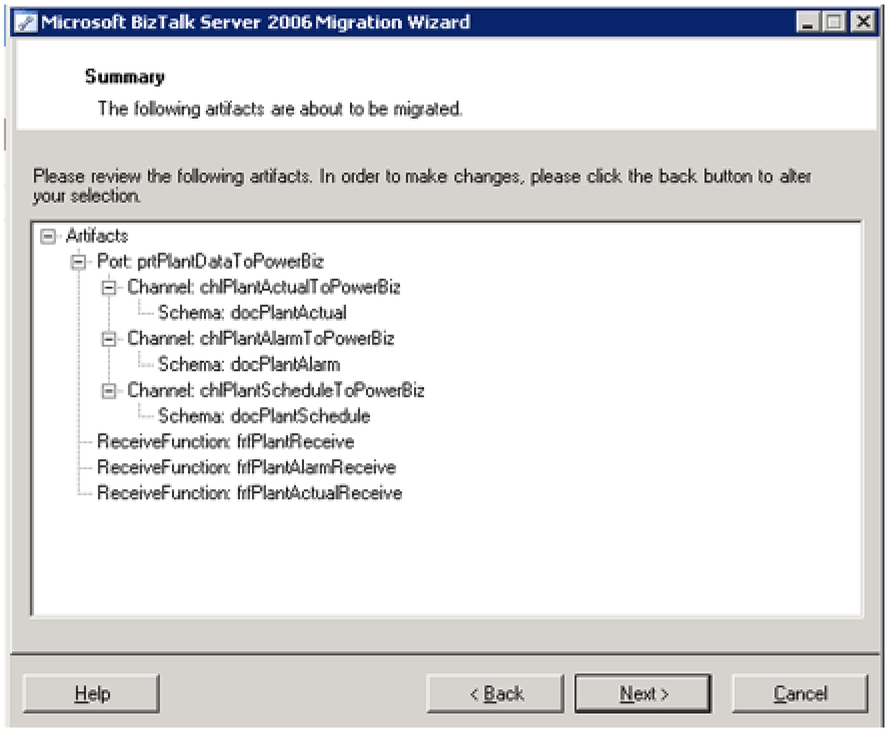
Ports to Migrate:

prtPlantDataToPowerBiz

Filter Existing Ports:

10. After you have selected all the required ports, the **Summary** page displays a summary of all the objects you included in the migration project. Click **Next**.

Summary of all the objects that have been included in the migration



Summary

The following artifacts are about to be migrated.

Please review the following artifacts. In order to make changes, please click the back button to alter your selection.

```
Artifacts
  Port: prtPlantDataToPowerBiz
    Channel: chlPlantActualToPowerBiz
      Schema: docPlantActual
    Channel: chlPlantAlarmToPowerBiz
      Schema: docPlantAlarm
    Channel: chlPlantScheduleToPowerBiz
      Schema: docPlantSchedule
  ReceiveFunction: frlPlantReceive
  ReceiveFunction: frlPlantAlarmReceive
  ReceiveFunction: frlPlantActualReceive
```

The Migration Wizard creates the migration project. The message migration project contains all the objects appearing in the Summary page.

11. On the **Completing the BizTalk Deployment Wizard** page, click **Finish** after successfully creating the project.

In the example, channel information for the selected ports is automatically migrated to the Send and Receive Pipeline objects. In addition, the project includes a bindingout.xml file in the Artifacts folder. This file contains the bindings necessary to create the new ports and subscriptions in BizTalk Server 2006, and deploys the new messaging objects.

Updating a Message Migration Project

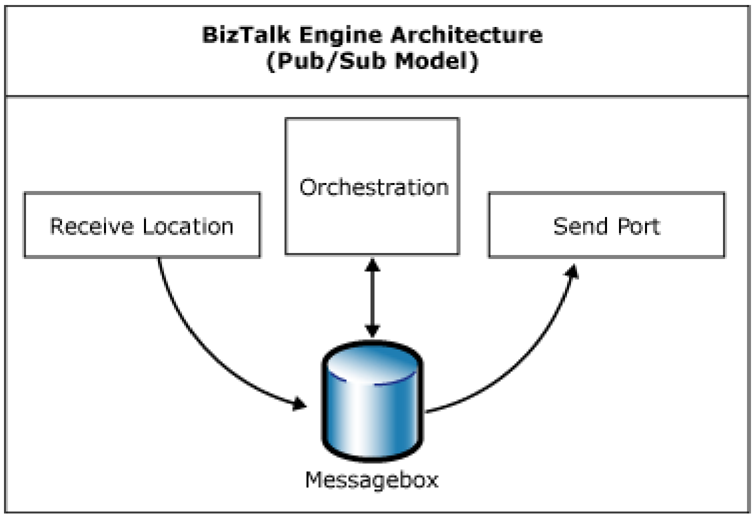
After the Migration Wizard creates your migration project, you must update the project to account for:

- Settings that are not transferred by the Migration Wizard
- Components that are not migrated by the Migration Wizard, such as orchestrations, application integration components (AICs), and pre-processors.

In addition, you can update your migration project to leverage some of the new BizTalk Server 2006 features. After you complete the project updates, you must build the solution and deploy it to a BizTalk Server 2006 environment.

BizTalk Server 2006 uses a publication and subscription (pub/sub) architecture based on the MessageBox database. When a receive port picks up a message, it disassembles the message and stores pre-existing and custom metadata, and the message itself, in the MessageBox database. BizTalk Server 2006 also configures custom metadata for storage in the MessageBox database by property promotion. This process enables you to flag schema nodes at design time for promotion to the MessageBox database at run time. In addition to the custom metadata you explicitly define by promoting properties, BizTalk Server 2006 has several pre-existing properties that it tracks by default.

BizTalk Server 2006 Pub/Sub Model



The pub/sub architecture in BizTalk Server 2006, in conjunction with filter properties, replaces the channel filters and dynamic routing found in BizTalk Server 2002. BizTalk Server 2006 adds property fields to the context data and made available at run time for filtering and other activities. Distinguished fields are only made available within the given orchestration and are not persisted outside of the orchestration itself.

When you configure a filter property on an item, BizTalk Server 2006 uses the pre-existing and custom context properties to create subscriptions to the message. A filter property is a Boolean expression based on one or more property fields. For example, you may create a send port filter that subscribes to all messages with a receive location property of "C:\temp\*.xml." This filter instructs BizTalk Server 2006 to forward all messages to your item (for example, a send port or orchestration) upon receiving messages with a context property that matches the receive location filter. It is not necessary to set filter ports on orchestrations if you have already bound a logical port to a send or receive port. The act of binding automatically creates the subscription to the message for you.

BizTalk Server 2002 allows filtering based on an xpath (World Wide Web Consortium (W3C) standard of a path-based XML query language) evaluation against a schema. BizTalk Server 2006 does not support generic xpath evaluation. The BizTalk Server 2006 environment only supports a limited subset of operators and does not allow the promotion of a repeating element.

The Migration Wizard does not migrate any custom channel filters, other than the default content-based routing (CBR) scenario. If you configure custom channel filters, you must manually migrate those filters to the send or receive ports. CBR scenarios, however, automatically promote the five required fields (source identifier, source ID, destination identifier, destination ID, and document type) and configure the necessary filters.

Given the change of architecture in BizTalk Server 2006, you only need a receive location to publish a message to an orchestration. Orchestrations can send and receive messages to and from the MessageBox database directly. Unfortunately, the Migration Wizard does not interpret whether the receive function, channel, and port combination is going to an orchestration or a send port. It therefore creates a receive location, send port, and an extra receive pipeline. You can delete the extra send port.

You can make various updates to the messaging objects that were migrated. For an explanation of the Migration Wizard's limitations, see Using the Migration Wizard to Create a Message Migration Project. If you encounter any of the following migration upgrade circumstances, (for example, channel filtering, dynamic routing on a channel, schema validation on a channel, or queue receive function and message queue migration), address these tasks manually through the BizTalk Server Administration Console and BizTalk Explorer in Visual Studio 2005.

