webMethods OnRamp for CommerceOne MarketSite Adapter Installation and User's Guide

Version 3.5 SP1

July 2012

WEBMETHODS

This document applies to webMethods OnRamp for Commerce One MarketSite 3.5 SP1 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ADAPTER-POC-IUG-35 SP1-20210326

Table of Contents

About this Guide

This guide describes how to install, configure, and use webMethods OnRamp for Commerce One MarketSite Version 3.5 SP1. It provides an overview of how C1 OnRamp operates and explains common tasks you can perform using C1 OnRamp.

To use this guide effectively, you should be familiar with:

- Other software applications and protocols you can use with C1 OnRamp, including the Commerce One MarketSite application.
- The Commerce One RoundTrip protocol.
- The setup and operation of webMethods Integration Server.
- Have a general idea about how to perform basic tasks with Software AG Designer.
- webMethods Trading Networks and understand the concepts and procedures described in the various Trading Networks guides.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at https://empower.softwareag.com/.

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.aspx and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Concepts

Overview

webMethods OnRamp for Commerce One MarketSite is a package you install on webMethods Integration Server so you can send xCBL business documents to and receive xCBL business documents from Commerce One MarketSite applications. Commerce One MarketSites are portals that businesses use to exchange goods and services worldwide.

All xCBL documents you send to or receive from MarketSites are transported in envelopes. Envelopes can also contain any number of attachments. The format of the envelopes is Commerce One's proprietary MML format and is based on MIME. C1 OnRamp uses HTTP or HTTPS as its envelope transport mechanism.

With C1 OnRamp, you can do the following:

- "Define MarketSite Aliases" on page 10
- "Build Services to Receive and Send Documents" on page 10
- "Validate Documents" on page 12
- "Implement RoundTrip" on page 12
- "View Logs" on page 12

Define MarketSite Aliases

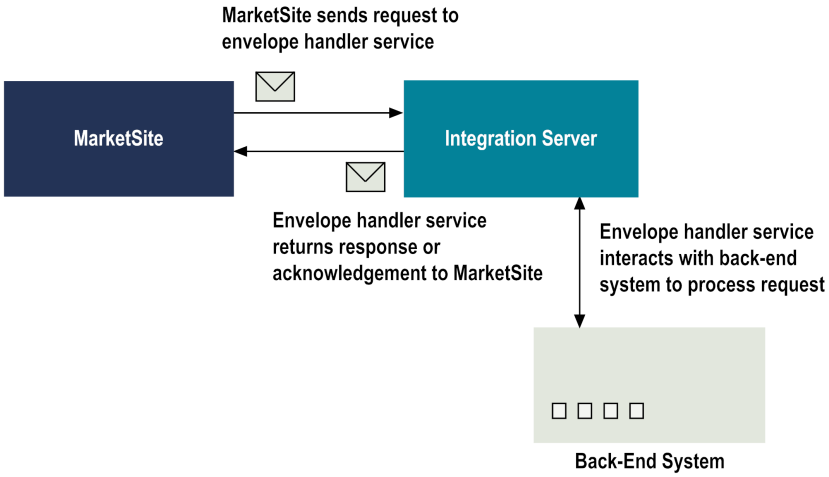
In C1 OnRamp, you define an *alias* for each MarketSite with which you want to exchange envelopes. After you define an alias, you can immediately test your synchronous and asynchronous connection to that alias by sending an envelope to MarketSite. You can later test your connection to a MarketSite alias whenever necessary.

Build Services to Receive and Send Documents

C1 OnRamp provides services you need to build the two types of services that enable you to exchange xCBL documents with MarketSite - *envelope handler services* and *send services*. The services run on webMethods Integration Server.

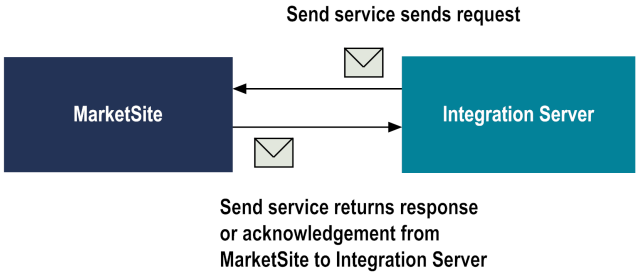
An *envelope handler service* does the following:

1. Receives incoming envelopes that contain xCBL request documents from MarketSite.
2. Interacts with your back-end systems to process each request.
3. For each request, returns a reply envelope to MarketSite that contains either an xCBL response document or an acknowledgement that means the request was received and a send service will send a response document later.



