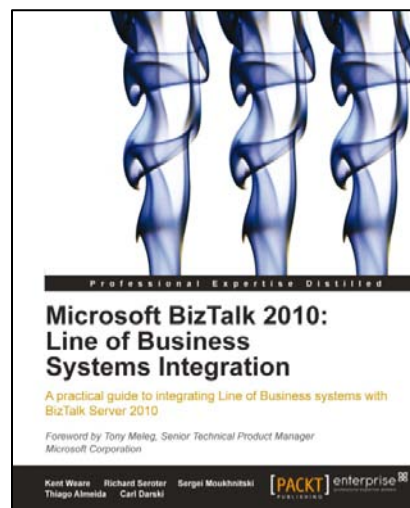


Microsoft BizTalk 2010: Line of Business Systems Integration

Kent Weare
Richard Seroter
Sergei Moukhonitski
Thiago Almeida
Carl Darski



Chapter No.9 "Microsoft Dynamics AX"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.9 "Microsoft Dynamics AX"

A synopsis of the book's content

Information on where to buy this book

About the Authors

Kent Weare, born in Regina, Saskatchewan, Canada, developed a love for Ice Hockey, Football and Technology. He attended the University of Regina where he obtained a Degree in Computer Science. After completing his undergrad degree, he spent time in India completing a Post Graduate diploma in Object Oriented Technology. He currently lives in Calgary, Alberta, Canada but remains a die-hard Saskatchewan Roughrider football fan.

Kent began his career at a small Internet startup before taking on a Junior roll with the Saskatchewan Government. Since then he has worked on projects for the Canadian Federal Government, a multi-national bank in the United States, Health Care projects in Eastern and Western Canada and has spent the last five years employed by a large Electricity Distribution company in Calgary. Kent's current responsibilities involve managing a Microsoft Solutions team that supports BizTalk, Exchange, Office Communication Server (OCS), and System Center.

During Kent's time at the Federal Government, he had an opportunity to participate in his first BizTalk project. Seven years later he is still "hooked" on BizTalk, having worked with every BizTalk version released since. In 2008, Kent was awarded his first Microsoft MVP award for BizTalk Server. He continues to be active in the BizTalk community and recently received his fourth consecutive MVP award. Kent maintains an active blog at

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

<http://kentweare.blogspot.com> and may be seen presenting BizTalk related material at local and international user groups.

I would first off like to thank my parents (Keith and Joyce) for their unconditional love, support, and direction growing up. They are strong proponents of working hard, treating others with respect, and taught me the difference between right and wrong.

To my twin brother (Kurt), sister (Kim), and their families, thank you for your interest in the book and the encouragement to keep plugging away at it until it was completed.

Throughout my career many people have seen something in me that I haven't necessarily seen in myself. I am convinced that without this support, I would have never been in a position get into the MVP program or write this book. Special thanks to Les Phillips, Dave Patel, Ron Naidu, Vasu Iyengar, Lucie Duval, Neal Nishikawa, Darren Jeffrey, Brian Dempsey, and Alan Skiffington for giving me an opportunity and then giving me the tools to be successful. I would also like to thank my mentors Nipa Chakravarti and Karl Smith for their insight and challenging me to grow as a Leader.

Lastly, this book would not have been a reality without the support of my loving wife Melissa and daughter Brooke. Writing a book is a huge undertaking that consumed late nights, early mornings, and pretty much any spare time in-between. Thank you Melissa for your patience, support, and putting up with me sitting in front of the computer for hours on end while I worked on the book. Brooke, thank you for not hitting the blue (power) button too many times while I was writing. At times you really pushed the limits of Microsoft Word's auto-recovery feature.

Richard Seroter is a solutions architect for an industry-leading biotechnology company, a Microsoft MVP for BizTalk Server, and a Microsoft Connected Technology Advisor. He has spent the majority of his career consulting with customers as they planned and implemented their enterprise software solutions. Richard started his career working for two global IT consulting firms that gave him exposure to a diverse range of industries, technologies, and business challenges. Then, he joined Microsoft as a SOA/BPM technology specialist where his sole objective was to educate and collaborate with customers as they considered, designed, and architected BizTalk solutions. One of those customers liked Richard enough to bring him onboard full time as an architect after

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

they committed to using BizTalk Server as their enterprise service bus. Once the BizTalk environment was successfully established, Richard transitioned into a solutions architect role where he now helps to identify enterprise best practices and apply good architectural principles to a wide set of IT initiatives.

Richard is the author of two books including *Applied Architecture Patterns on the Microsoft Platform* (Packt Publishing, 2010), which discusses where to use which part of Microsoft's platform technologies. He is also the author of *SOA Patterns for BizTalk Server 2009* (Packt Publishing, 2009), which takes a look at how to apply good SOA principles to a wide variety of BizTalk scenarios.

I'd like to thank Kent for bringing me onto this interesting project and allowing me to contribute some chapters. I have a sick passion for enterprise integration and Kent provided me a perfect outlet for more research into the topic.

Thanks to all of my co-workers who have inevitably influenced my thinking and made me a better architect and technologist. My manager, Nancy Lehrer, continues to be an exceptional mentor on my jagged path to superstardom. Finally, thanks to my family and my boys (Noah the human, Watson the dog) as they put up with late nights and absentmindedness.

Sergei Moukhnitski is a software architect with 15 years of experience developing software and systems. His area of professional interests is business process and integration technologies applied to ERP and CRM systems such as Microsoft Dynamics and SAP.

Thiago Almeida is a Senior Integrations Consultant for one of New Zealand's largest IT service providers. He has over eight years of experience working as a solutions architect and senior developer on projects for some of the country's largest companies. Thiago has been awarded Most Valuable Professional in BizTalk from Microsoft in 2009 and 2010 to acknowledge his exceptional contributions to the BizTalk Server technical community. He runs the Auckland Connected Systems User Group, being a frequent speaker, and maintains a blog on BizTalk Server at <http://connectedthoughts.wordpress.com>

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

He is also a Microsoft Connected Technology Advisor and a Microsoft Virtual Technical Solution Professional in Integration, providing advanced technical input in BizTalk Server and associated integration technologies on client engagements with Microsoft.

I would like to thank my teammates for working on so many great projects and sharing real world BizTalk Server knowledge with me. I would also like to thank the Microsoft employees, fellow MVPs, and the co-authors for all the input and help. On a more personal note, I would like to thank my beautiful wife Karla for the encouragement and for putting up with the late nights and weekends spent working on this book. New Zealand's great scenery, beautiful beaches, and nice weather during such weekends are purposely not included in this acknowledgement.

Carl Darski is a software consultant who has diverse professional experience in sectors including telecommunications, oil and gas, power and mining. As an electrical engineer, Carl quickly recognized the growing importance of software integration technologies. He has assisted several clients in implementing BizTalk as an integration standard.

I would like to thank all my coworkers at Ideaca who have helped me both technically and professional over the past several years. A special thanks to Shane James whose BizTalk knowledge never ceases to amaze nor disappoint me.

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

Microsoft BizTalk 2010: Line of Business Systems Integration

Microsoft BizTalk is an integration server solution that allows businesses to connect disparate systems. In today's business climate of mergers and acquisitions, more and more enterprises are forced to exchange data across Line of Business systems using integration brokers like BizTalk Server 2010. What is often overlooked when integrating these systems is the pre-requisite knowledge that ERP and CRM systems demand in order to effectively integrate them. No longer is this knowledge locked up in the heads of expensive consultants. Gain an edge within your organization by developing valuable skills in the area of Line of Business integration from this book.

This book will show you how to integrate BizTalk with Line of Business systems using practical scenarios. Each chapter will take a Line of Business system, introduce some pre-requisite knowledge and demonstrate how you can integrate BizTalk with that Line of Business system, and then provide guidance based upon real world experience, taking your BizTalk knowledge further.

This book will enable you to master how to integrate BizTalk with Line of Business systems. The book starts by highlighting the technical foundation of WCF-LOB adapters and the common steps and important properties pertaining to popular WCF-LOB adapters. You will then move on to an overview of how to integrate with Microsoft SQL Server using the WCF based SQL Server adapter. The book then dives into topics such as integrating with Dynamics CRM, building BizTalk/SAP integrated solutions using IDOCs, the differences between IDOCs and RFCs/BAPIs, and WCF Integration through the Windows Azure AppFabric Service Bus amongst others.

What This Book Covers

Chapter 1, Consuming ASK-based Adapters: Explore some of the inner workings of the WCF LOB SDK and WCF Custom Adapter.

Chapter 2, WCF-SQL Adapter: Learn how to retrieve and manipulate data using popular operations exposed by the WCF-SQL Adapter including Polling, Notifications, and Composite Operations.

Chapter 3, Integrating BizTalk Server and Microsoft Dynamics CRM: Discover different ways to integrate with Dynamics CRM 2011 and BizTalk Server including calling native Web Services and Proxy solutions. Also learn how to call a BizTalk exposed WCF Service via a CRM registered plug-in.

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

Chapter 4, WCF-SAP Adapter Sending and Receiving IDOCs: Understand how to install the WCF-SAP adapter's pre-requisite DLLs. Learn about extended, custom and out of the box IDOCs and how to send and receive them.

Chapter 5, WCF SAP Adapter RFCs and BAPIs: Distinguish the difference between SAP IDOCs, BAPIs and RFCs and when to use them.

Chapter 6, BizTalk Integration with Windows Azure AppFabric Service Bus: Discover Microsoft's AppFabric Service bus and learn how to build BizTalk solutions that complement Microsoft's Service bus in the Windows Azure Cloud.

Chapter 7, Integrating with SharePoint 2010: Build integrated SharePoint solutions using the Windows SharePoint Services Adapter and InfoPath.

Chapter 8, Integrating with SharePoint 2010 Web Services: Learn about manipulating SharePoint custom lists by consuming SharePoint's out of the box List Web Service.

Chapter 9, Microsoft Dynamics AX: Understand how to integrate with Microsoft Dynamics AX 2009 using the BizTalk adapter and .Net business connector.

Chapter 10, Integrating BizTalk Server and Salesforce.com: Discover how to establish bi-directional connectivity between Salesforce.com CRM and your on-premise services.

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

9

Microsoft Dynamics AX

In this chapter, we'll discuss integrating with Microsoft Dynamics AX 2009 using the BizTalk Dynamics AX adapter. It will be a complete walkthrough for the configuration setup in the Dynamics AX 2009 Application Integration Framework (AIF) module and related set up for batch jobs. We'll complete two BizTalk applications that will demonstrate both synchronous and asynchronous forms of integration and show the benefits of using the AIF module.

The completed solutions are included in the code for this chapter. In the first walkthrough example, we'll create a BizTalk application to populate the exchange rates table. In the second walkthrough example, we'll create a BizTalk application that retrieves messages from Dynamics AX 2009 via the AIF Queue. Also included is a console application that demonstrates how to leverage the .NET business connector to integrate with Dynamics AX 2009.

We will specifically cover:

- What is Dynamics AX?
- Methods of integration with AX
- Installing the adapter and .NET Business Connector
- Configuring Dynamics AX 2009 Application Integration Framework for BizTalk Adapter
- Synchronous walkthrough example – Currency Exchange Rates
- Asynchronous walkthrough example – Dynamics AX message outflow
- Other development and configuration notes

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book

What is Dynamics AX?

Microsoft Dynamics AX (formally **Microsoft Axapta**) is Microsoft's Enterprise Resource Planning (ERP) solution for mid-size and large customers. Much like SAP, Dynamics AX provides functions that are critical to businesses that can benefit from BizTalk's integration. Microsoft Dynamics AX is fully customizable and extensible through its rich development platform and tools. It has direct connections to products such as Microsoft BizTalk Server, Microsoft SQL Server, Exchange, and Office.

Often Dynamics AX is compared to SAP All in One. Those who are familiar with SAP are also familiar with high cost of implementation, maintenance, and customization associated with it. A Microsoft Dynamics AX solution offers more customizability, lower maintenance costs, and lower per-user costs than SAP. ERP implementations often fail in part due to lack of user acceptance in adopting a new system. The Dynamics AX user interface has a similar look and feel to other widely used products such as Microsoft Office and Microsoft Outlook, which significantly increases the user's comfort level when dealing with a new ERP system. For more information on Dynamics AX 2009 and SAP, please see <http://www.microsoft.com/dynamics/en/us/compare-sap.aspx>

Methods of integration with AX

Included with Dynamics AX 2009, Microsoft provides two tools for integration with Dynamics AX:

- Dynamics AX BizTalk Adapter
- .NET Business Connector

The BizTalk adapter interfaces via the **Application Interface Framework Module** (AIF) in Dynamics AX 2009, and the .NET Business Connector directly calls the Application Object Tree (AOT) classes in your AX source code.

The AIF module requires a license key, which can add cost to your integration projects if your organization has not purchased this module. It provides an extensible framework that enables integration via XML document exchange. A great advantage of the AIF module is its integration functionality with the BizTalk Dynamics AX adapter. Other adapters include a **FILE adapter** and **MSMQ**, as well as Web Services to consume XML files are included out of the box. The AIF module requires a fair amount of setup and configuration, which we will discuss later in this chapter. Other advantages include full and granular security, capability of synchronous and asynchronous mode integration mode, and full logging of transactions and error handling.

The Microsoft BizTalk AX 2009 adapter can execute AX actions (exposed functions to the AIF module) to write data to AX in both synch and asynch modes. Which mode is used is determined by the design of your BizTalk application (via logical ports). A one-way send port will put the XML data into the AIF queue, whereas a two-way send-receive port will execute the actions and return a response message. Asynch transitions will stay in the AIF queue until a batch job is executed. Setting up and executing the batch jobs can be very difficult to manage, something which we will discuss later in this chapter. Pulling data from AX can also be achieved using the BizTalk adapter. Transactions pushed into the same AIF queue (with an OUTBOUND direction in an async mode) can be retrieved using the AX adapter which polls AX for these transactions.

The .NET Business connector requires custom .NET code to be written in order to implement it. If your business requirements are for a single (or very small amount) of point-to-point integration data flows, then we would recommend using the .NET Business Connector. However, this often requires customizations in order to create and expose the methods. Security also needs to be handled with the service account that the code is running under. We'll touch on this later in this chapter and we've provided a working example with the source code for this chapter.

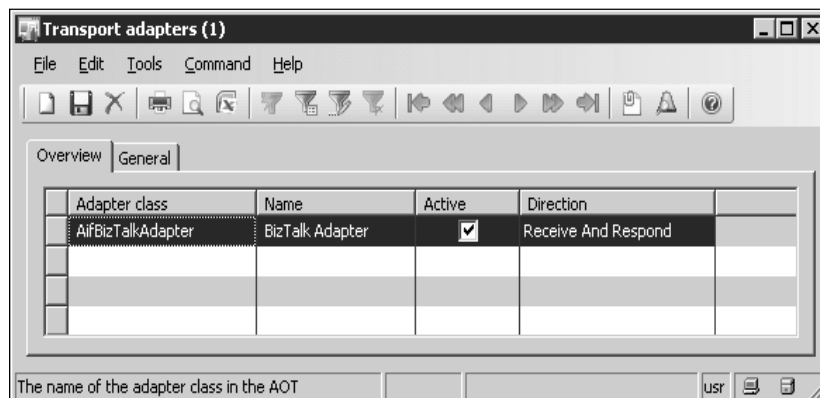
Installing the adapter and .NET Business Connector

The Microsoft BizTalk adapter for Dynamics AX 2009 and the .NET Business Connector are installed from your Dynamics AX Setup install setup under **Integration** on the **Add or Modify components** window. Each component is independent of one another; however the BizTalk adapter leverages components of the business connector. You are not required to install the Dynamics AX client on the BizTalk server. When installed in BizTalk adapter, you can simply select all the defaults from the install wizard. For the .NET business connector, you'll be prompted for the location of your Dynamics AX instance. This will be used only as a default configuration and can easily be changed.