This section contains:

- Migrating Legacy Properties for Channel Filtering
- Migrating Legacy Properties for Dynamic Routing
- Validating Schemas on a Channel
- Queue Receive Functions and Message Queues
- Using Macros for Output File Names in Send Ports
- Applying Maps to Send Ports and Send Port Groups

Migrating Legacy Properties for Channel Filtering

BizTalk Server 2006 does not migrate any channel filters configured in BizTalk Server 2002. To migrate them, the appropriate properties must be promoted in the migration project, despite several inherent limitations. For more information about accomplishing this task, see Copying Data to the Message Context as Property Fields. After the necessary properties have been promoted, you must configure the appropriate send port or send port group in BizTalk Explorer for filtering based on those properties. Specifically, you can configure a send port or send port group to filter the promoted schema properties based on any applicable value.

How to Migrate Legacy Properties for Dynamic Routing

In BizTalk Server 2002, dynamic routing on a channel is based on a channel name or a combination of the following five routing properties: source identifier, source ID, destination identifier, destination ID, and document type. The Migration Wizard automatically promotes these five properties and sets the property filter in the corresponding send port. The Migration Wizard does not support channel filter expressions, open source or open destination items.

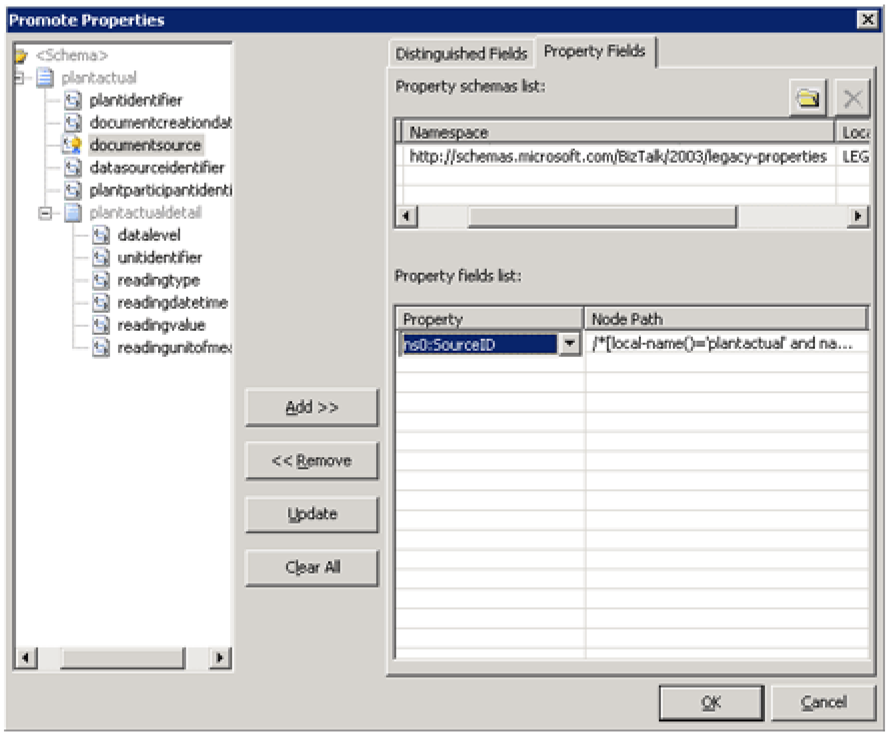
Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To promote properties from the Schema Editor

1. In Solution Explorer, double-click the schema to open it.
2. Select the **<Schema>** node.
3. In the **Properties** window, find the **Promote Properties** property and click the ellipsis (**...**) button.

   The properties used in dynamic routing are automatically promoted, and filters are created



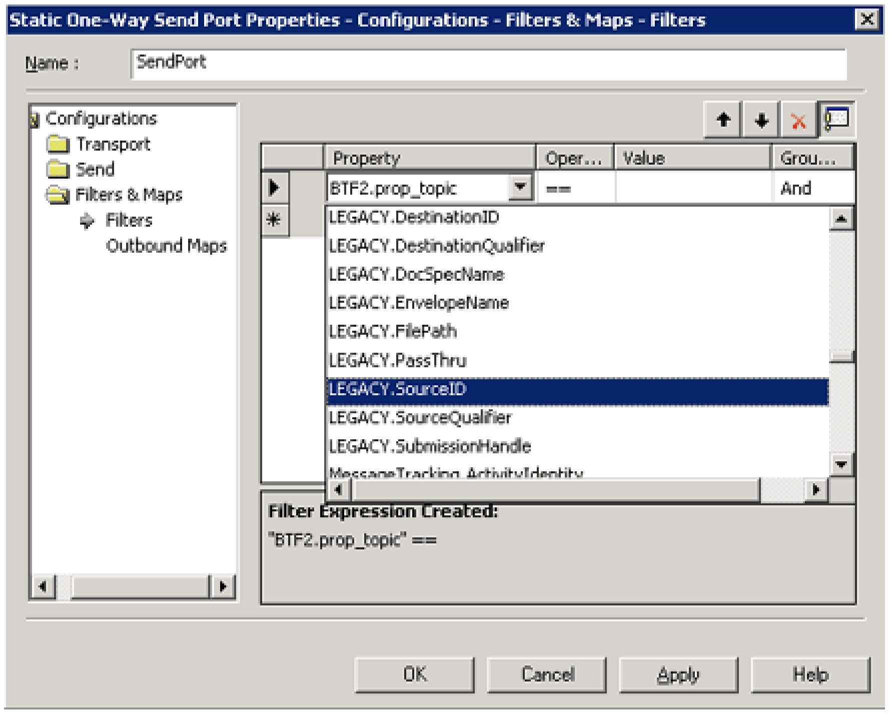
To configure a send port or send port group to filter the promoted schema properties

1. In BizTalk Explorer, select the port where the filters are used.

2. From the context menu, select **Properties**.

3. From the navigation tree pane, expand the **Configurations** folder, expand the **Filters & Maps** folder, and open **Filters**.

Dynamic Routing on Channel


Static One-Way Send Port Properties - Configurations - Filters & Maps - Filters

Name : SendPort

Property	Oper...	Value	Grou...
BTF2.prop_topic	==		And

LEGACY.DestinationID
LEGACY.DestinationQualifier
LEGACY.DocSpecName
LEGACY.EnvelopeName
LEGACY.FilePath
LEGACY.PassThru
LEGACY.SourceID
LEGACY.SourceQualifier
LEGACY.SubmissionHandle
MessageTracking.ActivityIdentity

Filter Expression Created:
"BTF2.prop_topic" ==

How to Validate Schemas on a Channel

Schemas must have a unique root node/namespace combination for the pipeline to resolve the schema. Because the Migration Wizard places all schemas into the same namespace, ensure that the migrated schemas have different root nodes, or alter the namespace after BizTalk Server 2006 migration completes.

Some flat file schema defined in Biz Talk Server 2002 may not migrate properly to BizTalk Server 2006 because XDR and XSD schemas are not fully compatible (for example, flat file XDR schema). If you migrate a receive pipeline created from a flat file, self-routing receive function in BizTalk Server 2002, BizTalk Server 2006 cannot identify the flat file schema for this receive pipeline. You can manually fix this issue if you build all the run time documents with the same schema.

Migrated schemas do not work in run time because BizTalk Server 2002 uses the XML Data-Reduced (XDR) schema standard whereas BizTalk Server 2006 uses the XML Schema Definition language (XSD) schema standard. A fundamental ambiguity exists when converting numbered data types to float values.

To resolve this issue, perform one of the following options:

- Use the standard XDR-XSD style sheet converter published by Microsoft which converts map Number data type to xs:float.

- If necessary, change the data type in the schema based on validating instances in the Schema Editor).
- Manually adjust the schemas (change the data type to xs:string in order to accept blank inputs) after migration.
- Turn off validation or reject documents which have blank input.

The Migration Wizard does not automatically configure schema validation for your send and receive pipelines. Perform the following steps to configure schema validation for your send and receive pipelines.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To configure the receive pipeline for validation

1. In BizTalk Explorer, open the **Receive Pipeline** and click the **XML Validator** in the Validator state.
2. In the **Properties** window, select the **(Collections)** from the Document Schema property.
3. Choose one or more schemas from the **Available Schemas** pane, and then click **Add**.
4. When you are finished, click **OK**.

To configure the send pipeline for validation

1. In BizTalk Explorer, open the **Send Pipeline** and drag an **XML Validator** shape from the Toolbox to the Pre-Assembly stage.
2. Click the **XML Validator**.
3. Choose one or more schemas from the **Available Schemas** pane, and then click **Add**.
4. When you are finished, click **OK**.

Queue Receive Functions and Message Queues

The Migration Wizard does not automatically migrate queue receive functions. For every queue receive function, the Migration Wizard creates a receive port using an MSMQT transport type.

Using Macros for Output File Names in Send Ports

BizTalk Server 2002 and BizTalk Server 2006 use different sets of keywords to insert dynamic text into output file names in send ports. For example, the Migration Wizard automatically replaces %tracking_id% with %MessageID%. For more information about macro keywords, see Restrictions on Using Macros in File Names.

How to Apply Maps to Send Ports and Send Port Groups

The Migration Wizard migrates maps to your send ports. If you want to add more maps, follow these steps.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To apply maps to send ports and send port groups

1. In BizTalk Explorer, double-click the send port you want to configure.
2. Click the **Filters and Maps** folder.
3. Click **Outbound Maps**.
4. Click **Map to Apply** and choose a map from the list.
5. Click **OK**.

How to Build and Deploy a Message Migration Project

This topic describes how to build and deploy your message migration project into BizTalk Server 2006.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To build and deploy a message migration project

1. Create a strong name assembly key file and associate it with your migration project, as described in How to Configure a Strong Name Assembly Key File.

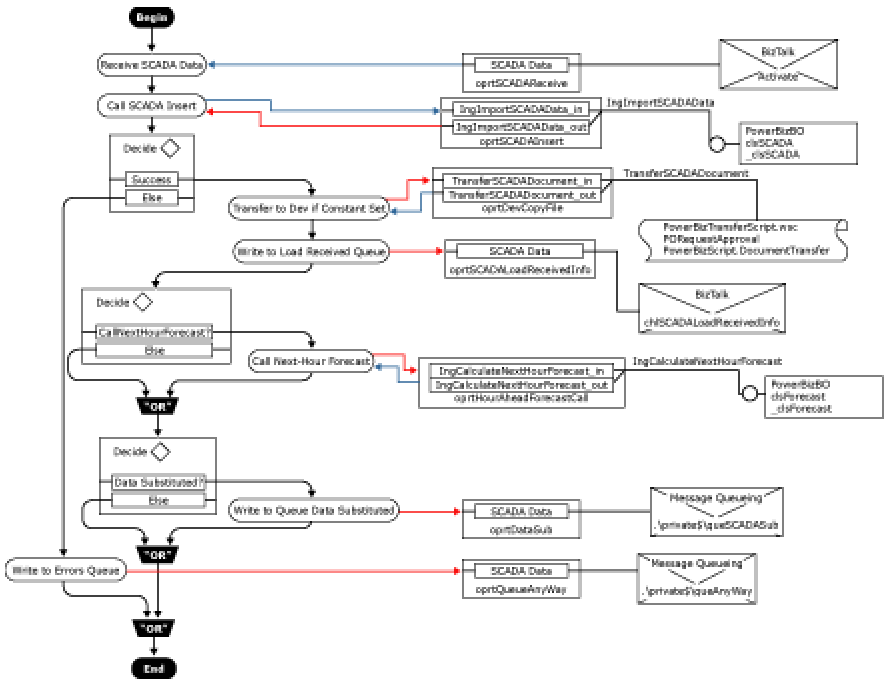
2. Build and deploy the project, as described in How to Deploy a BizTalk Assembly from Visual Studio. If any errors occur, correct them and then re-attempt the deployment.

Migrating Orchestrations

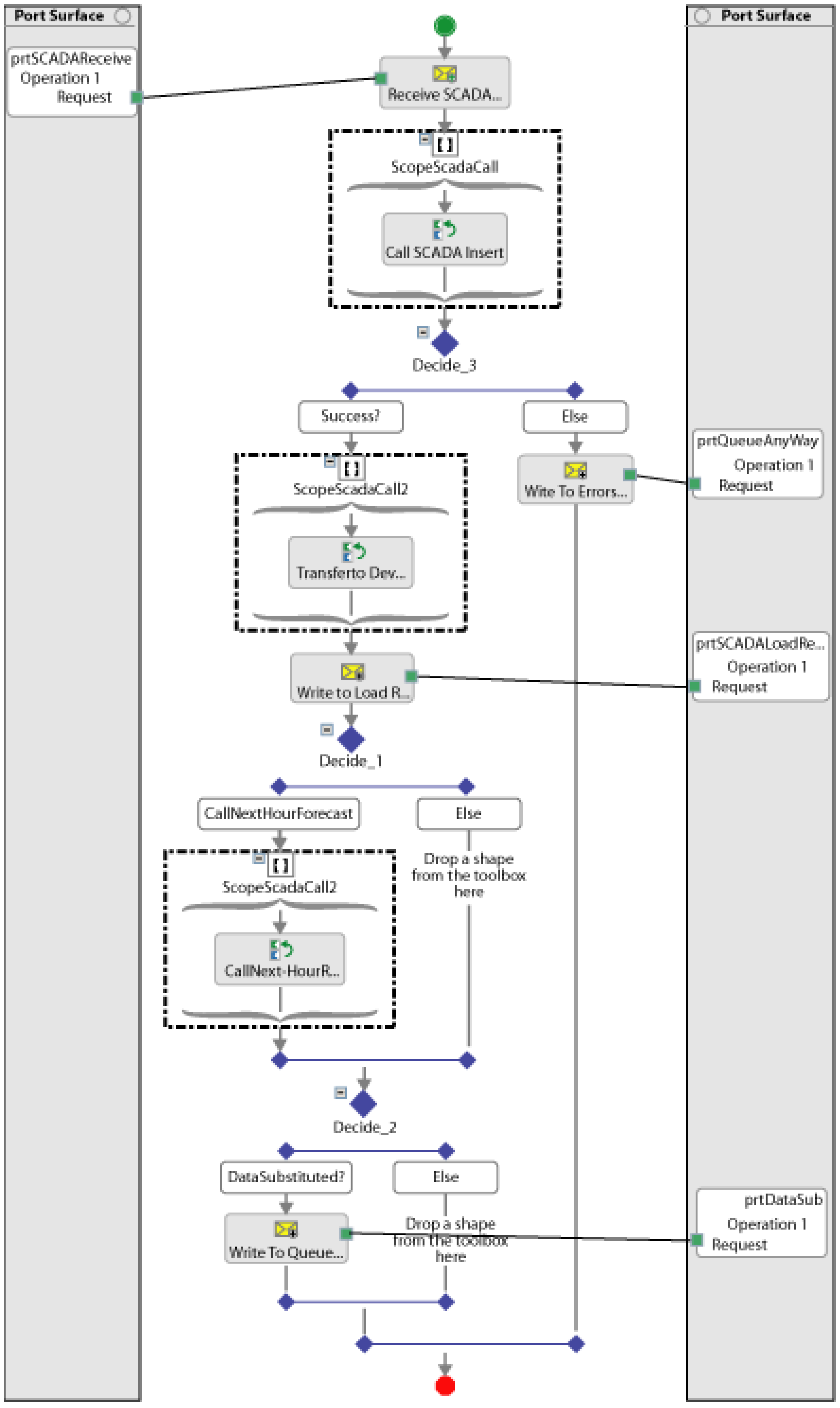
You must manually migrate BizTalk Server 2002 orchestrations. This section describes how to analyze a source BizTalk Server 2002 orchestration and migrate its functionality to the destination BizTalk Server 2006 orchestration.

The following diagrams show comparisons of BizTalk Server 2002 and BizTalk Server 2006 orchestrations, respectively.

BizTalk Server 2002 Orchestration



BizTalk Server 2006 Orchestration



The topics in this section describe the following steps you must take to migrate an orchestration:

1. Create a BizTalk Server Orchestration Project.
2. Create the orchestration items.
3. Define the schemas in the orchestration.
4. Migrate the orchestration implementation shapes.
5. Implement the control shapes.
6. Bind the orchestration for migration.

Repeat these steps for every orchestration item you identified in BizTalk Server 2002 Component Inventory.