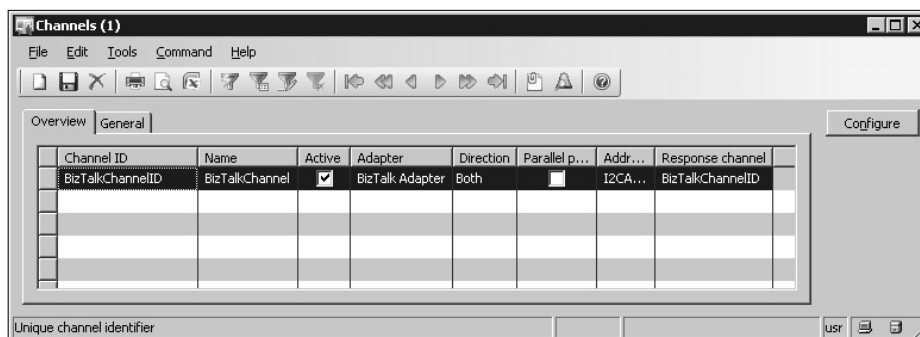
Configuring Dynamics AX 2009 Application Integration Framework for BizTalk Adapter

Configuration of the AIF module involves several steps. It also goes a long way to increasing your understanding of the granularity of Dynamics AX setup and security considerations that were taken into account for integration of what can be highly sensitive data. It is recommended that this setup be done with Admin level security, however, only full control of the AIF module is required. This setup is almost identical in version prior to Dynamics AX 2009; minor differences will be noted.

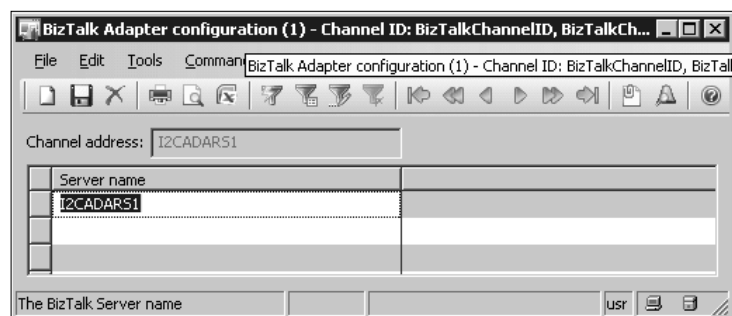
1. All AIF setup tables can be found in Dynamics AX under **Basic | Setup | Application Integration Framework**.
2. The first step is rather simple, however critical. In the **Transport Adapters** form, add in a new entry selecting **Adapter Class** drop down **AifBizTalkAdapter**, select **Active**, and **Direction** will be **Receive and Respond**. You also notice there are two other out-of-the-box adapters: FILE and MSMQ. This is a one-time setup that is effective across all companies.



3. Next, using the **Channels** form, set up an active channel for your specific BizTalk server. Select a meaningful and identifiable **Channel ID** and **Name** such as **BizTalkChannelID** and **BizTalkChannel**. Select the **Adapter** to **BizTalk Adapter**, check **Active**, set **Direction** to **Both**, **Response channel** equal to the Channel ID of **BizTalkChannelID**. Set the **Address** to your BizTalk Server (**I2CDARS1** as shown below).



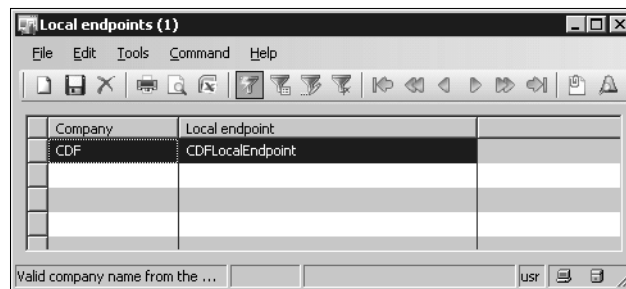
4. Then, click **Configure** to bring up the **BizTalk Adapter configuration** form. Again, set the **Server name** to the name of your BizTalk Server instance.



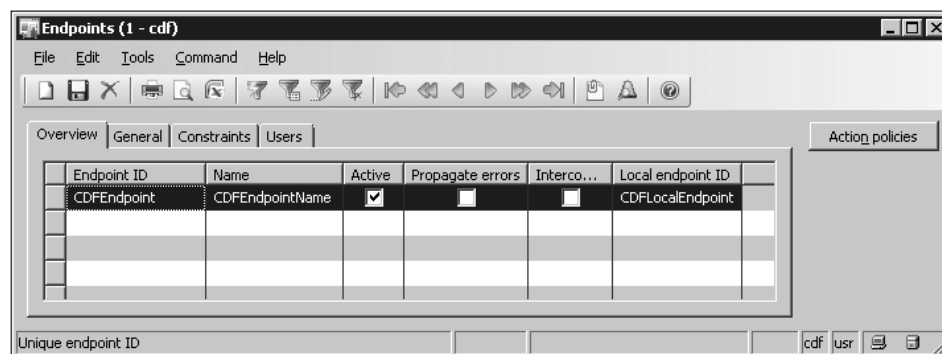
5. This configuration is required for secure connectivity during runtime. It is also required in order for Visual Studio to connect to your Dynamics AX 2009 instance when using the schema import wizard (we'll do this later in this chapter). This is also a one-time setup across all companies.
6. Now, in this step, we will set up a local endpoint from the **Local endpoints** form. Each company that your BizTalk application requires connectivity with will require a local endpoint. Select the **Company** you want to configure (CDF in our case) and type in **CDFLocalEndpoint** as shown.



We recommend using this type of naming convention (**Company** + 'LocalEndpoint'), as often your BizTalk applications will need to specify this value. Often, the incoming message will contain data to determine which company it is intended for, thus, you'll need to dynamically determine the local endpoint during runtime. See the walkthrough example later in this chapter where we'll need to send currency exchange rates to multiple Dynamics AX companies.



7. Next, we need to create an endpoint from the **Endpoints** form. The endpoint links together our local endpoint, BizTalk channel, BizTalk service account, and active Dynamics AX actions. Again, each company will require a local endpoint and thus we'll use the same naming conversion.
8. We'll start by typing in the **Endpoint ID** to **CDFEndpoint** and select the **Local endpoint ID** to **CDFLocalEndpoint**.



9. On the **General** tab, specify the **Outbound channel ID** to **BizTalkChannelID**, which we created above. This will also allow asynchronous outbound messages to be retrieved from Dynamics AX 2009 using the BizTalk adapter.

Endpoints (1 - ceu)

File Edit Tools Command Help

Overview General Constraints Users

Identification

Endpoint ID: CEUEndpoint

Name: CEUEndpoint

Active: ☒

Propagate errors: ☐

Overview

Intercompany organization: ☐

Company:

Channel

Outbound channel ID: BizTalkChannelID

Local endpoint ID: CEULocalEndpoint

Default encoding format: UTF-8

Unique channel identifier

10. On the **Constraints** tab, we'll check **No constraints**. If required, you can specify constraints such as Customer and Vendor. Now, set the record to **Active** from the **General** tab and save.

Endpoints (1 - cdf)

File Edit Tools Command Help

Overview General Constraints Users

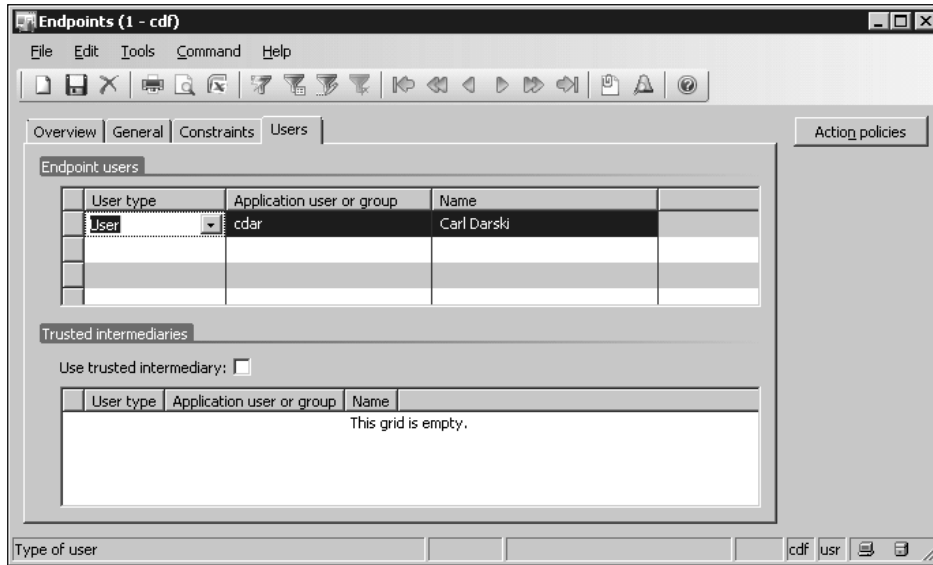
No constraints: ☒

Constraint type	Constraint ID	Name
This grid is empty.		


Action policies

Unique endpoint ID

11. On the **Users** tab, we'll add the AX BizTalk user (linked to the Active Directory account that our BizTalk host instance is running under) by selecting **User type** to **User**, and then selecting the AX BizTalk (**cdar** in the example shown) user. Thus, all the records in AX will have this account as created by user. Depending on your integration requirements, this user may also need other associations such as employee.



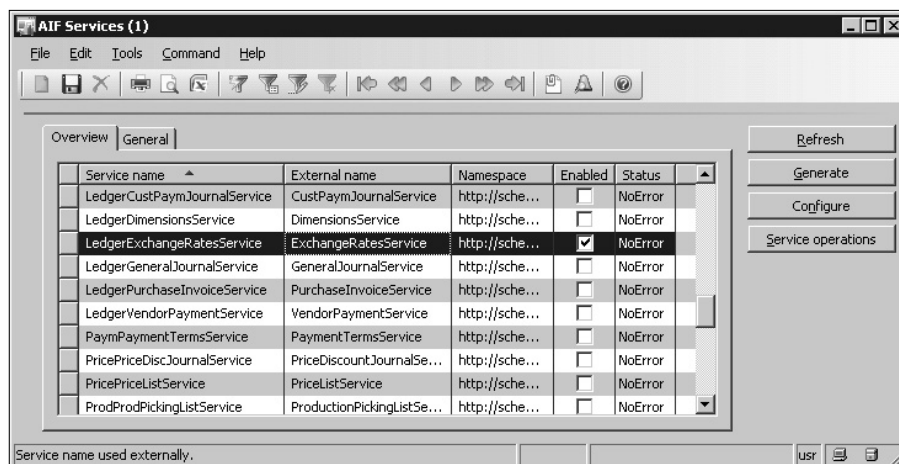
12. The next thing to do (after we have the AIF base configuration setup complete in AX) is to determine which **AIF Services** we are going to use. This will allow us to know what services to and thus which we need to enable. Each service has several actions associated with it. If there are no services out of the box with actions that meet our needs, we can either customize existing services or create our own. This is where the speed and ease of customizations in Dynamics AX are of great advantage.


[

 Note: Prior to AX 2009 actions were not grouped in services, thus no AIF Services table existed.

]

13. For this example, (we'll use this later in this chapter) we must use the out-of-the-box service **LedgerExchangeRatesService**. So, we will enable this service in the **AIF services** form as shown in the following screenshot:

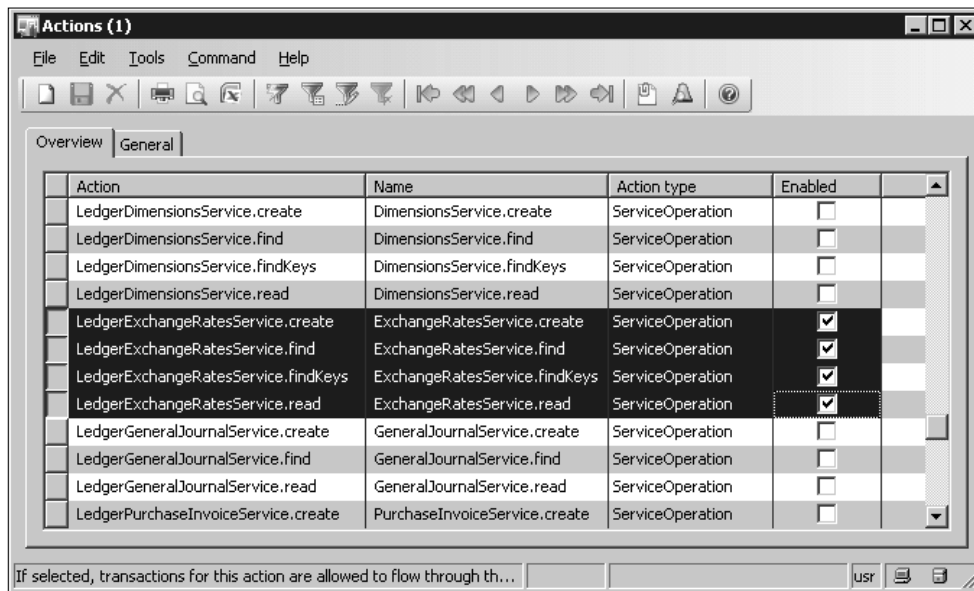


14. On the **AIF Services** form, we can examine the AX schema(s) for each by highlighting a **Service name**, clicking **Service operations**, selecting a specific **Action**, and then clicking **Parameter schema**. Actions include **create**, **delete**, **find**, **findkeys**, **read**, and **update**. Select the **SalesOrderService** and notice that there is no outbound schema for **delete** and **update** actions because these do not return a message.

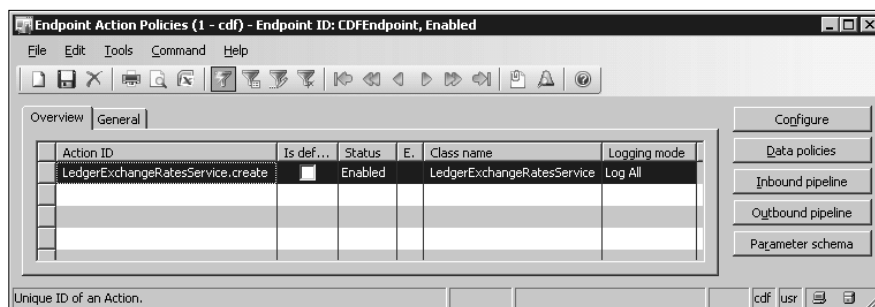
A basic record in the Exchange Rates table only requires three fields: (1) Currency code, (2) Exchange Rate, and (3) From Date. We don't typically specify an end date as AX logic handles this when the next day's rate is entered.