In This Section

- How to Create a BizTalk Server 2006 Orchestration Project
- How to Create the Orchestration Items
- How to Define the Schemas in an Orchestration
- Migrating Orchestration Implementation Shapes
- How to Implement the Control Shapes
- How to Bind an Orchestration for Migration

How to Create a BizTalk Server 2006 Orchestration Project

Start the orchestration migration process by creating a new BizTalk project for orchestrations and add an orchestration item.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To create a BizTalk Server 2006 Orchestration Project

1. In Visual Studio Solution Explorer, right-click the Solution, select **Add**, and click **New Project**.

2. In the **Add New Project** dialog box, select **Empty BizTalk Server Project** from the **Templates** area.

3. In the **Name** field, enter a name for the project, and then click **OK**.

4. Assign a strong name key to the project.

5. Expand the project you just created, right-click the **References** folder, and click **Add Reference**.

6. Click the **Projects** tab.

7. Click the **Messaging Migration project** you created as part of messaging migration process.

8. Click **Select**, and then click **OK**.

How to Create the Orchestration Items

Start the migration process by creating a new BizTalk Server 2006 project to house your migrated orchestrations.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To create the orchestration items

1. Right-click the orchestration project you just created, point to **Add**, and then click **Add New Item**.

2. In the **Add New Item** dialog box, select the **BizTalk Projects Items** folder and **BizTalk Orchestration** from the **Templates** area.

3. In the **Name** field, enter a name for the orchestration.

4. Right-click the Orchestration Surface and choose **Properties**.

5. In the **Properties** window, select **Long Running** from the **Transaction Type** list.

6. Click **Open**.

How to Define the Schemas in an Orchestration

Determine which types of schemas are used within your orchestration from your inventory assessment. For each unique schema, you must define a message within each orchestration with a unique name. After you have added a message name and defined the message type, all future references to the message type are done through the message name. If there are additional required schemas beyond what was migrated when you created your orchestration project, add the schemas to your project or reference a BizTalk assembly that contains those schemas. For more information about creating the orchestration project, see Creating a BizTalk Server 2006 Orchestration Project.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To define the schemas in an orchestration

1. Open the orchestration view for the orchestration you have migrated.
2. Right-click the **Message** folder, and then click **New Message**.
3. Right-click the message just created, and then click **Properties**.
4. In the **Properties** window, set the identifier with a unique message name.
5. Click the **Message Type** list, expand the **Schemas** node and select **<Select from references assembly...>**.
6. Select the **Project** name from the **Current Projects** area.
7. Select the schema **Type** name, and then click **OK**.
8. Repeat this process for all schemas that you use in an orchestration.

Migrating Orchestration Implementation Shapes

The most distinctive change from BizTalk Server 2002 to BizTalk Server 2006 is in the area of orchestrations. The Orchestration Designer tool is no longer hosted in Microsoft Visio 2003; it is included as part of the Visual Studio 2005 design environment, and it contains several new shapes and functions. Because of the changes to the orchestration architecture, there is no orchestration Migration Wizard. The following table outlines the high-level shape and functionality mapping, in addition to explaining some of the new shapes. For more detailed information about these shapes, see Orchestration Shapes.

BizTalk Server 2002 Object	**BizTalk Server 2006 Object**	**Description**
Action	No equivalent	BizTalk Server 2002 used the Action shape as a placeholder for a call out to an implementation shape. BizTalk Server 2006

		has no equivalent, as all shapes include the implementation details and functionality.
Decision	Decide	BizTalk Server 2006 provides nearly identical functionality for the Decision shape.
While	Loop	BizTalk Server 2006 provides nearly identical functionality for the While shape.
Fork/Join	Parallel Actions	BizTalk Server 2006 provides nearly identical functionality for the Fork/Join shapes.
Transaction	Scope	BizTalk Server 2006 uses the Scope shape to configure transactions around specific steps in an orchestration, similar to the Transaction shape in BizTalk Server 2002. The Scope shape also provides a number of additional configuration options. See Orchestration Shapes.
End	End	The End shape already exists on an Orchestration Surface because it is required.
Abort	Terminate/Suspend	BizTalk Server 2006 uses the Terminate and Suspend shapes to stop an orchestration's processing, similar to the Abort shape in BizTalk Server 2002. The Terminate and Suspend shapes also provide additional functionality options. See Orchestration Shapes.
COM Component	Send, Request-Response Port	There are two ways to call a COM component from a BizTalk Server 2006 orchestration: wrap the component in a Runtime Callable Wrapper (RCW) and call it from within an orchestration Expression shape; or wrap it in a Web service and call the service from within the orchestration.
Script Component	Send, Request-Response Port	There are two ways to call a Script component from a BizTalk Server 2006 orchestration: wrap the component in an RCW and call it from within an orchestration Expression shape; or wrap it in a Web service and call the service from within the orchestration.
Message Queuing	Send, Send Port	In BizTalk Server 2006, you can configure the Send shape and a send port (which can reference a send pipeline, or send messages

		directly to a BizTalk Message Queue) to behave like Message Queuing shapes in BizTalk Server 2002. The Send shape and send port provide additional configuration options. See Orchestration Shapes.
BizTalk Messaging	Send, Send Port Receive, Receive Port	In BizTalk Server 2006, you can configure the Send and Receive shape and a send and receive port (which can reference a send or receive pipeline) to behave like BizTalk Messaging shapes in BizTalk Server 2002. The Send shape and send port provide a number of additional configuration options. See Orchestration Shapes.
Implementation shapes	Port	Defines where and how messages are transmitted. BizTalk Server 2002 provides some of the functionality available with ports through various implementation shapes.
No equivalent	Role Link	Enables you to create a collection of ports that communicate with any partner that fulfills an abstract role, such as a customer or a shipper. The Business Process Execution Language (BPEL) standard provides this functionality.
No equivalent	Construct Message	BizTalk Server 2002 implicitly constructs messages.
No equivalent	Message Assignment	Enables you to assign messages and message parts.
No equivalent	Expression	Enables you to create any kind of expression, such as setting the value of a variable or calling a method on a .NET component.
SubmitSync call from orchestration.	Transform	Enables you to map the fields in one message to those of another. In BizTalk Server 2002, this is done by calling SubmitSync on a channel.
No equivalent	Call Orchestration	Enables your orchestration to call another orchestration synchronously.
No equivalent	Start Orchestration	Enables your orchestration to call another orchestration asynchronously.

No equivalent	Delay	Enables you to build delays in your orchestration based on a timeout interval.
No equivalent	Listen	Enables your orchestration to conditionally branch depending on the type of message received or the expiration of a timeout.
No equivalent	Group	Enables you to group actions into a single visual unit that can be collapsed or expanded.
No equivalent	Compensate	Enables you to write code to compensate for actions already performed by the orchestration when an error occurs.
No equivalent	Throw Exception	Enables you to explicitly throw an exception in the event of an error.

After you have created an orchestration and added the required schemas, you must migrate the implementation shapes from the BizTalk Server 2002 orchestration to the BizTalk Server 2006 orchestration. There are four implementation shapes available in BizTalk Server 2002: the Message Queuing Shape, the BizTalk Messaging shape, the Windows Script Component shape, and the COM Component shape.

Follow the BizTalk Server 2002 orchestration implementation shapes in a top-down approach, converting and implementing one shape at a time to the BizTalk Server 2006 orchestration. This ensures that you do not miss any shapes. It is also consistent with the linear nature of orchestrations. As you traverse the source BizTalk Server 2002 orchestration, convert each shape as described in the procedures in this section.

In This Section

- How to Create a New Send Port for Message Queues
- How to Migrate BizTalk Messaging Shapes
- How to Migrate Windows Script Components
- How to Migrate COM Components

How to Create a New Send Port for Message Queues

BizTalk Server 2006 does not contain a shape that enables you to post directly to a queue. Instead, it provides a send port that is bound to a queue. For each message queuing shape within your BizTalk Server 2002 orchestration, create a new send port.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To create a new send port

1. In Orchestration Designer, from the **Toolbox**, drag a **Port** shape to the Port Surface.

2. Click **Next** in the **Port Configuration Wizard** dialog box.

3. Enter a name for the port and then click **Next**.

4. Enter a name for the port type. Leave the default setting for all other settings and then click **Next**.

5. In the **Port Binding** dialog box, in the **Port direction of communication** list, select **I'll always be sending messages on this port**.

6. In the Port binding list, select **Specify now**.

7. In the **Transport** list, select **BizTalk message queuing**.

8. In the **URI** field, enter the path to the queue using the following format: msmq://DIRECT=OS:queuepathhere.

9. Select **Microsoft.Biztalk.DefaultPipelines.XmlTransmit** from the **Send pipeline** list.

10. Click **Next** and then click **Finish**.

How to Migrate BizTalk Messaging Shapes

There are two types of BizTalk Messaging shape configurations: Receive and Send. Migrate them as follows.