Each AIF service will have its own security key associated with it. You must give the DynamicsAX BizTalk user account that access by assigning user permission to the required security keys in Dynamics AX 2009. In prior versions of Dynamics AX 2009, (where the AIF did not have services) security permissions had to be handled manually. This can be done by creating a security group with the permission needed to execute each AIF action and associating the Dynamics AX BizTalk user with the group.

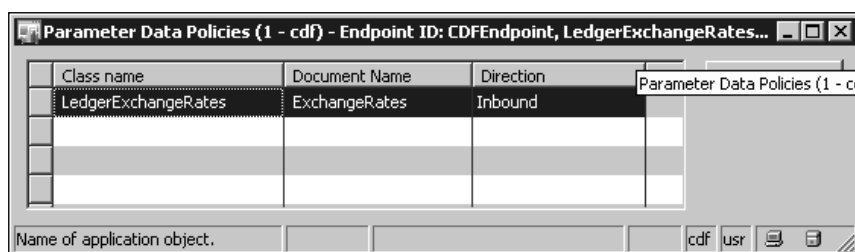
- Now that the AIF service is enabled, we can go over to the **Actions** form and see the four actions that are enabled. These are the actions that are a part of the AIF **LedgerExchangeRatesService** service. Notice we can't enable any default action individually without enabling the service (in previous versions of Dynamics AX each individual action was required to be enabled).



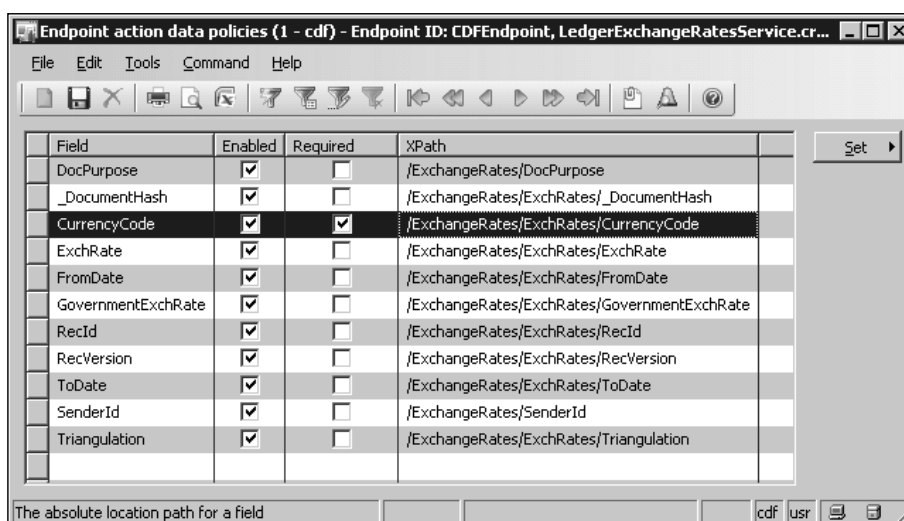
- For each Dynamics AX company, you'll need to enter a currency relative to the base currency. Thus if you have one company whose currency is USD based, and another that's CAD based, then you'll need to enter a rate in each company for each currency. The USD-based company will require a USD/CAD rate to be entered, and the CAD-based company will need a CAD/USD rate entered. Thus, you'll need to repeat the following AIF **Actions** setup for each company's **Endpoint** if you're using the **LedgerExchangeRatesService** AIF Service.
- Now, we go back to the **Endpoints** form and click on **Action policies** to bring up the **Endpoint Action Policies** form. Although we have activated the **LedgerExchangeRatesService**, the **Endpoint Action Policies** are action-specific, thus we need to create a record for each. Here, we'll select the **Action ID** to **LedgerExchangeRatesService.create** from the drop down; set the **Status** to **Enabled**, and **Logging Mode** to **Log All**. Now, we need to save the record. This will enable the **Configure** button on this form as shown in the following screenshot:



4. Next, click **Configure** to bring up the **Parameter Data Policies** window.



5. Click on **Data policies**; that will bring up the **Endpoint Action data policies** form. You'll notice that only the required fields are enabled. Here, we can get very granular in our security and data policies; however, with more complex AX actions this can get very tedious. Thus, simply click **Set** and then click **Enable all**.



If you add customization to any Dynamics AX services, adding a new field for example, you need to (1) update the AIF Services by hitting the Refresh button (as in the figure above) and (2) enable the field in the Endpoint action data policies table.

Typically, an integration policy would be not to override the value in the Exchange Rate table (based on currency & date) if it already exists. In fact, if we attempt to send a duplicate rate, AIF will throw an error. We could simply catch and ignore this error; however, this would be introducing unnecessary clutter in the AIF exception log. Since we can expose this table to AX business users, we want to limit as much as possible the amount of errors that are thrown from the AIF module.

6. So, before we attempt to push an exchange rate we want to query AX so as to see if a particular rate already exists. To do this, we can use the out-of-the-box action **LedgerExchangeRates.find**. Note that the schema for the action will return a list of exchange rates. So now we'll need to repeat the above steps to enable this **Action** in the AIF **Actions** form, add it to the **Action policies** of our AIF **Endpoint**, and enable all the fields in the **Endpoint Action data policies** form. This completes the configuration of the AIF module in Dynamics AX in order to do the examples in this chapter that are next.

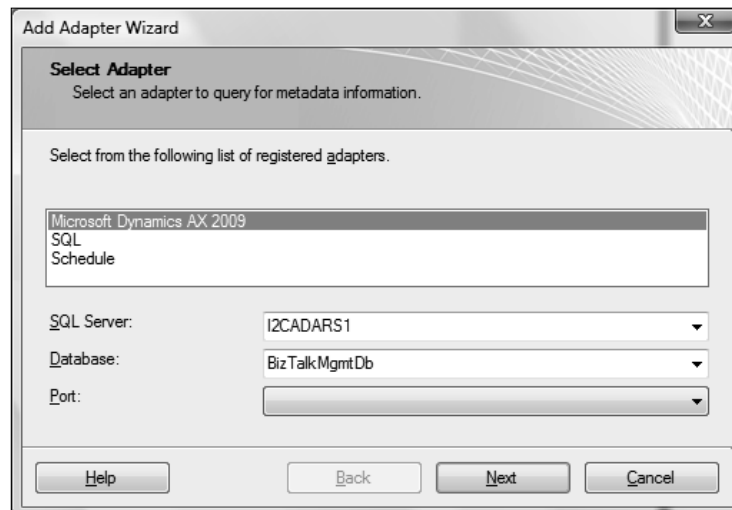
Synchronous walkthrough example— currency exchange rates

The best way to understand the behavior and setup required to use the Dynamics AX BizTalk adapter is to walk through a full example. A common requirement for many companies that use Dynamics AX is to populate the currency exchange rates. Typically, this needs to be daily when the closing rates are published. The **Exchange Rates(General ledger | Setup Exchange rates)** table in AX is not a shared table across companies, thus depending how your organization is set up; you may or may not need to populate the Exchange rate table using multiple companies with multiple rates. We'll further explain this shortly in the following example. We've found this to be an excellent practical example to begin with when learning to create integration applications with BizTalk and the Dynamics AX AIF module.

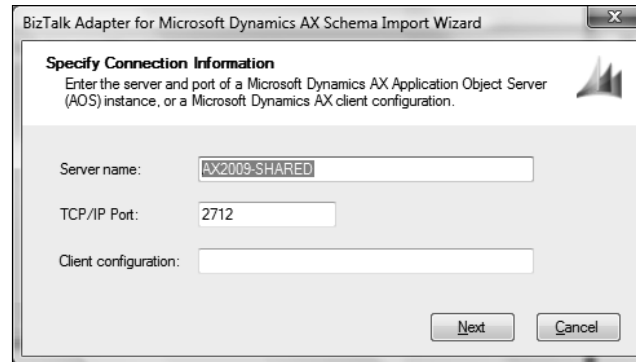
For this example, we are going to assume that we have two companies in our Dynamics AX implementation; one that is Canadian Dollar (CAD) based, and one that is United States Dollar (USD) based. Thus, we will need to repeat the AIF setup procedure (shown previously in this chapter) in order to use the **LedgerExchangeRates.create** and **LedgerExchangeRates.find** actions in both companies. For the remainder of this example, we'll refer to these as **daxCADCompany** and **daxUSDCompany**. The complete solution, titled **Chapter9-AXExchangeRates**, is included with the source code for this chapter.

Dynamics AX schemas

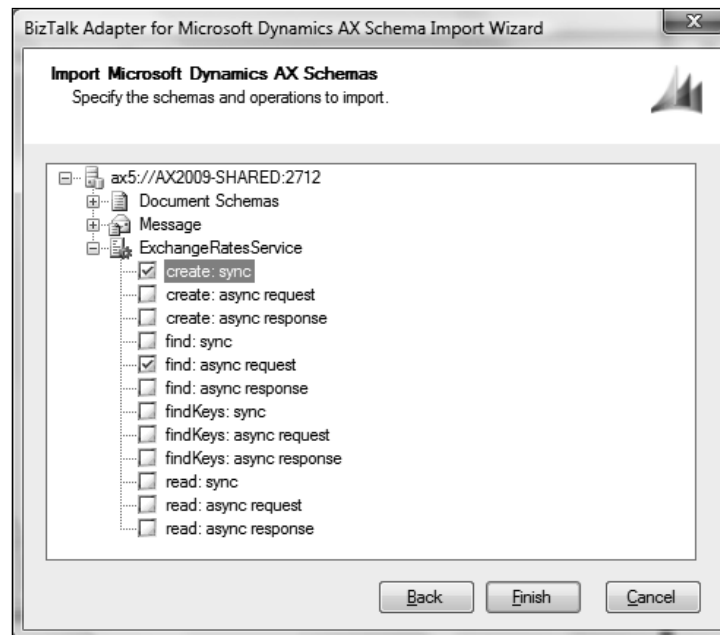
We'll start by creating a new BizTalk project, **Chapter9-AXExchangeRates**, in Visual Studio. After the AIF actions setup is complete (as shown previously in this chapter), the next step is to generate the required schemas that are needed for our BizTalk application. This is done by right clicking on the BizTalk project in Visual Studio 2010, click **Add**, highlight and click **Add Generated Items**. This will bring up the **Add Generated Items** window, under the **Templates** section – **Visual Studio installed template**, select **Add Adapter Metadata**, and click **Add**. This will bring up the **Add Adapter Wizard** window (shown in the following screenshot), so we'll simply select **Microsoft Dynamics AX 2009** and click **Next**. Now, we'll need to fill in the AX server instance name (**AX 2009-SHARED** in our example) under **Server name**, and the **TCP/IP Port** number (**2712**, which is the default port number, but this can differ). Now, click **Next** from the BizTalk Adapter for Microsoft Dynamics AX Schema Import Wizard window.



Specify the connection information in the next step.



In the next window, you should see all the active AIF services. Note that since the AIF services table is a global table, so you will see all the active services in your Dynamics AX instance. This does not mean that each endpoint, thus each company, is configured to accept the actions that each AIF service listed has available. This is the point where you first verify that your connectivity and AIF setup is correct. An error here using the wizard typically is due to an error in the AIF channel configuration.

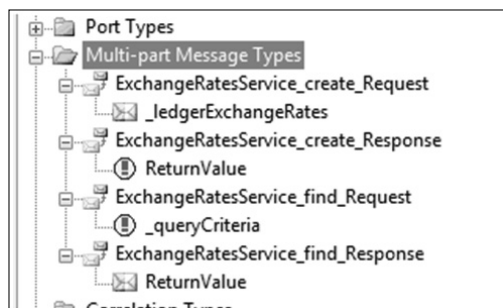


In the wizard window above, you can see the AIF services that are enabled. In our case, the **ExchangeRatesService** is the only service currently enabled in our Dynamics AX instance. Under this service, you will see three possible modes (**sync**, **async request**, and **async response**) to perform these actions. All three will actually produce the same schemas. Which mode and action (**create**, **find**, **findkeys**, or **read**) we use is actually determined by the metadata in our message we'll be sending to AX and the logical port configurations in our orchestration. Now, click **Finish**.

Now in the project solution, we see that the wizard will generate the two artifacts. The first **ExchangeRates_ExchangeRates.xsd** is the schema for the message type that we need to send when calling the **LedgerExchangeRates.create** action and it is also the same schema returned in the response message when calling the action **LedgerExchangeRates.find**. Since we are actually dealing with the same AX table in both actions, Exchange Rates, both actions will in part (one will be the inbound message, the other will be the outbound message) require the same schema.

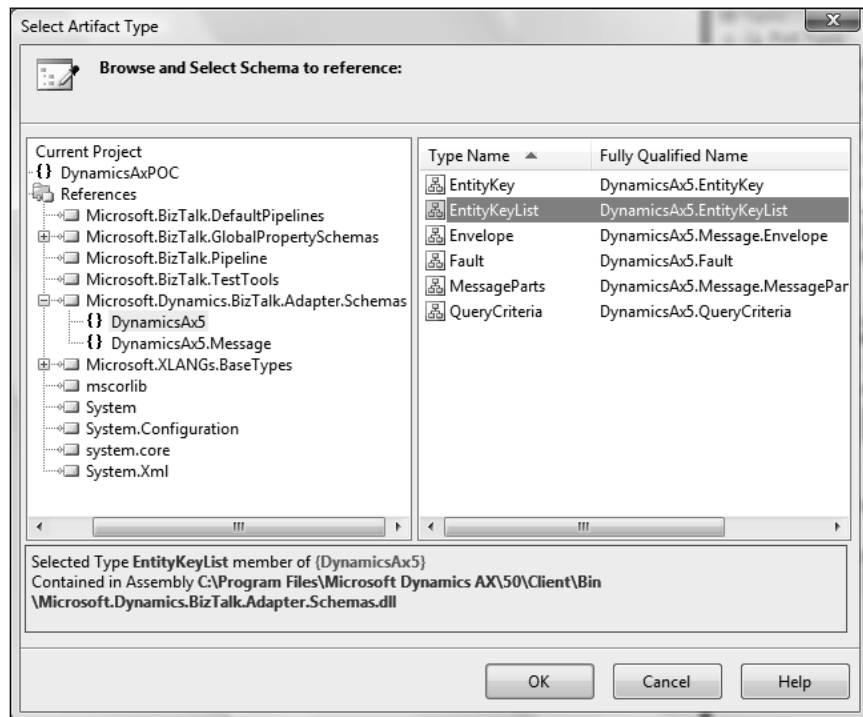
The second artifact, **BizTalk Orchestration.odx**, is also generated by default by the wizard. In the orchestration view, we can see that four **Multi-part Message Types** were also added to the orchestration. Rename the orchestration to something more meaningful such as **ProcessExchangeRates.odx**.

Now that we have defined our message type that will be returned in our response message, we need to define what the request type will be. Notice from the orchestration view that two messages, **ExchangeRatesService_create_Response** and **ExchangeRatesService_find_Request**, have types which Visual Studio has in error 'does not exist or is invalid'.



For the out-of-the-box *find* action, we need the message type **DynamicsAX5.QueryCriteria**. The other message type is return by AX when calling a create action is **DynamicsAX5.EntityKey** (if we called a createList action, the returned message would be of type **DynamicsAX5.EntityKeyList**).

The schemas for these message types are in the **Microsoft.Dynamics.BizTalk.Adapter.Schemas** assembly, which can be found in the bin directory of the install location of the BizTalk adapter. Add this reference to the project in Visual Studio. Then, re-select the appropriate message type for each Message Part that is invalid from the Select Artifact Type window as shown.

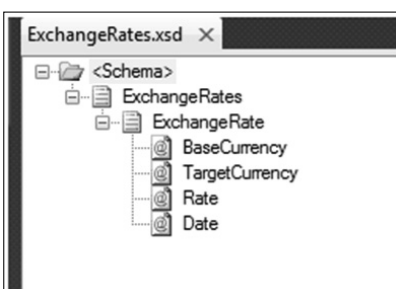


Next, depending on your organization, typically you may want to either populate the noon exchange rates or closing rates. For our example, we will use the closing USD/CAD exchange rates from the Bank of Canada. This is published at 16:30 EST on the website (<http://www.bankofcanada.ca/rss/fx/close/fx-close.xml>).

Since this source is already in XML, download and save a sample. We then generate a schema from Visual Studio using the BizTalk Schema Generator (right click the solution, **Add Generated Items, Add Generated Schemas**, using the **Well-Formed XML (Not Loaded)** document type. This will generate the schema for the message that we need to receive by our BizTalk application daily. In the example provided, the schema is **ClosingFxRates.xsd** (the wizard will generate four other .xsd files that are referenced in **ClosingFxRates.xsd**).

A simple way to schedule the download of this XML data file is to use the Schedule Task Adapter (<http://biztalkscheduledtask.codeplex.com/>), which can be downloaded and installed at no cost (the source code is also available). Download and install the adapter (requires Microsoft .NET Framework Version 1.1 Redistributable Package), then add using the BizTalk Server Administration Console with the name **Schedule**. We will use this adapter in our receive location to retrieve the XML via http. There are also RSS adapters available to be purchased from, for example, /nsoftware (<http://www.nsoftware.com/>). However, for this example, the scheduled task adapter will suffice.

Now, since the source of our exchange rates is a third-party schema, and your specific requirements for the source will most likely differ, we'll create a canonical schema **ExchangeRates.xsd**. As you can see in the schema below, we are only interested in a few pieces of information: **Base Currency** (USD or CAD in our example), **Target Currency** (again USD or CAD), **Rate**, and finally, **Date**. Creating a canonical schema will also simplify the rest of our solution.



Now that we have all the schemas defined for our message types defined or referenced, we can add the messages that we require to the orchestration.

We'll begin by adding the message **msgClosingFxRates**. That will be our raw input data from the Bank of Canada with the message type from the generated schema **ClosingFxRates.RDF**.

For each exchange rate, we'll need to first query Dynamics AX to see if it exists, thus we'll need a request message and a response message. Add a message **msgAXQueryExchangeRatesRequest**, which will be a multi-part message type **ExchangeRatesService_find_Request**, and **msgAXQueryExchangeRatesResponse** that will be a multi-part message type **ExchangeRatesService_find_Response**.

Next, we'll create the messages for the XML that we'll send and receive from Dynamics AX to create an exchange rate. Add a message **msgAXCreateExchnageRatesRequest**, which will be a multi-part message type of **ExchangeRatesService_create_Request**, and **msgAXCreateExchnageRatesResponse** that will be a multi-part message type **ExchangeRatesService_create_Response**.

Finally, we'll need to create two messages, **msgExchangeRatesUSDCAD** and **msgExchangeRatesCADUSD**, which will have the message type of the canonical schema **ExchangeRates**. These messages will contain the exchange rates for USD to CAD and for CAD to USD respectively. We'll create these two messages just to simplify our orchestration for this example. In practice, if you're going to deal with several exchange rates, you will need to add logic to the orchestration to loop through the list rates that you're interested in and have only one message of type **ExchangeRates** resent several times.

As we talked about earlier in the chapter, we'll need to know the Dynamics AX company name in order to populate the metadata in our message sent to AX with the correct source and destination endpoint. For this example, we'll hard code these two company names with **daxCADCompany** and **daxUSDCompany** variables of type System.String. We'll add a variable, **needToLoadFxRateUSD**, of type System.Boolean that we'll use in our orchestration. Also, we'll hard code the variable **serviceAccountName** with the Active Directory service account that our BizTalk host instance is running under. Finally, we need a variable named **xpathQueryCADResultsExpression** of type System.String that we'll use to determine our query response message. The following is an orchestration view on the messages and variables:



Now, we'll need to add a logical receive port and two logical send ports in our orchestration.

Create a new logical receive port for the XML closing exchange rates from the Bank of Canada with the following settings:

Port Name	ReceiveClosingFxRates_Port
Port Type Name	ReceiveClosingFxRates_PortType
Communication Pattern	One-Way
Port direction of communication	I'll always be receiving messages on this port
Port Binding	Specify Later

Create a new send-receive logical port to Query Dynamics AX with the following settings:

Port Name	QueryAXExchangeRates_Port
Port Type Name	QueryAXExchangeRates_PortType
Communication Pattern	Request-Response
Port direction of communication	I'll be sending a request and receiving a response
Port Binding	Specify Later

Create a new second send-receive logical port to send the exchange rates create message with the following settings:

Port Name	CreateAXExchangeRates_Port
Port Type Name	CreateAXExchangeRates_PortType
Communication Pattern	Request-Response
Port direction of communication	I'll be sending a request and receiving a response
Port Binding	Specify Later

Next, we'll need to create two maps to transform the Bank of Canada XML message to our canonical exchange rates messages. See the details for these two maps in the code example provided.

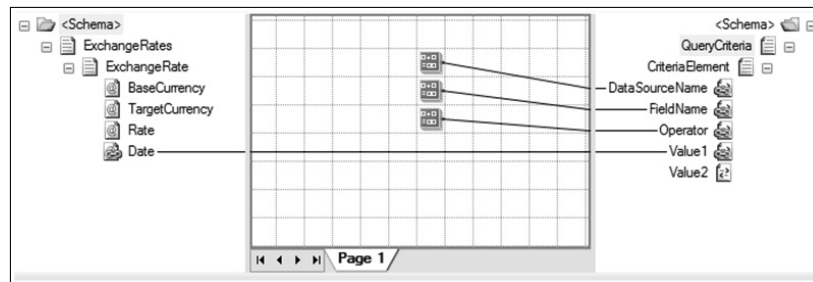
- ClosingFxRates_To_ExchangeRatesCADUSD.btm
 - CAD Base Currency – USD Target Currency
- ClosingFxRates_To_ExchangeRatesUSDCAD.btm
 - USD Base Currency – CAD Target Currency

Again, to limit the complexity of this example, we are somewhat hard-coding these two maps for our specific needs. If we needed to create many messages with different base/target currency combinations, then we would add logic to our orchestration and create a more complex map.

Now, we'll need to create two more maps to transform from our canonical ExchangeRates schema to (1) Query Dynamics AX for the particular rate and (2) ExchangeRatesService_ExchangeRates AIF schema generated by the BizTalk Dynamics AX adapter.

Dynamics AX query message

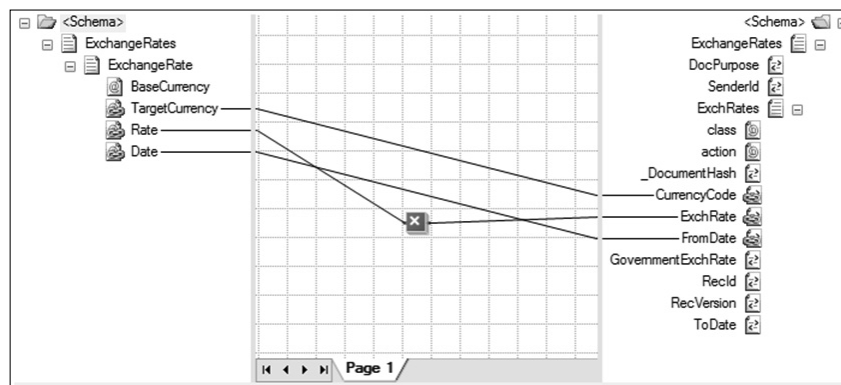
TheDynamicsAx5.QueryCriteria schema is shown in our map **ExchangeRates_To_AXQueryExchangeRatesRequest.btm**. Each company in Dynamics AX will have a base currency, thus all exchange rates will be relative to that base. For each date, there can only exist one rate for a specific target currency, thus we'll construct a query and filter specifically for the date. Our query will return all exchange rates for a specific date with many target currencies, so we'll write an XPath expression to find the one we're interested in (shown later in this walkthrough).



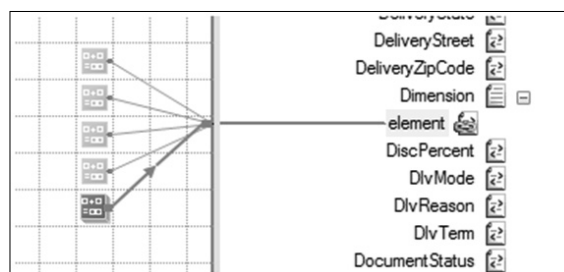
In the map above, we'll use String Concatenate functoids to store constants that we need to populate the **DataSourceName**, **FieldName**, and **Operator** elements of the **QueryCriteria** schema. The **DataSource** is the Dynamics AX 2009 table name, ExchRates. The **FieldName** is the column, the table, fromDate. Finally, the **Operator** we'll set to Equal.

Dynamics AX create message

As shown in the map `ExchangeRates_To_AXCreateExchangeRatesRequest.btm`, the transform is relatively simple. Dynamics AX exchange rates are based on a scale relative to 100 rather than 1, thus we'll need to multiply the value from our **ExchangeRates** schema by 100 using the Multiplication functoid as shown below. Notice that we can use this transform for any Base/Target currency combination.

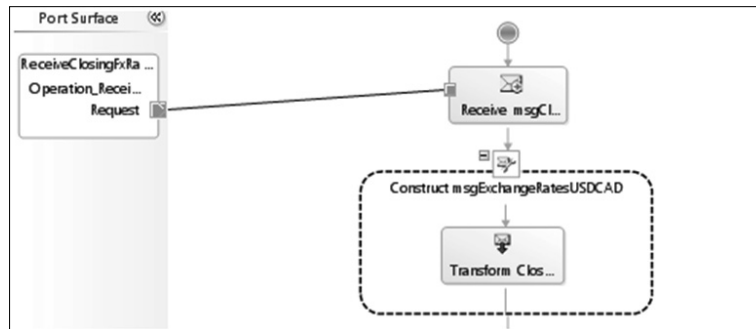


Also, notice that there are no dimensions in the **ExchangeRates** Dynamics AX schema. Typically, in most business-orientated messages, such as Sales Order, the schema would contain a single element for the dimensions. In this case, we would recommend first adding the exact number of String Concatenate functoids as dimension in your map and add links to the **element** field under the **Dimension** record in order from top to bottom in your Dynamics AX schema as shown below:



In each String Concatenate functoid as shown above, add zero length string input, and then add in the appropriate mapping logic for each dimension as an input to the relative concatenate functoid. This will ensure your output message will have the correct number of dimension elements and in the correct order, each time you add/delete/change a link in your map. To keep your maps readable, as a standard practice, you may choose to create a new page for mapping dimensions.

Continuing to build our orchestration, we'll add a **Receive** shape at the top to receive the **msgClosingFxRtes** message from the Bank of Canada and set the **Activate** property to True. Next, we'll add two **Construct** shapes to construct canonical messages **msgExchangeRatesUSDCAD** and **msgExchangeRatesCADUSD** each containing a transform shape using **ClosingFxRates_To_ExchangeRatesCADUSD.btm** and **ClosingFxRates_To_ExchangeRatesCADUSD.btm** respectively.



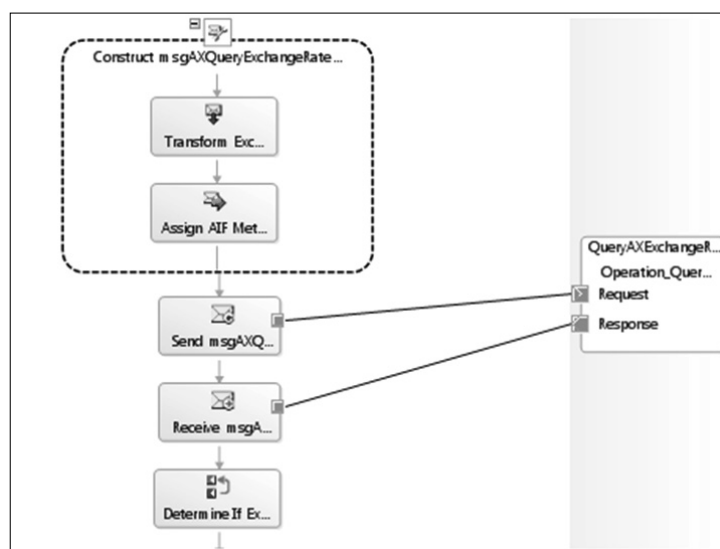
Orchestration setup

Now, we'll add one more **Construct** shape to construct our first **msgAXQueryExchangeRatesRequest** message (this will be to query our USD-based Dynamics AX company). We'll add a **Transform** shape for **ExchangeRates_To_AXQueryExchangeRatesRequest.btm** and use the **msgExchangeRatesUSDCAD** as an input to the transform. After the transform shape, we'll need to add in Message Assignment shape to add out AIF metadata. The contents of the Message Assignment Shape are:

```
msgAXQueryExchangeRatesRequest (DynamicsAx5.MessageId) = System.String.
Format (" {0:B} ", System.Guid.NewGuid() );
msgAXQueryExchangeRatesRequest (DynamicsAx5.Action) = "http://schemas.
microsoft.com/dynamics/2008/01/services/ExchangeRatesService/find";
msgAXQueryExchangeRatesRequest (DynamicsAx5.DestinationEndpoint) =
daxUSDCompany + "LocalEndpoint";
msgAXQueryExchangeRatesRequest (DynamicsAx5.SourceEndpoint) =
daxUSDCompany + "Endpoint";
msgAXQueryExchangeRatesRequest (DynamicsAx5.SourceEndpointUser) =
serviceAccountName;
```

This step is where we can see that our naming conversion in the AIF setup is very important. The AIF metadata in our message determines which **SourceEndpoint** and **DestinationEndpoint** (**Endpoint** and **Localendpoint** tables in the AIF setup respectively) this message is intended for. It also determines which AIF **Action** the message is executed against. The **SourceEndpointUser** must also explicitly set in the metadata for each message.