Receive (Activate): The Migration Wizard has previously migrated the channel and port combination that this shape receives messages from and has converted the items to a receive location and receive pipeline. Because the migration items are not created until you deploy your solution, you must set up ports that do not specify binding, and set the binding after the ports are created during deployment. Follow these steps to migrate the Activate BizTalk Messaging shape.

Prerequisites

To perform the procedures in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To migrate the Activate BizTalk Messaging shape

1. In Orchestration Designer, from the **Toolbox**, drag a **Port** shape to the Port Surface.
2. In the **Port Configuration Wizard** dialog box, click **Next**.
3. Enter a name for the port, and then click **Next**.
4. Enter a name for the port type. Leave the default settings for the remainder of the settings, and then click **Next**.
5. In the **Port Binding** dialog box, in the **Port direction of communication** list, select **I'll always be sending messages on this port**.
6. Select **Specify later** from the **Port binding** list, and then click **Next**.
7. Click **Finish**.

Send: The Migration Wizard has previously migrated the channel and port combination that this shape sends messages to. Because the migration items are not created until you deploy your solution, you must set up ports that do not specify binding, and set the binding after creating the ports during deployment. For each BizTalk Messaging shape that sends messages, perform the following steps to migrate the items.

To migrate each BizTalk Messaging shape that sends messages

1. In Orchestration Designer, from the **Toolbox**, drag a **Port** shape to the Port Surface.
2. In the **Port Configuration Wizard** dialog box, click **Next**.
3. Enter a name for the port, and then click **Next**.
4. Enter a name for the port type. Leave the default settings for the remainder of the settings, and then click **Next**.
5. In the **Port Binding** dialog box, in the **Port direction of communication** list, select **I'll always be sending messages on this port**.
6. Select **Specify later** from the **Port binding** list, and then click **Next**.
7. Click **Finish**.

8. Repeat these steps for each BizTalk Message shape in your BizTalk Server 2002 orchestration that you have configured to send messages.

How to Migrate Windows Script Components

There are three options for migrating Windows scripting objects:

- Use the Expression shape in BizTalk Server 2006 and recode the script in the BizTalk Expression Editor using C with some modified BizTalk language syntax.
- Pull the script out of the Windows Script Component (WSC) and re-code it in a .NET assembly. Call the assembly from a BizTalk Server 2006 orchestration.
- Register and call a classic WSC on a BizTalk Server 2006 computer. The WSC looks like any other COM component. Add a reference to the WSC by performing these steps:

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To add a reference to the Windows Script Component (WSC)

1. Right-click the **BizTalk Server Project** and then select **Add references**.
2. Click the **COM** tab, then click **Browse** for the WSC in the list of registered COM objects on the computer.
3. Click **Select** and then click **OK**. A Runtime Callable Wrapper (RCW) is then constructed for this WSC COM object so you can call it from an expression shape.

How to Migrate COM Components

BizTalk Server 2006 resides on the scalable, secure, and dynamic Microsoft® .NET Framework. Built completely new from the bottom up, it does not natively communicate directly with older technologies, such as Component Object Model (COM). To facilitate .NET to COM interaction, the Framework Class Library (FCL) contains a set of namespaces and tools that greatly simplify calling COM objects from .NET code, and .NET code from COM objects. It does this by using a RCW and COM Callable Wrappers (CCW), respectively. BizTalk Server 2006 uses CCWs to call .NET components from COM; it uses RCWs to call COM components from .NET. After a COM object has been wrapped in a RCW, BizTalk Server 2006 accesses it through a reference to the wrapper. It can then be accessed transparently like any other .NET class. Interoperability classes take care of all the infrastructure requirements and enable the programmer to focus on object instantiation and method invocation. For more information about .NET Framework 2.0, go to the .NET Developer Center at http://go.microsoft.com/fwlink/?LinkId=58548.

Perform the following steps to migrate your COM objects to BizTalk Server 2006.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, Visual Studio 2005 must be installed.

To migrate COM objects to BizTalk Server 2006

1. Install your custom COM components (in either the COM or COM+).
2. Add a reference to the COM component by right-clicking the **BizTalk Server Project** and selecting **Add references**.
3. Click the **COM** tab, browse for the Windows Script Component (WSC) in the list of registered COM objects on your computer. Click **Select** and then click **OK**. A RCW is automatically built for this COM component and signed with the same .snk file used in the BizTalk Server project, OR
4. Open the **Visual Studio® 2005 Command Prompt** in Programs/Microsoft Visual Studio 2005/Visual Studio Tools/Visual Studio 2005 Command Prompt.
5. At the command prompt, type:

   Tlbimp PathToDll /out:pathToDllWithNewFileName /keyfile:mypublickey

   For example, **Tlbimp "C:\Temp\ PowerBizBO.dll" /out: "C:\Temp\ PowerBizBORCW.dll" /keyfile: "C:\Temp\ mykey.snk"**

6. Right-click the **References** folder within your Visual Studio project and select **Add Reference**.
7. In the **Add Reference** dialog box, select **Browse** and navigate to the location where you created the RCW, select the **DLL** file, and then click **Open**.
8. In the **Add Reference** dialog box, click **OK**.
9. In the **Properties** section for both newly added references, ensure that **Copy to Local** is set to **True**.
10. At the command line prompt, type: **gacutil /if pathtomyRCW**

How to Implement the Control Shapes

After you have completed the preliminary steps, you can systematically migrate control shapes from BizTalk Server 2002 to BizTalk Server 2006 shapes. Follow the BizTalk Server 2002 orchestration in a top-down approach, converting and implementing one shape at a time into the BizTalk Server 2006 orchestration. This ensures that you do not miss any shape and is consistent with the linear nature of orchestrations. As you traverse the source BizTalk Server 2002 orchestration, convert each shape as follows.

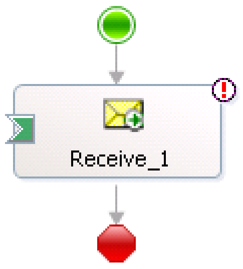
Because actions in BizTalk Server 2002 do not have an equivalent in BizTalk Server 2006, you must reference the implementation port to convert the shape.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

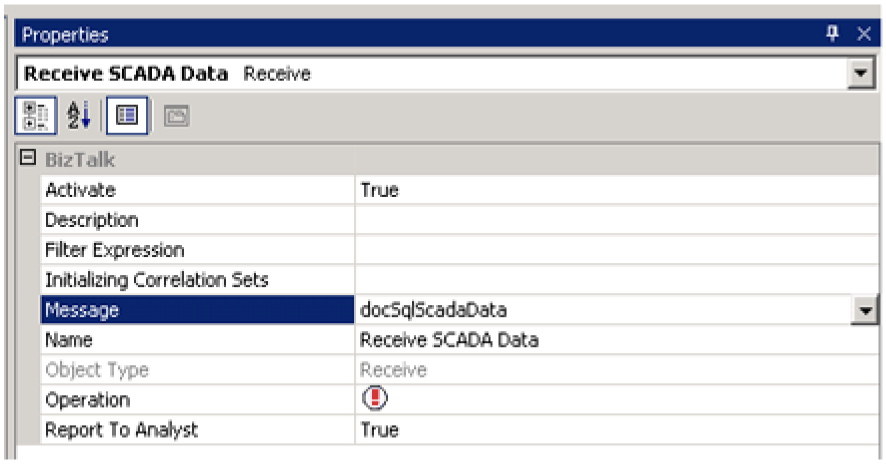
To migrate an action connected to a BizTalk Activate shape

1. Drag a **Receive** shape onto the Orchestration Surface and place it at the top.
2. **Receive shape**



3. Click the **Receive** shape and open the **Properties** window.
4. Enter a name for the shape.
5. Select **True** from the **Activate** list.
6. Select a schema from the **Message** list.