Now, we'll add in a send shape followed by a **Receive** shape to **Send** the **msgAXQueryExchangeRatesRequest** and receive the **msgAXQueryExchangeRatesResponse** messages. Now, connect the send and receive shapes to the **QueryAXExchangeRates_Port** port as shown in the falling image:



Next, we'll add in an expression shape. Since we'll receive ALL the exchange rates corresponding to a **FromDate** that we specified in our query message, we need to determine whether the USD-based company is a corresponding entry for the CAD currency. The contents of the expression shape to do this are:

```
xpathQueryCADResultsExpression = System.String.Format("/*[local-
name()='ExchangeRates']/*[local-name()='ExchRates']/*[local-
name()='CurrencyCode' and text()='{0}']", "CAD");

// If we find a result, then don't need to load
if (xpath(msgAXQueryExchangeRatesResponse.ReturnValue,
xpathQueryCADResultsExpression) != null)
{
    needToLoadFxRate = false;
}
```

Since the query returns a list of exchange rates, we'll use XPath to quickly determine whether the rate for our base currency exists for the date. You'll probably notice that we've hard coded the **CAD** currency code in our **xpathQueryCADResultsExpression**. In a more complicated scenario with multiple target currencies, you'll need to add logic to find the corresponding currency code you're interested in. Also, since it's very possible that each day we won't have a rate for each currency (due to statutory holidays for example); you will need to include this logic for each currency/date combination.

Now, we'll add in a **Decide** shape based on the **needToLoadFxRate** Boolean variable we set in the expression shape. If we've found a rate for that date we'll do nothing, otherwise we'll construct the **msgAXCreateExchangeRatesRequest** message and send it to Dynamics AX.

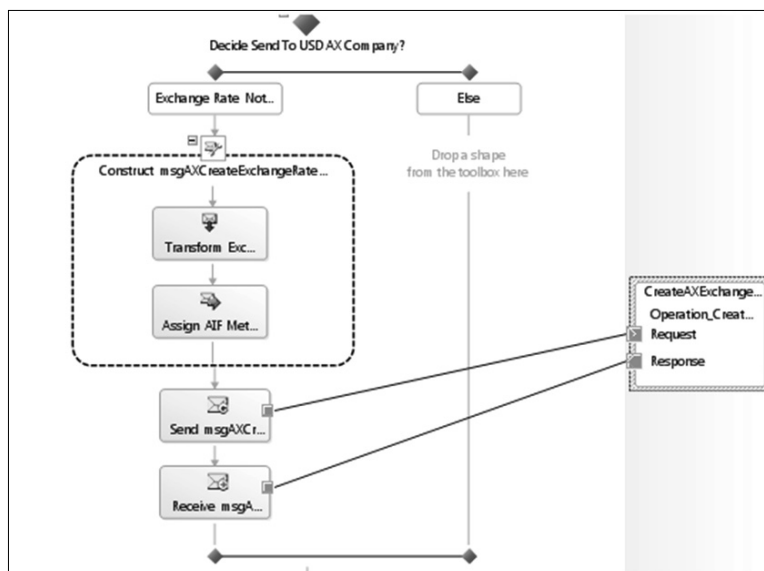
In the branch of the decide shape where the rate was not found, we'll add in a Construct Message shape containing a transform followed by a Message Assignment shape. The map we'll use is **ExchangeRates_To_AXCreateExchangeRatesRequest.btm** with the **msgExchangeRatesUSDCAD** message as an input.

In the expression shape, we'll need again to add the AIF metadata. The contents are:

```
msgAXCreateExchangeRatesRequest(DynamicsAx5.MessageId) = System.  
String.Format("{0:B}", System.Guid.NewGuid());  
msgAXCreateExchangeRatesRequest(DynamicsAx5.Action) = "http://schemas.  
microsoft.com/dynamics/2008/01/services/ExchangeRatesService/create";  
msgAXCreateExchangeRatesRequest(DynamicsAx5.DestinationEndpoint) =  
daxUSDCompany + "LocalEndpoint";  
msgAXCreateExchangeRatesRequest(DynamicsAx5.SourceEndpoint) =  
daxUSDCompany + "Endpoint";  
msgAXCreateExchangeRatesRequest(DynamicsAx5.SourceEndpointUser) =  
serviceAccountName;
```

Notice that only the **DynamicsAX5.Action** is different from the previous query message sent to AX. Also note the use of the **daxUSDCompany** variable here.

Now, add in a **Send** shape followed by a **Receive** shape to send the **msgAXCreateExchangeRatesRequest** and receive the **msgAXCreateExchangeRatesResponse** messages. Now, connect the send and receive shapes to the **CreateAXExchangeRates_Port**.



This completes the steps for sending a USD – CAD exchange rate to our USD-based Dynamics AX Company. To complete our example, we'll now add in the logic for the USD – CAD exchange rate to our CAD-based Dynamics AX Company. So, we'll repeat the above from the Orchestration Setup step using the **msgExchangeRatesCADUSD** and the **daxCADCompany** in place of **msgExchangeRatesUSDCAD** and **daxUSDCompany**. See the complete code example.

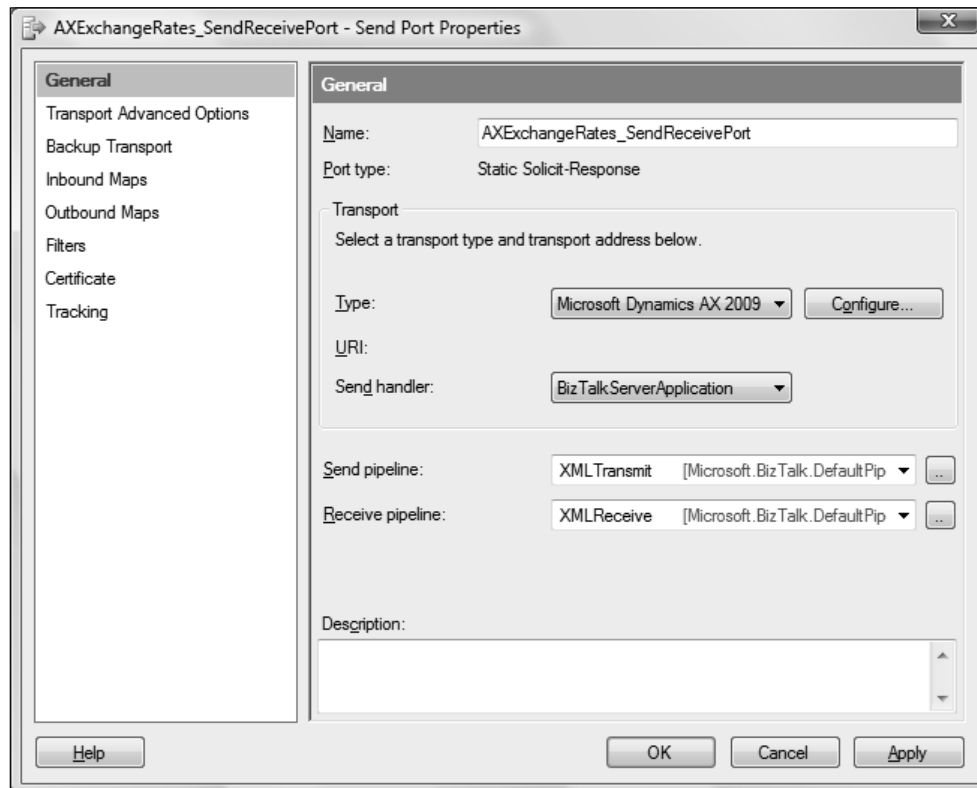
In the example source code provided, there's no error handling built into the orchestration. If there's an error in sending a message to Dynamics AX, the port will throw an exception. Quite often, especially in versions of Dynamics AX prior to AX 2009, the error message can be quite generic or cryptic at times. Connectivity errors will have some meaningful error message; however, logical errors will not. This is because the AIF module is an external facing interface, so the detailed error message most helpful is found in the **Exceptions (Basic | Periodic | Application Integration Framework | Exceptions)** form in the AIF module.

Now that we can build and deploy the solution, the next step will be to set up our Port configuration in the BizTalk Server Administration Console.

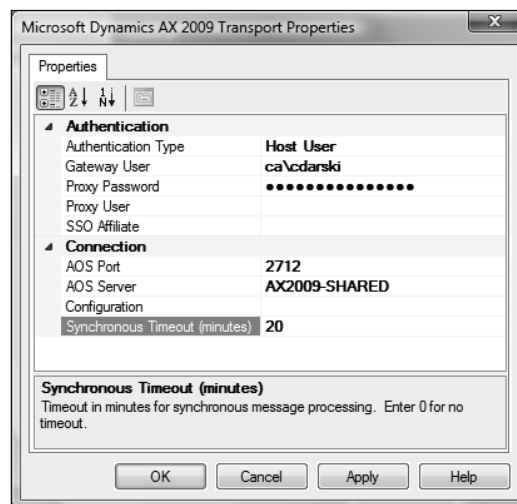
Port configuration

We'll begin by creating a **Static Solicit-Response Send Port** and type in a **Name** of **AXExchangeRates_SendReceivePort**.

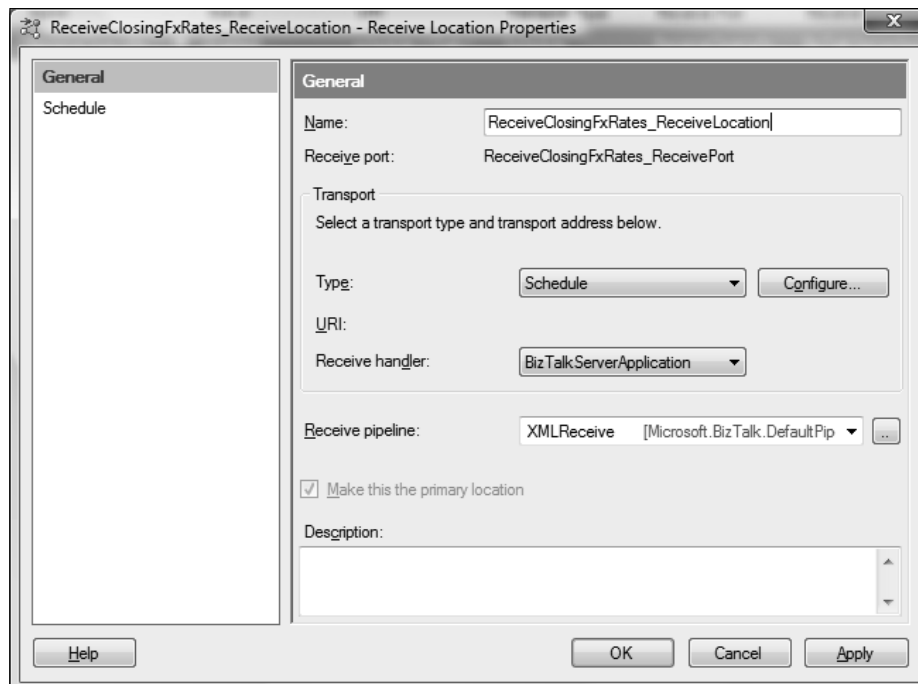
1. We'll set the **Type** to **Microsoft Dynamics AX 2009**, and set the **Send pipeline** and **Receive pipeline** to **XML Transmit** and **XML Receive** respectively.



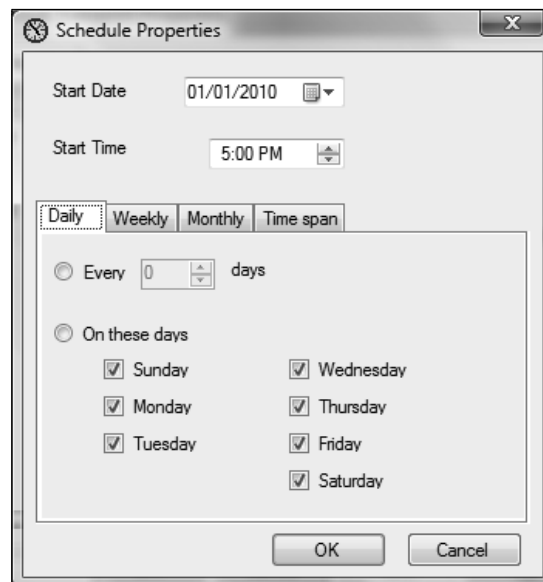
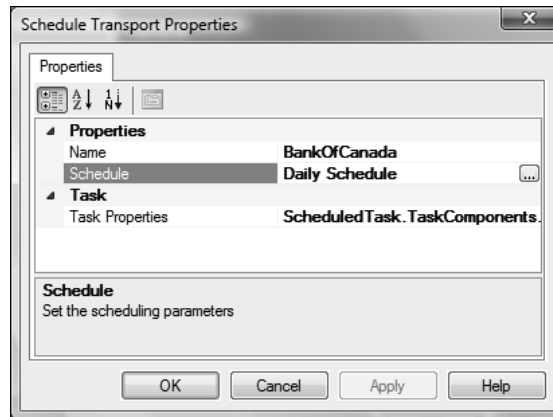
2. Click **Configure** to bring up the **Microsoft Dynamics AX 2009 Transport Properties** window.
3. Here, we'll select an **Authentication Type** to **Host User** from the drop-down list and set the **Gateway User** to our Active Directory service account that our BizTalk Host instance is running under. We'll set the **AOS Port** to **2712** and **AOS Server** to our Dynamics AX 2009 server name **AX 2009-SHARED**. We can leave the default Synchronous Timeout to 20 minutes.



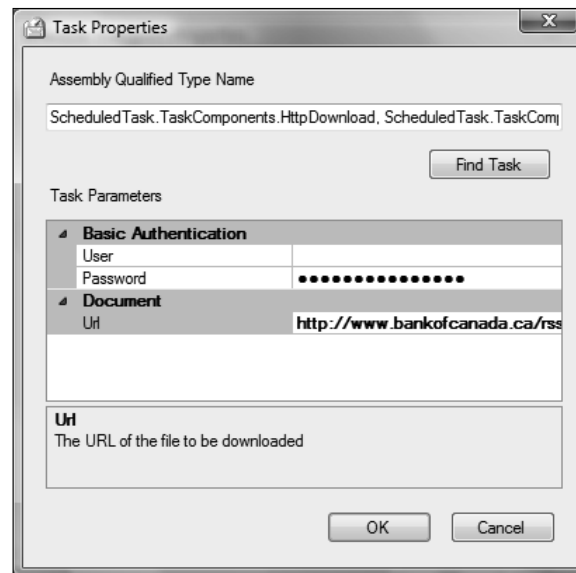
4. Click **Ok** to save this configuration.
5. Next, we'll continue with the port configuration by creating a **One-way Receive Port**, named **ReceiveClosingFxRates_ReceivePort**, with a receive location named **ReceiveClosingFxRates_ReceiveLocation**. We'll use a **Type** of **Schedule** and set the **Receive pipeline** to **XML Receive**.



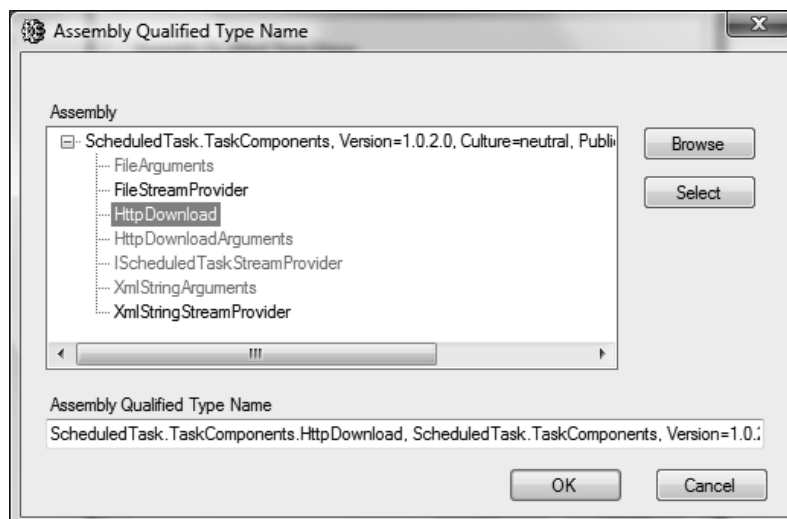
- Next, click **Configure** to bring up the Schedule Transport Properties window. Here, we'll type in a **Name** of **Bank of Canada**, and set the configuration **Schedule** to **Daily Schedule** with a **Start Time** of **5:00 pm** (assume our server time is EST). Note we'll configure it to run daily, regardless of weekends or bank holidays (when no new rates are available), as our orchestration does the query to AX before attempting to create a new exchange rate.



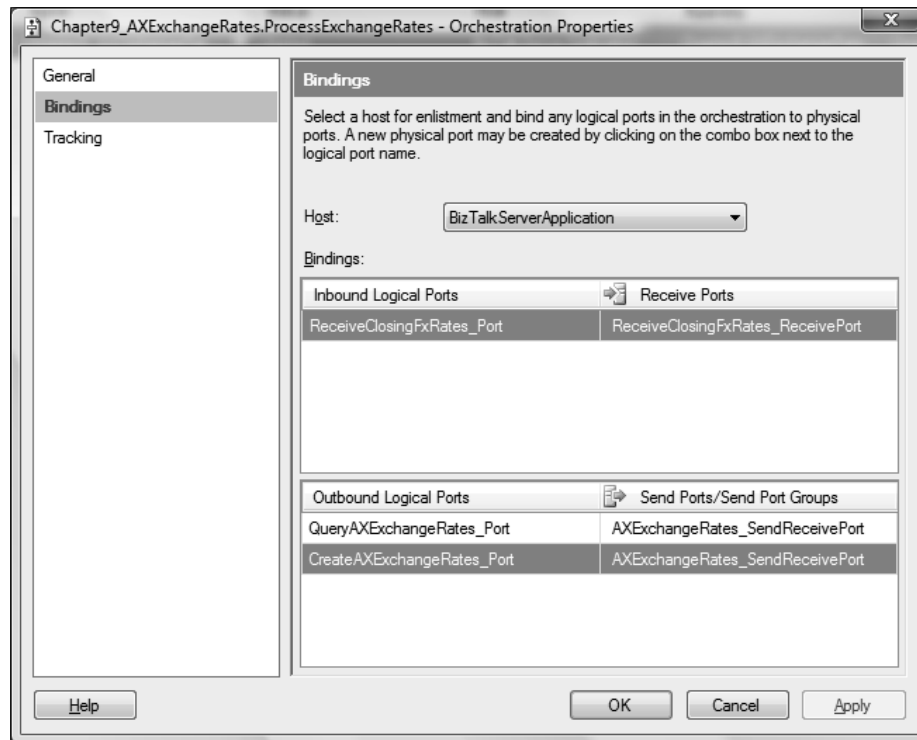
- Now, click on the **Task properties** field from the **Schedule Transport Properties** window to bring up the **Task Properties** window.



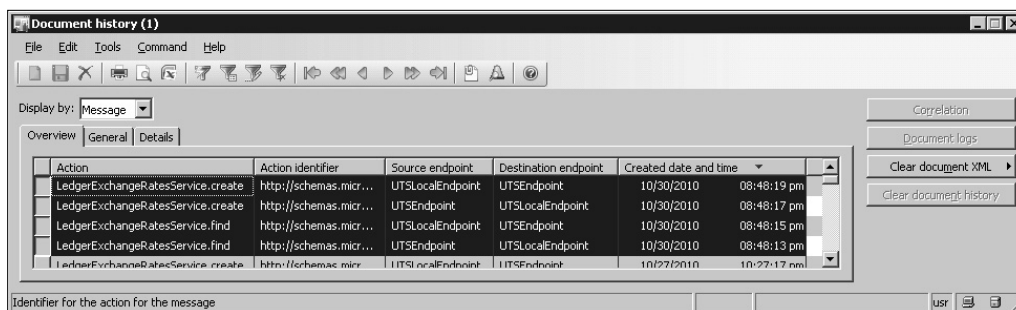
8. Here, click **Find Task**, which brings up the **Assembly Qualified Type Name** window. Click on the **Browse** button, select the **ScheduledTask.TaskComponents.dll** assembly and click open. Now, double click **HttpDownload** and click **Select**. Click **Ok**. Finally, in the **Task Properties** window, type in the URL `http://www.bankofcanada.ca/rss/fx/close/fx-close.xml`, leave the user name blank as no authentication is required. Click **OK** twice and save.



9. On the Orchestration Properties, we'll assign the logical receive port to the physical port we just created and set both Outbound Logical Ports to the physical send port to connect to Dynamics AX 2009.



10. After we start our application, it's now time to test it. We can either wait until 5:00 pm, or simply change the **Start Time** in the **Schedule Properties** window to a couple minutes past the current time. You'll see a log message in the Windows event log indicating the configuration has changed for the schedule task.
11. After the scheduled task adapter has fired, and the orchestration has successfully completed, navigate to the **Document history** form (**Basic | Periodic | Application Integration Framework | Document History**) in Dynamics AX 2009. Here, you'll see a log for all four messages that were sent and received from our BizTalk application. You can also view the XML message by highlighting one of the Document history log entries, clicking **Document logs** to bring up the **Document log** form, and finally clicking **View XML**.



12. Verify that the exchange rate has been created in the table (**General Ledger | Setup | Exchange Rates**).
13. Note that the **Document history** form is company specific and will log all transactions both successful and those that went to error. However, the **Exceptions** form log is not company specific. Thus, attempting to link an exception to the document history entry can prove to be difficult.

Asynchronous walkthrough example— Dynamics AX message outflow

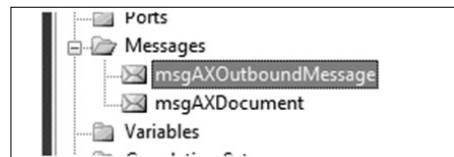
Now that the setup for sending data using AIF XML documents to AX is complete, we can also use the BizTalk adapter to retrieve data from Dynamics AX via this AIF module. This is done using the AIF **Queue manager**. Throughout Dynamics AX, there are **Send electronically** buttons that can allow you to push data into the AIF Queue with an **Outbound** direction. Similarly, asynchronous incoming messages have an **Inbound** direction parameter.

We found that rather than repeating code in several BizTalk solutions when you need to retrieve data from the AIF Queue, it's relatively simple to create a general solution to accomplish this. This solution will retrieve all data via the BizTalk Dynamics AX adapter by polling the Queue at a set interval of time. The minimum polling interval is 1 minute, thus any messages you put in the AIF Queue will not be immediately consumed by BizTalk. The complete solution (Chapter9-AXMessageOutflow) is included with the source code for this chapter.

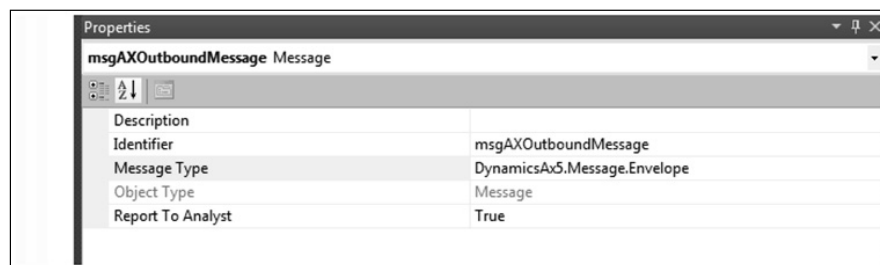
We'll start by creating a new BizTalk project, **Chapter9-AXMessageOutflow**, in Visual Studio. Add in a new orchestration, **ProcessOutboundAXMessage.odx**, which will be the only orchestration required for this example. Also, we'll need to add reference to the **Microsoft.Dynamics.BizTalk.Adapter.Schemas** assembly and sign the project with a strong name key.

Message setup

Next, we'll add two messages to our orchestration: **msgAXOutboundMessage** and **msgAXDocument**. These will be the only two messages required in this example.



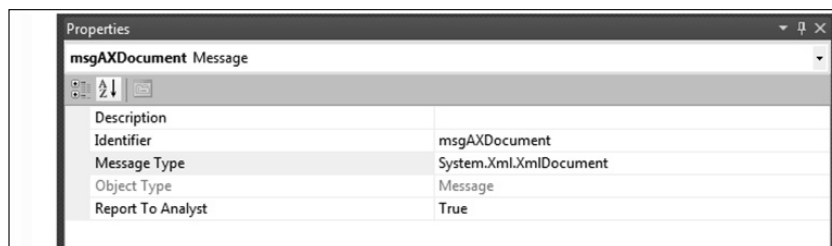
The first message, **msgAXOutboundMessage**, is of type **DynamicsAX5.Message.Envelope**. The schema is located in the referenced Microsoft.Dynamics.BizTalk.Adapter.Schemas assembly.



All outbound messages from the AIF Queue are of this type. As you can see from the sample screenshot below, we have some metadata in the header node but what we are really interested in is the XML contents of the Body node. The contents of the **MessageParts** node in the Body node will be of type **ExchangeRatesService_ExchangeRates.xsd** that we used in the previous example to send this same message to AX. Thus, all the schemas we require for both inbound and outbound transactions can be generated using the adapter.



For the second message, since we don't want to specify a document type, we will use **System.Xml.XmlDocument** for the **Message Type**.



Using the **System.Xml.XmlDocument** message type allows for great flexibility in this solution. We can push any message to the AIF queue, and no changes to this BizTalk application are required. Only changes to consuming applications may need to add the AX schema of the message in order to process it.

Orchestration setup

Next, we create a new a logical port that will receive all messages from Dynamics AX via the AIF Queue with the following settings:

Port Name	ReceiveAXOutboundMessage_Port
Port Type Name	ReceiveAXOutboundMessage_PortType
Communication Pattern	One-Way
Port direction of communication	I'll always be receiving messages on this port
Port Binding	Specify Later

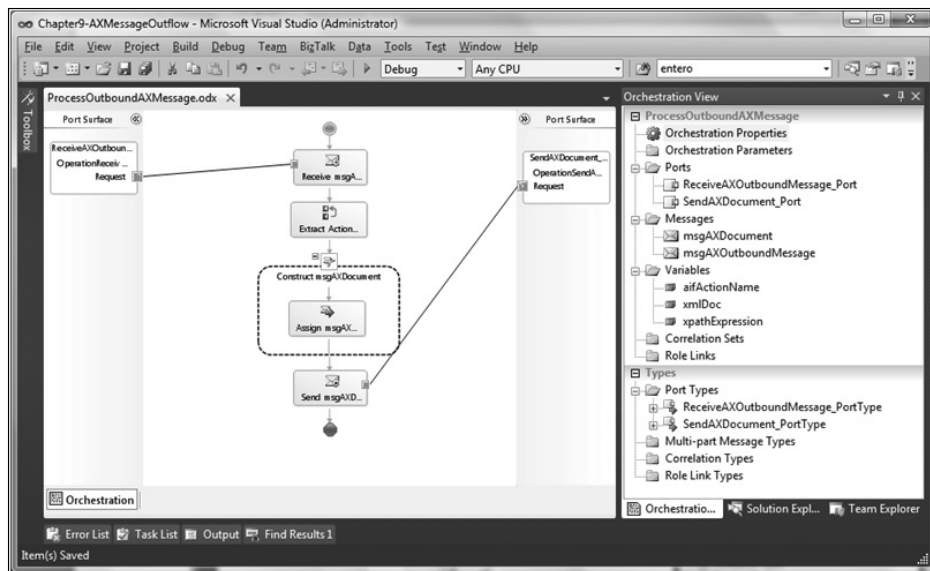
Also, create a new send port. For this example, we'll just send to a folder drop using the FILE adapter so that we can easily view the XML documents. In practice, other BizTalk applications will most likely process these messages, so you may choose to modify the send port to meet your requirements. Send port settings:

Port Name	SendAXDocument_Port
Port Type Name	SendAXDocument_PortType
Communication Pattern	One-Way
Port direction of communication	I'll always be sending messages on this port
Port Binding	Specify Later

Next, we will need to add the following to the orchestration:

- Receive shape (receive msgAXOutboundMessage message)
- Expression shape (determine the file name for msgAXDocument)
- Message assignment (construct msgAXDocument message)
- Send shape (send msgAXDocument message)

We'll also add two variables (**aifActionName** and **xpathExpression**) of type System.String and **xmlDoc** of type System.Xml.XmlDocument.



In the expression shape, we want to extract the AIF Action so that we can name the outbound XML documents in a similar fashion. This will allow us to easily identify the message type from AX.