Select a schema from the Message list



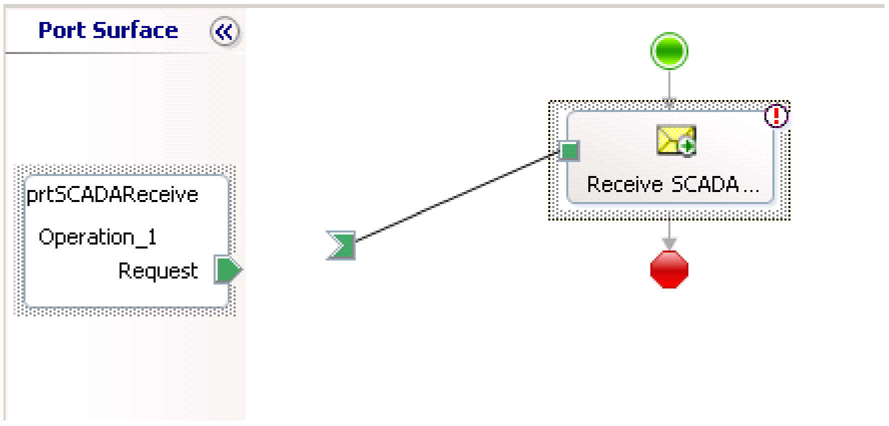
Properties

Receive SCADA Data Receive

BizTalk	
Activate	True
Description	
Filter Expression	
Initializing Correlation Sets	
Message	docSqlScadaData
Name	Receive SCADA Data
Object Type	Receive
Operation	
Report To Analyst	True

7. Connect the **Receive** shape to the **Port** you created for the **Activate** shape.

Connect the Receive shape to the Port you created for the Activate shape



To migrate an action connected to a COM component

1. Drag a **Scope** shape onto the Orchestration Surface and place below the last shape.
2. Click the **Scope** shape and then open the **Properties** window.
3. Enter a name for the **Scope**.
4. Select **Atomic** from the **Transaction Type** list.
5. Drag an **Expression** shape onto the Orchestration Surface and place it inside the **Scope** shape.

6. Click the **Expression** shape and open the **Properties** window.

7. Enter a name for the **Expression**.

8. Open the **Orchestration View**.

9. Expand the **Scope** node you just created.

10. Right-click **Variables** and select **New Variable**.

11. In the **Properties** window, set the **Identifier** to the name by which you reference your object.

12. Select **<.NET Class>** from the **Type** list.

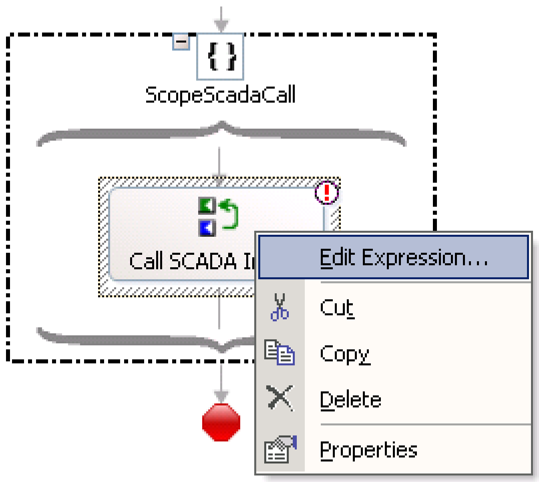
13. In the **References** pane, expand your project name. (It is prefixed with Interop.)

14. In the **Type** pane, select the class you want to use and then click **OK**.

15. Analyze the code you are calling, and determine what is needed to invoke the method. If any ByRef (ref) variables are passed, you must determine the Scope requirement for the variable. If the variable needs to be accessed from elsewhere in the orchestration, you need to define the variables at the orchestration variable level as follows. Otherwise, define the variable in the Scope shape as previously described.

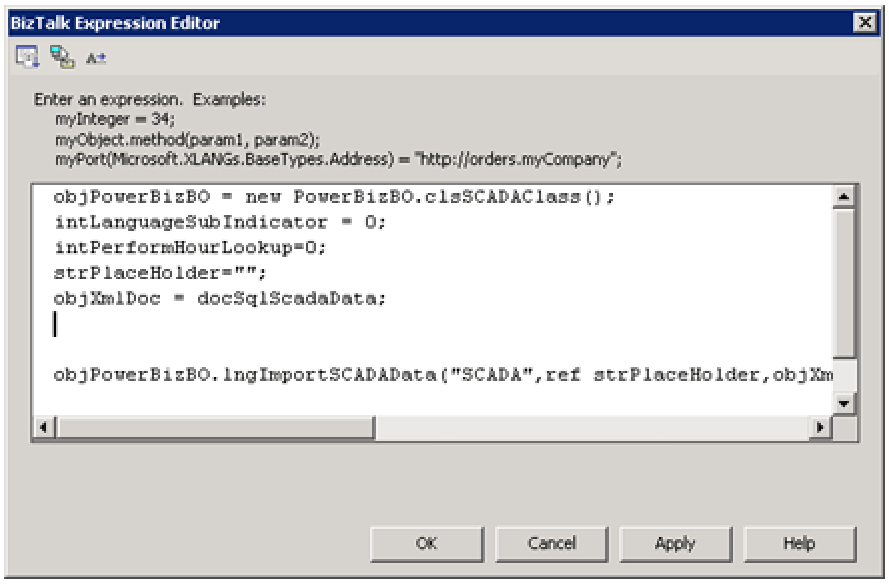
16. Right-click on the **Expression** and select **Edit Expression**.

Right-click on the Expression and select Edit Expression



17. Create code in the BizTalk Expression Editor (see the screen that follows) to instantiate the object and call its method.

19. **Create code to instantiate the object and call its method**


BizTalk Expression Editor
Enter an expression. Examples:
myInteger = 34;
myObject.method(param1, param2);
myPort(Microsoft.XLANGs.BaseTypes.Address) = "http://orders.myCompany";
OK
Cancel
Apply
Help

```
objPowerBizBO = new PowerBizBO.clsSCADAClass();
intLanguageSubIndicator = 0;
intPerformHourLookup=0;
strPlaceHolder="";
objXmlDoc = docSqlScadaData;

objPowerBizBO.lngImportSCADAData("SCADA",ref strPlaceHolder,objXm
```

To migrate an action connected to message queuing

1. In Orchestration Designer, drag a **Send** shape onto the Orchestration Surface and place it at the top.
2. Click the **Receive** shape and open the **Properties** window.
3. Enter a name for the **Receive** action.
4. Select a schema from the **Message** list.
5. Connect the **Send** shape to the **Port** you created for the **Activate** shape.

The following example describes how to convert a Windows Script Component (WSC) to a BizTalk Script. If you choose to convert the script to a .NET component, add it as a .NET reference using the **.NET** tab. Click **Browse** to select the .NET component and then click **Add**.

To migrate an action connected to Windows Script Component

1. Drag a **Scope** shape onto the Orchestration Surface and place it below the last shape.
2. Click the **Scope** shape and open the **Properties** window.
3. Enter a name for the **Scope**.
4. Select **Atomic** from the **Transaction Type** list.

5. Drag an **Expression** shape onto the Orchestration Surface and place it inside the **Scope** shape.

6. Click the **Expression** shape and open the **Properties** window.

7. Enter a name for the **Expression**.

8. Open the **Orchestration View**.

9. Analyze the code you are planning to migrate. Make a list of all the variables you must define and their scope. For example, determine if they need to be accessed later in the orchestration. The following list is an example of variables that must be migrated, listed by variable name, type, and scope.

   objFile, System.IO.File, Private

   objWrite, System.IO.StreamWriter, Private

   strNow, System.String, Private

   strFilleName, System.String, Private

   strTempTileName, System.String, Private

   objDoc, System.Xml.XmlDocument, Private

   intTransferFlag, System.Int16, Private

10. Expand the **Scope** node you just created.

11. Right-click **Variables** and select **New Variable**.

12. In the **Properties** window, set the **Identifier** to the name by which you reference your object.

13. Select **<.NET Class>** or one of the primitive types from the **Type** list.

14. If you selected **<.NET Class>,** in References pane, expand your project name. (It is prefixed with Interop.)

15. In the **Type** pane, select the class you want to define and then click **OK**.

16. For all the required variables with a global expression, you need to define the variables at the orchestration variable level. Right-click on the **Expression** and select **Edit Expression**.

17. Implement the code in the BizTalk Expression Editor using C programming language with some modified BizTalk Server 2006 language syntax.

In BizTalk Server 2006, the Decision shape is renamed to Decide. Follow these steps to migrate the Decision shape.

To migrate a Decision shape

1. Drag a **Decide** shape onto the Orchestration Surface and place it below the last shape.
2. Click the **Decide** shape and open the **Properties** window.
3. Enter a name for the **Decide** action.
4. Right-click the **Rule_1** shape (within the Decide shape) and click **Edit Boolean Expression**.
5. Create a Boolean expression based on message values, variables or code calls. In the example, BizTalk Server 2006 references a variable set by a previous COM call.
6. If Rule_1 evaluates to True, any shapes placed below the rule execute. Otherwise, any shapes under the Else line execute.