Next, we'll put the following inside expression shape below receive to extract the AIF action name, which we'll use later in our outbound message:

```
aifActionName = msgAXOutboundMessage(DynamicsAx5.Action);
aifActionName = aifActionName.Substring(55,aifActionName.
LastIndexOf('/') - 55);
```

Now, we need to extract the contents of the body message, which is the XML document that we are interested in. Inside the message assignment shape, we will use XPath to extract the message. What we are interested in is the contents of the Body node in the DynamicsAX5.Message.Envelope message we will receive from AX via the AIF Queue. Add the following code inside the assignment shape to extract the XML, assign it to the message we are sending out, and set a common name that we can use in our send port:

```
// Extract Contents of Body node in Envelope Message
xpathExpression = "/*[local-name()='Envelope' and namespace-
uri()='http://schemas.microsoft.com/dynamics/2008/01/documents/
Message']/*[local-name()='Body'
and namespace-uri()='http://schemas.microsoft.com/dynamics/2008/01/
documents/Message']";
```

```
xmlDoc = xpath(msgAXOutboundMessage, xpathExpression);

// Extract the XML we are interested in
xmlDoc.LoadXml(xmlDoc.FirstChild.FirstChild.InnerXml);

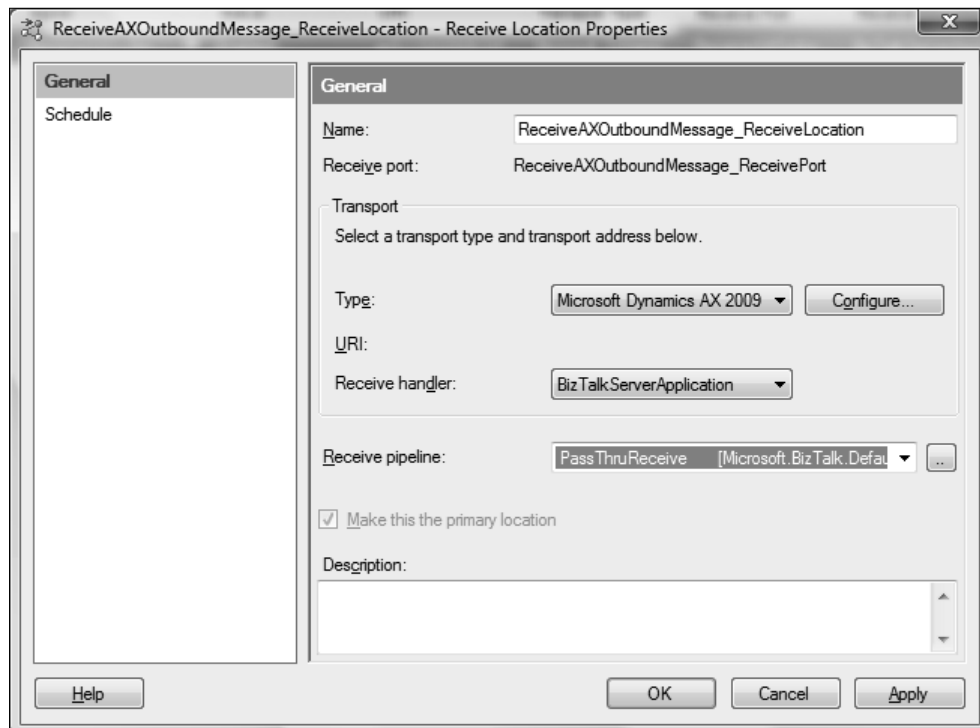
// Set the message to the XML Document
msgAXDocument = xmlDoc;

// Assign FILE.ReceivedFileNameproperty
msgAXDocument(FILE.ReceivedFileName) = aifActionName;
```

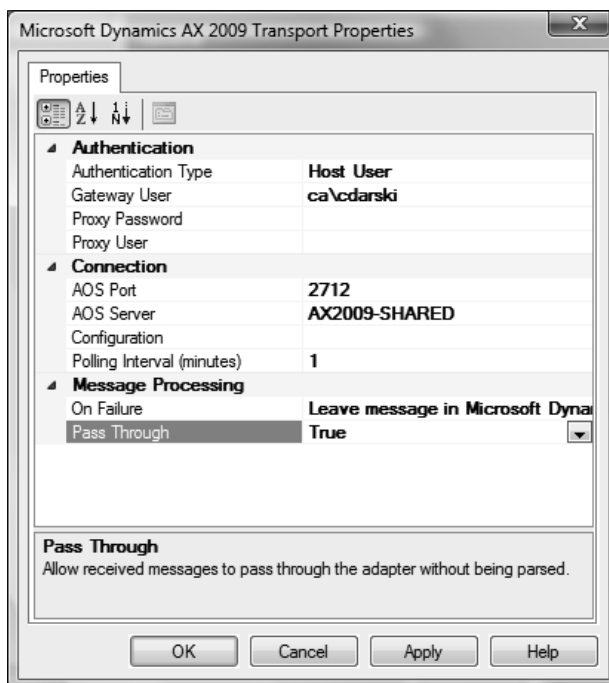
We can now build and deploy the solution. Next, we'll go through setting up the physical ports and binding them to our orchestration.

Port configuration

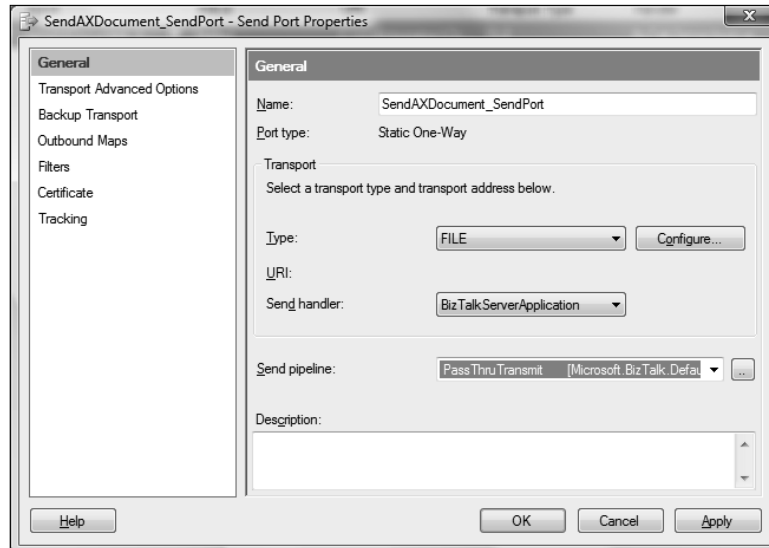
Create a new receive port, **ReceiveAxOuboundMessage_ReceivePort**, and a new receive location, **ReceiveAxOuboundMessage_ReceiveLocation**. Set the type to Microsoft Dynamics AX 2009, and we need to select a **PassThruReceive** for the **Receive pipeline**.



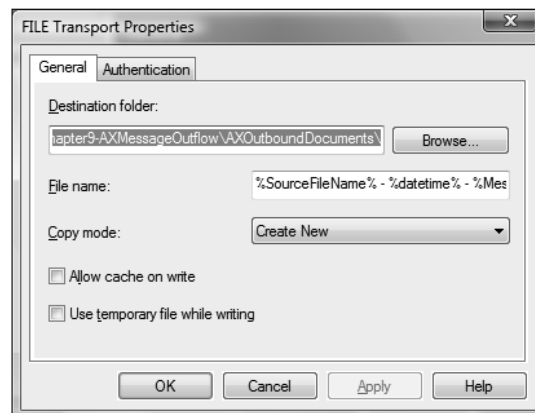
Click **Configure** to bring up the **Microsoft Dynamics AX 2009 Transport Properties** window. Here, we'll set the **Authentication Type** to **Host User**, set the **Gateway User** to the Active Directory account that our BizTalk Host Instance is running under, **AOS Port** to **2712**, and **AOS Server** to our AX server name (**AX 2009-SHARED**). Also, we set the **Polling interval** to **1** minute, and finally, we must set the **Pass Through** to **True**. This allows our solution to use the adapter without parsing the message. If we set this to **False**, the adapter will attempt to resolve the message type in the AIF queue before removing it and set it to an error status if it's unsuccessful.



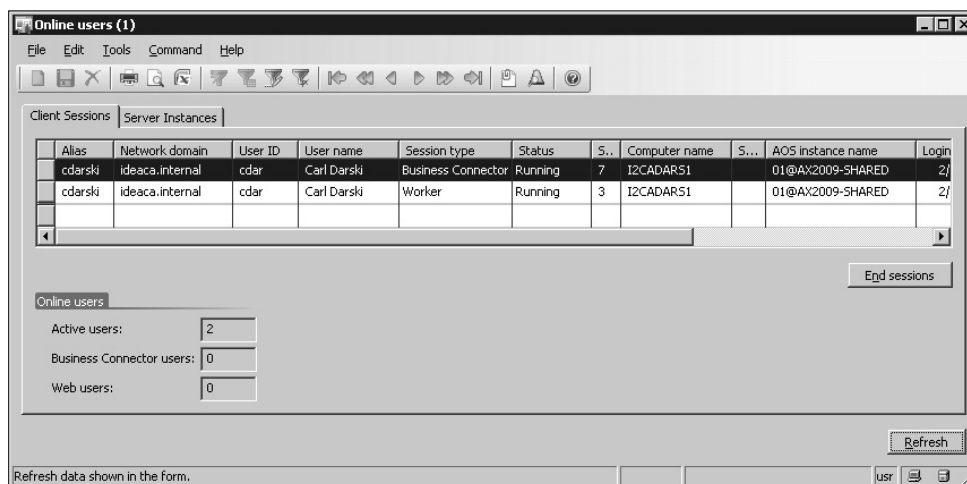
Next, we'll create a **Static One-way Send Port** with **Name** set to **SendAxDocument_SendPort** to bind to our logical send port. Set the **Type** to **File**, and **Send pipeline** to **PassThruTransmit** as we'll perhaps be sending different message types out as XML documents.



Click **Configure** to bring up the **FILE Transport Properties** window. Set the **Destination folder** to **C:\LOBIntegration\Chapter09\Chapter9-AXMessageOutflow\AXOutboundDocuments** and File name to **%SourceFileName% - %datetime% - %MessageID%.xml**. The **SourceFileName** for each message type will be identical as it was set in our orchestration above to the Dynamics AX action name. The **datetime** is added to the file name property because we often find it helpful in troubleshooting errors or sometimes it is required for future integration logic.

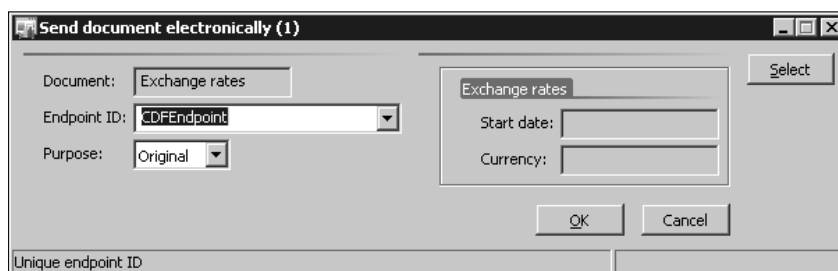


Now, we bind the two ports to the orchestration and start the BizTalk application. Since there's nothing in the AIF Queue for us to pick up, we won't see anything much on the BizTalk server side. However, if you log into Dynamics AX, go to **Online users (Administration | Online users)** and you can verify that your BizTalk application is indeed connected. There will be one connect, with **Session type** of **Worker** that will verify your connection. Click the refresh button continuously, and you will see the **Session type** of **Business Connector** appear every minute (our application polling interval set on the port).



Now that we have verified that we have connectivity, we need to send a message to the AIF Queue in order to run a test. Disable the receive location for now so that we can see what's happening inside AX before it's removed from the Queue.

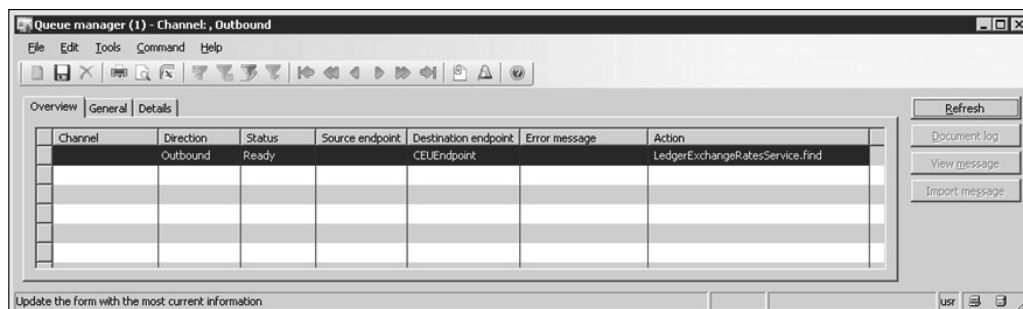
There are many forms in Dynamics AX (Exchange Rates for example) where you'll see a **Send Electronically** button. Click on the button from the Exchange Rates table and you'll see the **Send document electronically** window pop up. You need to select an **Endpoint ID** and can put filter criteria from the Select button if you wish. Notice that only Endpoints that have the LedgerExchangeRates AIF service actions enable will appear in the drop-down list.



Click **Ok** to send the message to the AIF Queue. Now, try and click on **Send electronically** on the **Chart of Accounts** table (**General Ledger | Chart of Accounts**) and notice the error message as shown in the following screenshot. Since we have not activated this AIF Service, nor added the services' actions to any Endpoints, Dynamics AX does not allow us to send out this message.



Next, we'll open up the AIF **Queue manager** form (**Basic | Setup | AIF | Queue manager**) and here we can see the message we just sent from the Exchange Rates tables. Notice that the log entry is missing Channel and Source Endpoint. This is because by default, the **Send Electronically** button is an asych process that requires a batch to be run in order to specify the Channel (BizTalk in our case) based on the configuration of the **Destination endpoint**.

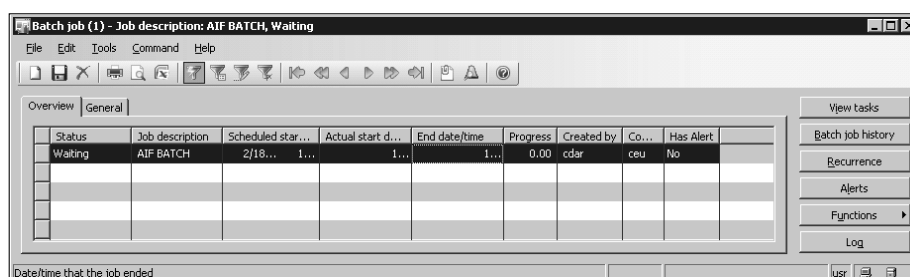


Batch setup in Dynamics AX

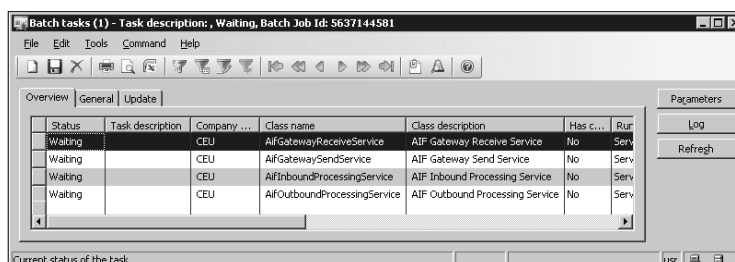
Now we need to set up a batch in order to send this **Outbound** message through the BizTalk channel. If we had previously sent out messages from BizTalk in an asynchronous mode, it would also have appeared here and required a batch in order to be processed. Note it is also possible to customize methods in Dynamics AX that still write messages to the AIF queue, but do not require a batch in order to be picked up by our BizTalk application.

Dynamics AX 2009 allows for batch jobs to be run on the server; however, previous versions required an active client in order to run batch jobs. As you can imagine, keeping open an active client to run a continuous batch job, or manually starting a batch job when required can be very cumbersome in a live production environment. Even with server-side batch job capability, you opt to only do synchronous integration to eliminate the need for any batch job altogether.