To migrate an action connected to BizTalk Messaging Send

1. Drag a **Send** shape onto the Orchestration Surface and place it below the last shape.
2. Click the **Send** shape and open the **Properties** window.
3. Enter a name for the **Send** action.
4. Select a schema from the Message list.
5. Connect the **Send** shape to the **Port** you created for the BizTalk Messaging shape.

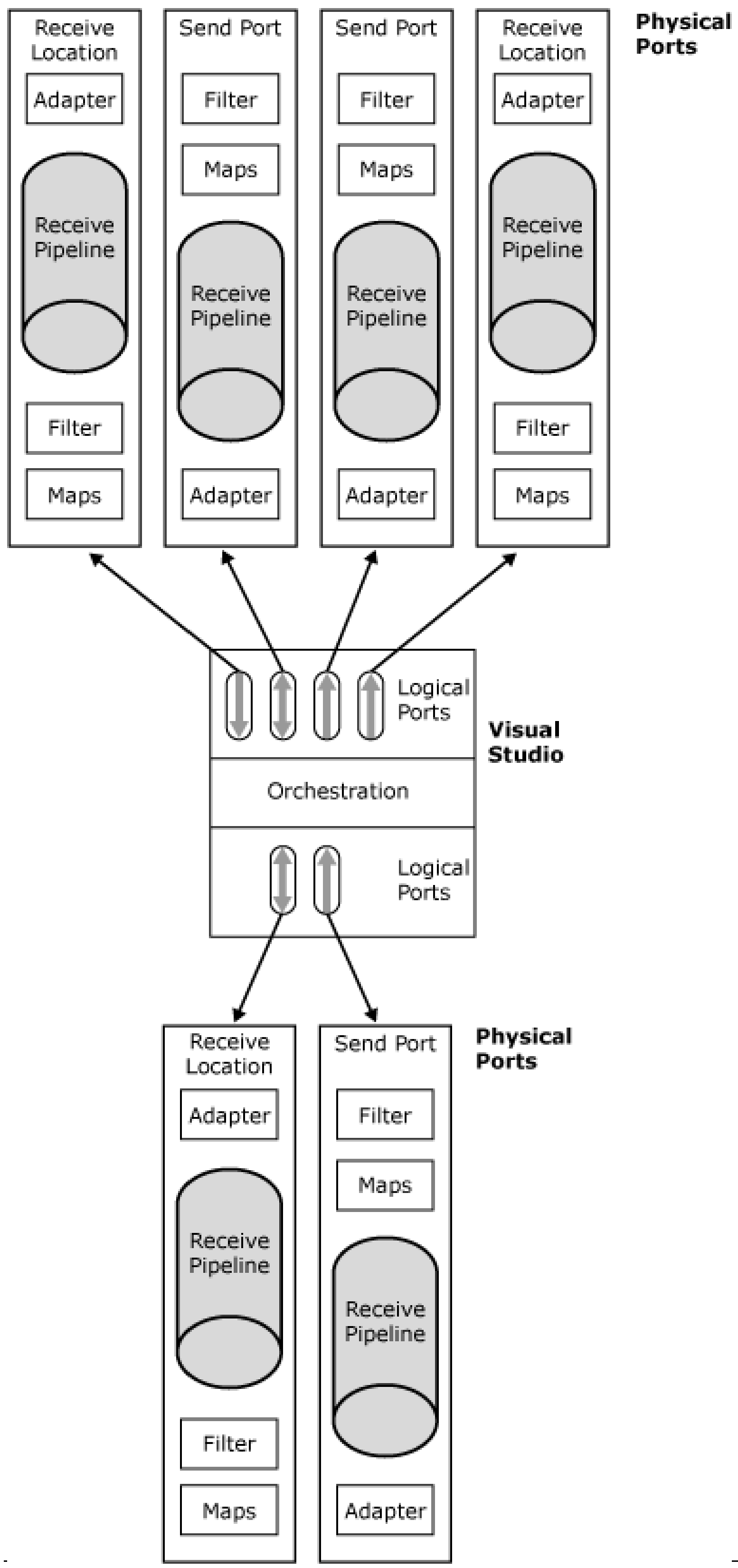
How to Bind an Orchestration for Migration

After you deploy the orchestration, you must bind it. This topic explains how to do that in a migration scenario.

Deploying an orchestration project consists of deploying the assembly, binding the orchestration, and enlisting and starting the orchestration. For deploying the assembly and enlisting and starting the orchestration, perform the steps as normal for BizTalk Server 2006. For more information about deploying orchestrations, see Managing Orchestrations.

BizTalk Server 2006 orchestrations do not rely on a channel and port to receive messages for processing. Instead, orchestrations use the message context data to subscribe to messages for processing. Messages are sent to an orchestration directly from the messaging engine and the orchestration returns the message directly to the MessageBox database when processing is complete. The message is then picked up by one or more BizTalk Server items, such as send ports or another orchestration, for further processing. For an overview of the BizTalk Server 2006 pub/sub architecture, see Updating a Message Migration Project.

Messages are sent to an orchestration directly from the message engine.



Receive Location
Adapter
Receive Pipeline
Filter
Maps
Send Port
Filter
Maps
Receive Pipeline
Adapter
Send Port
Filter
Maps
Receive Pipeline
Adapter
Receive Location
Adapter
Receive Pipeline
Filter
Maps
Physical Ports
Logical Ports
Visual Studio
Orchestration
Logical Ports
Receive Location
Adapter
Receive Pipeline
Filter
Maps
Send Port
Filter
Maps
Receive Pipeline
Adapter
Physical Ports

The orchestration creates a subscription through orchestration binding. Binding enables you to tie logical ports (found on the Port Surface in orchestrations) to physical ports and create a subscription to messages received through physical receive ports and messages intended for physical send ports. You can do this either at design time or at a later time using these logical port properties:

- **Specify Now** enables you to tie logical ports to physical ports at design time. You would typically use Specify Now if you already know the physical port addresses during design time.

- **Specify Later** enables you to tie logical ports to physical ports during deployment. If you use Specify Later, use BizTalk Explorer to create the bindings. After you have bound an orchestration, BizTalk Server 2006 creates the necessary subscriptions between your orchestration and the receive and send ports it uses.

BizTalk Server does not automatically start send and receive ports and orchestrations after compiling and deploying the migrated BizTalk project and running the Migration Wizard. You must ensure you start the appropriate ports, orchestrations, hosts, and so on, or your messages become suspended or may never get retrieved.

As part of the migration process, BizTalk Server 2006 creates a binding file containing the migrated send and receive ports. The Migration Wizard only generates one binding file for the send and receive ports in a solution. BizTalk Server 2006 creates these ports when you deploy the assembly and attach the binding file. After you add an orchestration to a project, BizTalk Server 2006 creates additional binding information around the assembly, but does not append it to the original binding file. If you deploy the migration project assembly with the original binding file created by the Migration Wizard, it generates an exception stating that the orchestration is not properly bound. You can work around this issue by deploying the assembly without the Migration Wizard binding file and then configure your send and receive ports manually. Conversely, you can choose to create another project and leave the projects separate with distinct binding information.

Perform the following steps to bind an orchestration's logical ports to the ports that were created as part of the migration process.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the local administrators group. In addition, BizTalk Server 2006 and Visual Studio 2005 must be installed.

To bind an orchestration for migration

1. Open **BizTalk Explorer** and expand the **Orchestrations** node.
2. Right-click the orchestration you deployed and click **Bind**.

3. In the **Port Binding Properties** window, bind the **Inbound** ports by selecting the port from the list that corresponds to the logical port (the orchestration port).

4. In the **Port Binding Properties** window, bind the **Outbound** ports by selecting the port from the list that corresponds to the logical port (the orchestration port) and then click **OK**.

Migrating Artifacts from BizTalk Server 2006

When you perform an in-place upgrade from BizTalk Server 2006 to BizTalk Server 2004, all of your artifacts are automatically migrated and placed in the default application, BizTalk Application 1 by default. This is because BizTalk Server 2004 did not include the concept of a BizTalk application, but all BizTalk Server 2006 artifacts must be associated with an application. You can then view and manage the artifacts from within the default application, or you can move them into separate applications in order to take advantage of the new application deployment features of BizTalk Server 2006. For more information about creating applications and moving artifacts, see Creating and Modifying BizTalk Applications. For background information about deploying and managing applications, see Understanding BizTalk Application Deployment and Management.