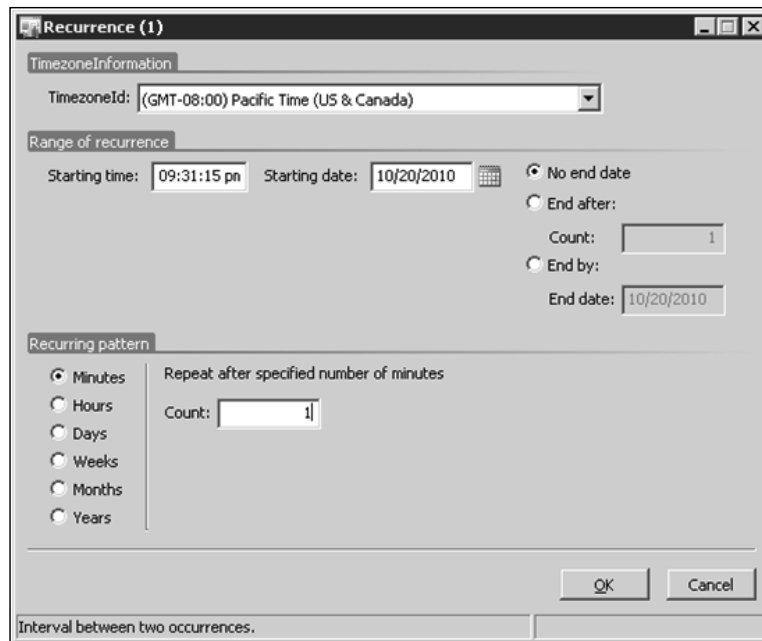
So, we'll need to create a batch for this example. First, we'll create a new entry in the **Batch job (Basic | Inquiries | Batch job)** table. We'll type in Job description of **AIF BATCH** and hit **Save**.



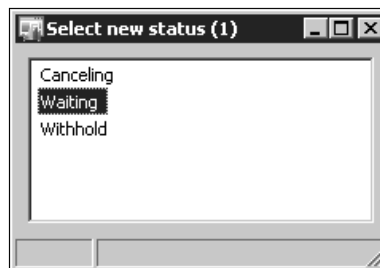
Next, click on **View Tasks** button, which will bring up the **Batch tasks** form. Here, we need to add four new entries, one for each AIF class (**AifOutboundProcesssingService**, **AifInboundProcessingService**, **AifGatewaySendService**, **AifGatewayReceiveService**) from the drop-down list and set the **Company accounts** for each. Since the batch tasks are company specific, you can imagine this may add significant management overhead to your integration processes.



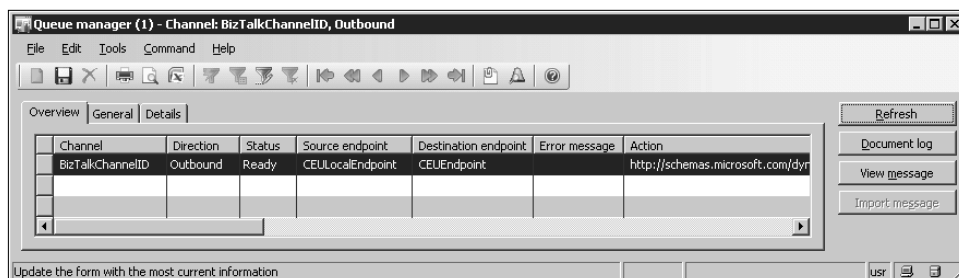
After configuring the batch tasks, save and close the window. Go back to the **Batch Job** form and click on the **Recurrence** button that will be the window below. Here, we can configure several parameters, but for our example simply set the **Recurring pattern Count** to 1 and select **Minutes**.

The image shows a Windows-style dialog box titled "Recurrence (1)". It has three tabs: "TimezoneInformation", "Range of recurrence", and "Recurring pattern". The "TimezoneInformation" tab is active, showing a dropdown for "TimezoneId" set to "(GMT-08:00) Pacific Time (US & Canada)". The "Range of recurrence" tab shows "Starting time" as "09:31:15 pm", "Starting date" as "10/20/2010", and "No end date" selected. The "Recurring pattern" tab shows "Minutes" selected, with "Repeat after specified number of minutes" and "Count" set to "1". There are "OK" and "Cancel" buttons at the bottom right, and a label "Interval between two occurrences." at the bottom left.

Click **Ok**. Again back on the **Batch job** form, click on the **Functions** button, which brings up the **Select new status** window.

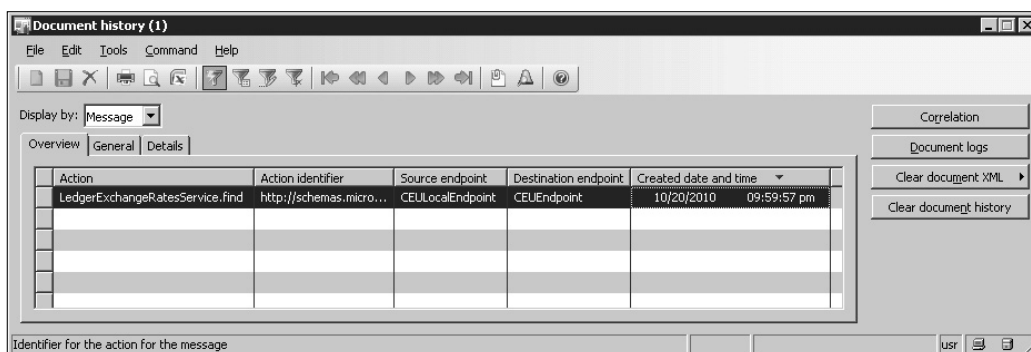
The image shows a Windows-style dialog box titled "Select new status (1)". It contains a list box with three items: "Canceling", "Waiting", and "Withhold". The "Waiting" item is currently selected and highlighted. There are "OK" and "Cancel" buttons at the bottom.

Select **Waiting** to activate the batch job. Now, we go back to the AIF **Queue manager** form. Click refresh after approximately one minute and you'll see the same **Outbound** record we previously sent. However, you'll notice the **Channel** and **Source endpoint** have now been populated. Also note that the status is still set to **Ready**.

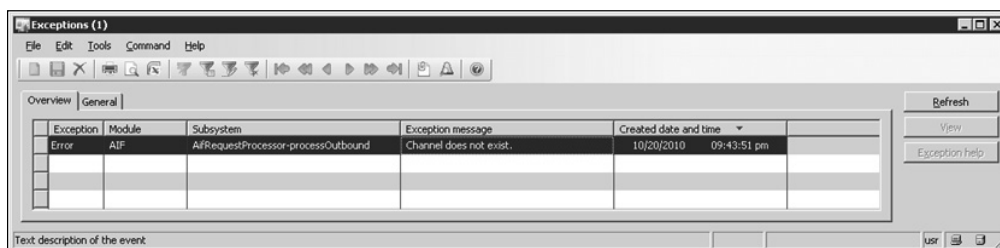


Now, we go back and enable the receive location on your BizTalk solution. You should see this record be removed from AIF Queue, and find the XML document in the folder specified on the send port.

Note that this record we just pulled from the AIF Queue will appear in the AIF **Document history** table. You can view the XML by selecting a record, clicking on **Document logs**, and then clicking on **View XML**.



If we have any errors, a detailed message will appear in the AIF **Exceptions (Basic | Periodic | Application Integration Framework | Exceptions)** form and the status of the outbound record in the Queue manager will change to Error. It's possible to fix the error and then change the status back to Ready.



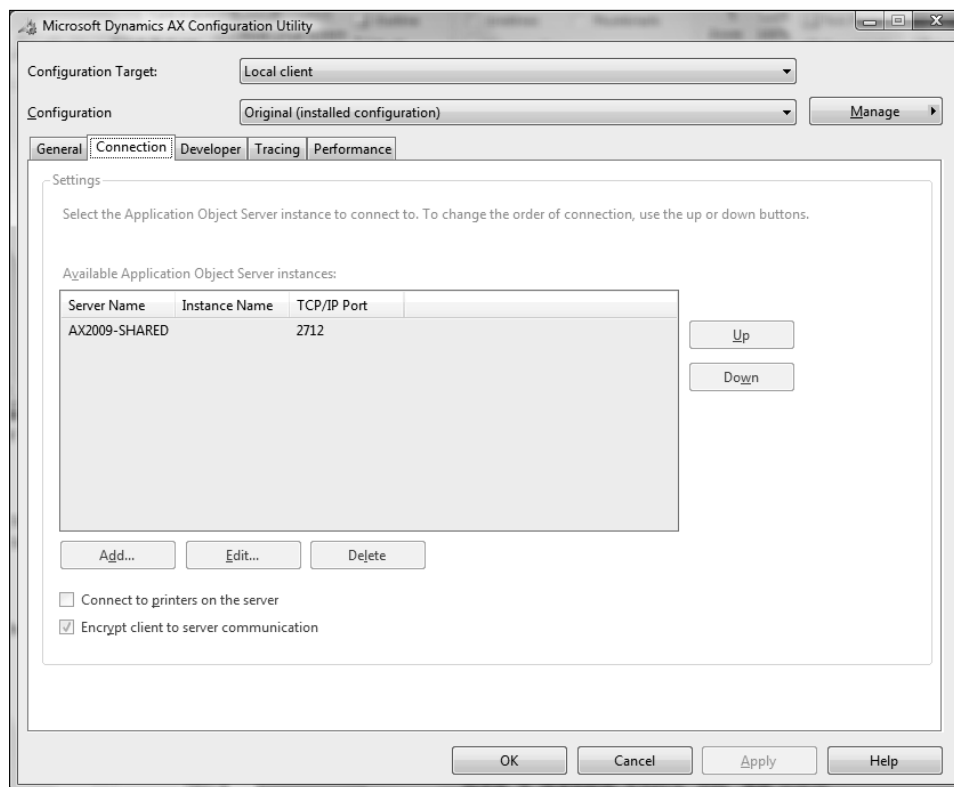
For example, if we had not specified an outbound channel in our Endpoint configuration, the error seen above would have appeared when the batch job executed. Thus, we could go back, add in the BizTalk channel to the configuration, change the status of the message in the Queue to Ready; then, the next time the batch job executed, the message would be ready to be extracted by our BizTalk application.

Using the .NET business connector

The .NET business connector allows for integration with AX without BizTalk. The BizTalk Dynamics Adapter actually leverages the .NET business connector. However, it runs independently and does not require BizTalk to be installed.

Install the business connector from the Dynamics AX setup install. You'll see it under the optional components on the AX install screen. You do not need to install the Dynamics AX client or any other Dynamics components in order for it to operate. Take caution that you install the correct version that exactly matches the version of your AX environment; otherwise at runtime you receive a generic error message in your windows event log with no clear indication as to what is wrong.

The business connector acts as a GUI-less client that can be called from a custom .NET assembly. On installation, you'll be prompted for a Dynamics environment that will be used for an initial configuration; this can be changed after installation. As shown in the example included with the source code for this chapter, the AX instance to which the business connector is configured to can be specified at runtime. The best practice is to use the configuration file that your Dynamics AX system administrator should provide (this is the same configuration file that the AX client uses in order to establish a connection to the AOS). If at runtime, you do not specify a configuration, the default configuration from the business connector setup will be used. Run the **Microsoft Dynamics AX Configuration Utility** to manage your default configuration, shown as follows:



In the attached console application example (complete solution **Chapter9-AXBusinessConnector** is included with the source code for this chapter) we create a simple session object to query a table in Dynamics AX. A more practical example would be to call a static method with parameters that returns, for example, an `AxaptaContainer` object. See <http://msdn.microsoft.com/en-us/library/aa659581.aspx> for more details on the use of the .NET Business Connector as we could write an entire chapter on this topic alone which is not the intent of this book.

Note that it's very important to correctly dispose of each connection as the .NET business connector will not handle this on its own if an error occurs. As you can imagine, going unmonitored could have disastrous consequences that could result in requiring a restart of your AOS.

Each connection is similar to a connection using the Dynamics AX client in terms of security. Thus, be aware of possible performance issues if your AX user is a member of multiple security groups. Also, keep in mind that classes in Dynamics AX are not thread safe, so verify that your integration logic takes this into account. Use the .NET business connector with your custom code with care.

This example simply does a query on an AX table for the company name, however, it does point out a few key concepts including disposal of your connection, use of retries, and authentication. In the **Program.cs** file, simply replace the constants, build, and run.



Note we need to reference **Microsoft.Dynamics.BusinessConnectorNet** and **System.Web** assemblies, and the code provided has a Target framework set to **.Net Framework 3.5**.

Other development and configuration notes

Development

It is very possible that your integration needs with Dynamics AX will require some sort of customization in order to take advantage of the AIF module. Many organizations are very much against over customizing third-party systems. However, Dynamics AX was built on the concept of customizations, thus a happy medium can usually be reached in custom development requirements for integration.

A custom service can be created to integrate on practically any Dynamics AX object, which will allow for find, create, read, update, and delete actions to be customized. Dynamics AX versions prior to AX 4.0 SP2 do not allow for delete actions to be performed in AIF actions. If required, you can customize the existing code functionality to perform delete action if required, for example, by setting a custom flag. However, you will still need to use existing actions that are exposed.

For development, it is recommended the user under which Visual Studio is running be a member of the Admin security group in your development instance of Dynamics AX. Your production BizTalk service account can also be a member of the Admin role, which makes it a powerful account, but eliminates the need for granular security requirements.

Configuration

Typically, your organization will have multiple AOS instances running in order to ensure high availability and acceptable online performance. You may be able to schedule all your high resource demand integration tasks during off-peak hours by running batch job processes as mentioned previously in this chapter. However, typically, business level integration is required (or preferred) in near real time. In order to minimize the impact to online users, you may want to consider creating a separate AOS for integration tasks only.

On the AIF services form, the Refresh button will scan the entire **AOT (Application Object Tree)** for AIF Services and populate all the found results. If the user logged in only has partial permission to a particular service that joins multiple objects, then the schema generated may only be partly complete. Therefore, access to this table (or even more specifically the refresh button) should be limited to system administrators if at all possible.

Maintenance

Scheduling integration tasks during off peak hours also has the disadvantage that your support staff for both BizTalk and Dynamics AX is much less available to diagnose errors. Support for AX integration is often a combination of business knowledge and BizTalk technical integration skill. We highly recommend at very minimum that your BizTalk support team has view access to the AIF **Queue manager**, **Exceptions**, and **Document history** forms and all the AIF configuration forms. View access to the **On-line user's** form can also be very helpful in confirming connectivity.

After an unscheduled outage of your Dynamics AX instance, you often need to restart the BizTalk host instance to reconnect to the AOS. Thus, we would recommend running your BizTalk application for Dynamics AX under a separate host instance from your other applications. The same holds true for any services leveraging the .NET Business Connector.

Summary

In this chapter, we discussed the advantages of using the BizTalk Dynamics AX 2009 adapter for our BizTalk integration applications. We went through the required configuration setup of the Application Integration Framework module in Dynamics AX 2009.

In our first example, we used the out-of-the-box AIF services

LedgerExchangeRatesService to create a currency exchange rate application that demonstrated the synchronous integration mode of the BizTalk adapter. In the second example, we created a generic solution to retrieve outbound message from Dynamics which demonstrated the set requirement for asynchronous integration including batch jobs.

Finally, we touched on using the **.NET Business Connector** for integration with Dynamics AX and discussed the implications to consider when writing custom applications without leveraging the AIF module. In the next chapter, we'll discuss the integration with Salesforce CRM. A popular CRM that runs in the "cloud".

Where to buy this book

You can buy Oracle Microsoft BizTalk 2010: Line of Business Systems Integration from the Packt Publishing website: <http://www.packtpub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book>.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our [shipping policy](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.



www.PacktPub.com

For More Information:

www.PacktPub.com/microsoft-biztalk-2010-line-of-business-systems-integration/book