The following is an additional step that you may need to take when upgrading to BizTalk Server 2006:

When importing a binding file from BizTalk Server 2004 into BizTalk Server 2006, if the binding file includes settings for an adapter, verify that the Name attribute of the TransportType element in the binding file is the same as that configured for the adapter in the BizTalk Server Administration Console (under Platform Settings > Adapters).

In particular, you should verify that this is the case. Some transports for which this may be an issue are as follows:

•MQS

•MSMQ

•MSMQT

•POP3

•Windows SharePoint Services

For more information about editing binding files, see Customizing Binding Files. For background information about binding files, see Binding Files and Application Deployment.

How to Import Artifacts from a BizTalk Server 2006 .msi File

This topic describes how to import the artifacts contained in an .msi file that you created by using BTSInstaller in BizTalk Server 20045 into a BizTalk Server 2006 application. When you upgrade to BizTalk Server 2006, any artifacts in the databases are automatically upgraded as well. If you have an application that only exists in an .msi file, however, and not in the BizTalk Server 2004 databases, you might want to follow the procedure in this topic to import its artifacts into BizTalk Server 2006.

You cannot use the BizTalk Server 2006 Import Wizard to import .msi files generated by BTSInstaller into the BizTalk Server 2006 databases. Instead, you must use the command line to install the artifacts as well as import them into the BizTalk Server 2006 databases. This is the same method that you used in BizTalk Server 2006. The difference is that at the same time they are imported, the artifacts are automatically associated with the default BizTalk application, BizTalk Application 1 by default. You can then view, manage, and deploy them as a unit within this application.

Prerequisites

To perform the procedure in this topic, you must be logged on with an account that is a member of the BizTalk Server Administrators group. In addition, if the assemblies have been configured for installation in the global assembly cache, you must be logged on as a member of the Administrators group on the local computer. For more detailed information about permissions, see Permissions Required for Deploying and Managing a BizTalk Application.

To import an .msi file created with BTSInstaller

1. Open the BizTalk Server Administration console as follows: Click **Start**, point to **Programs**, point to **BizTalk Server 2006**, and then click **BizTalk Server Administration**.

2. In the console tree, expand BizTalk Server 2006 Administration, expand the BizTalk group, expand Applications, and then expand the default application (BizTalk Application 1 by default).

3. Click the subfolders in the default application folder, and make a list of any artifacts that they contain. The artifacts from the BizTalk Server 2004 .msi file will be imported into the default application. You will then move the artifacts out of this application into a new application. You can refer to this list to make sure you are not moving any artifacts that already existed in the default application before you imported the artifacts from the .msi file.

4. Copy the .msi file onto the computer running the BizTalk Management database for the group.

5. At a command prompt, navigate to the location of the .msi file, and then type the following command:

MSIEXEC /i *filename***.msi**

The application is installed on the local computer. Any deployment options that were specified for the assemblies are carried out, such as installing them to the global assembly cache. In addition, artifacts are copied to their specified installation locations. Finally, the artifacts in the .msi file are imported into the BizTalk Management database and associated with the default application. The artifacts display in the administration console in the subfolders of this application.

6. Create an application to contain the artifacts, as described in How to Create an Application.

7. Move the artifacts that you imported into the new application, as described in How to Move an Artifact to a Different Application. Do not move any of the artifacts that existed in the default application before you imported the artifacts into it.

8. If necessary, add to the application any artifacts that are required for your application to function which were not included in the original .msi file, such as non-BizTalk Server .NET assemblies. For instructions, see How to Create or Add an Artifact.

9. Export the application into a new .msi file, as described in How to Export a BizTalk Application.

10. If you intend to update a BizTalk Server 2004 application by installing the new .msi file, uninstall the existing application from any computers that are running it, and delete the old .msi file. You can uninstall the application by using Add or Remove Programs in Control Panel.

11. If you want, you can delete the application that you just imported from the BizTalk Server databases. You might want to do this if you do not anticipate needing to make any further modifications to it in this BizTalk group. For instructions, see How to Delete a BizTalk Application from the BizTalk Group.

    You can now use this .msi file to import the application and its artifacts into a different BizTalk group or install the application onto the computers that will run it. For more information, see How to Import a BizTalk Application and How to Install a BizTalk Application.

Interoperating with Previous Versions of BizTalk Server

The topics in this section describe how Microsoft BizTalk Server 2006 interoperates with BizTalk Server 2002 and BizTalk Server 2006.

You may want to upgrade your BizTalk Server 2002 or BizTalk Server 2006 installation to BizTalk Server 2006, but maintain trading with partners who continue to use BizTalk Server 2002 or BizTalk Server 2006. This interoperability is supported for various messaging scenarios and transport protocols, as well as for standard BizTalk Server features, such as reliable messaging, security, and BizTalk Server Web services.

In This Section

- Messaging Interoperability
- Protocol Interoperability
- Web Services Interoperability
- BizTalk Server Features with No Interoperability Impact

Messaging Interoperability

Messages can flow reliably between BizTalk Server 2006 and BizTalk Server 2002 or BizTalk Server 2006. In BizTalk Server 2006, you implement reliable messaging in two ways:

- By using a message envelope which contains the metadata of messages in BizTalk Framework (a schema for describing reliable messaging).
- By sending a message through the Message Queuing (MSMQ) adapter.

Secure Multipurpose Internet Mail Extensions (S/MIME) handles all security issues. In BizTalk Server 2002, you secure a message through a messaging port; in Biz Talk Server 2004, you secure it by configuring a custom send pipeline (encrypt, encode, and assemble); and in BizTalk Server 2006, you secure it by using an S/MIME encoder component in a custom send pipeline and configuring an encryption certificate on a send port.

Protocol Interoperability

BizTalk Server 2006 can exchange messages with BizTalk Server 2002 by using any of the following transports:

- FILE
- HTTP
- Web services
- MSMQ
- SQL
- SMTP (to send e-mail messages from BizTalk Server 2002)

- POP3 (in BizTalk Server 2006, to receive e-mail messages sent by BizTalk Server 2002)

BizTalk Server 2006 can exchange messages with BizTalk Server 2006 by using any of its transport adapters. Those include but are not limited to the following:

- FILE
- HTTP
- Web services
- MSMQ
- MQ Series
- SQL
- FTP
- SMTP (to send e-mail messages from BizTalk Server 2006)
- POP3 (in BizTalk Server 2006, to receive e-mail messages sent by BizTalk Server 2006)

Web Services Interoperability

Web services in BizTalk Server 2002 can be used by BizTalk Server 2006, but not conversely. Web services in BizTalk Server 2006 can be used by BizTalk Server 2006 and conversely.

BizTalk Server Features with No Interoperability Impact

The following BizTalk Server features have no specific interoperability issues. The following list identifies BizTalk Server features and their functions in BizTalk Server 2002, BizTalk Server 2006, and BizTalk Server 2006.

- BizTalk Framework with File Transfer Protocol (FTP)

  There is no FTP support in BizTalk Server 2002. BizTalk Server 2006 and BizTalk Server 2006 can exchange messages using FTP.

- BizTalk Framework with Flat File

  Flat File is no different in BizTalk Server 2002, BizTalk Server 2006, and BizTalk Server 2006 environments compared to other applications.

- Processing XML data

BizTalk Server 2006 can process any XML data sent to it by BizTalk Server 2002 as long as the XDR schema is migrated to XSD schema in BizTalk Server 2006. BizTalk Server 2006 can process any XML data sent to it by BizTalk Server 2006 and conversely. There are no unique considerations in the BizTalk Server 2002, BizTalk Server 2006, and BizTalk Server 2006 environments.

- XDR and XSD Schema Conversion

BizTalk Server 2006 XML tools fully support the conversion of any XDR schemas to XSD schema. BizTalk Server 2006 does not support conversions from XSD schema to XDR schema. However, you can find such a conversion tool in the marketplace. If BizTalk Server 2002 applications process new XML messages from BizTalk Server 2006, you must define both XDR and XSD schemas for the XML message.

- SQL adapter

A SQL adapter exists in BizTalk Server 2002, BizTalk Server 2006, and BizTalk Server 2006. The interoperability between BizTalk Server 2002 or BizTalk Server 2006 and BizTalk Server 2006 using a SQL Server database is straightforward. There is no difference between BizTalk Server 2006 interoperability with BizTalk Server 2002, BizTalk Server 2006, or other applications through the SQL Server database.