Send services do one of the following:

- Sends MarketSite an envelope containing an xCBL response document when an envelope handler service only sent an acknowledgement.
- Sends envelopes containing xCBL request documents and, optionally, attachments to MarketSite. For each request, the send service waits for an xCBL response document or acknowledgement from MarketSite and returns a reply envelope containing the document or acknowledgement to webMethods Integration Server.



You can use the C1 OnRamp Test Send/Receive utility to test your envelope handler and send services. Specifically, you can test whether you can receive an envelope from a MarketSite, whether you can send an envelope to a MarketSite, and whether a service you built operates as expected.

MarketSite requires *document destination identifiers* (DDIDs) to direct envelopes to their destinations. If your back-end systems cannot handle DDIDs, you can build your envelope handler services to convert the DDIDs in envelopes it receives from MarketSite to sender and recipient IDS. You can build your send services to convert the sender and recipient IDs it receives from your back-end system to DDIDs. The services do the conversion using DDID mappings you create.

C1 OnRamp includes a package named WmMarketConnectExample that contains sample envelope handler, send, and startup services.

Validate Documents

C1 OnRamp includes the Commerce One CXP parser and a package named WmxCBL that contains all SOX schemas for xCBL versions 3.5, 3.0, and 2.0 r3. You can use the parser and the schemas to validate the xCBL documents you send and receive.

C1 OnRamp offers three different methods for validating xCBL documents.

- In your C1 OnRamp configuration, you identify the SOX schemas you want to use for document validation (typically, the schemas in the WmxCBL package). You also specify your default validation settings - whether you want to validate all outgoing xCBL documents by default and whether you want to validate all incoming xCBL documents by default.
- In individual envelope handler and send services, you can set parameters or call a service that overrides the default validation settings in the C1 OnRamp configuration.
- During development, for debugging purposes, you can validate an xCBL document against a SOX schema you specify.

Implement RoundTrip

C1 OnRamp helps suppliers implement Commerce One's RoundTrip protocol by providing services that let you map buyers' shopping cart data to RoundTrip-compliant forms. (The RoundTrip protocol is identical to SAP's Open Catalog Interface (OCI) protocol.)

C1 OnRamp provides services you use to build RoundTrip response services. RoundTrip response services create RoundTrip responses from buyers' shopping carts and convert the responses into HTML or XML you can imbed in the HTML forms you display on the buyers' Web browsers. The forms specify a URL that points the buyer application and provide a Submit button. When a buyer clicks the Submit button, the buyer's Web browser submits the form to the specified URL.

View Logs

C1 OnRamp provides three logs, as follows:

- **C1 OnRamp Log.** The C1 OnRamp log contains information about C1 OnRamp processing. OnRamp creates one processing log per day and stores the logs in the *Integration Server_directory*\logs\WmMarketConnect directory. You control the amount of detail the C1 OnRamp log contains.
- **C1 OnRamp Envelope Log.** You can have C1 OnRamp store envelopes you send, receive, or both during a C1 OnRamp session by turning on envelope logging. With each envelope it stores in the C1 OnRamp envelope log, C1 OnRamp stores the associated reply envelope and any errors. You control the number of envelopes C1 OnRamp stores in the log; after the maximum is reached, C1 OnRamp discards the oldest envelopes to make room for new ones. Envelopes are not persisted between sessions.
- **C1 OnRamp Errors Log.** The C1 OnRamp errors log contains all errors that occur during C1 OnRamp processing. Errors are persisted across C1 OnRamp sessions; if you want to delete them, you must do so manually

2 Installing and Uninstalling OnRamp for Commerce One MarketSite

Overview

This chapter explains how to install, upgrade, and uninstall OnRamp for Commerce One MarketSite 3.5 SP1. The instructions use the Software AG Installer and the Software AG Uninstaller wizards. For complete information about the wizards or other installation methods, or to install other webMethods products, see the *Installing webMethods Products On Premises* for your release.

If you are installing C1 OnRamp on an AS/400 system, contact Software AG Global Support for installation and upgrade requirements and instructions.

Requirements

For a list of operating systems, Commerce One MarketSite products, and webMethods products supported by OnRamp for Commerce One MarketSite, see *webMethods Adapters System Requirements* .

C1 OnRamp has no hardware requirements beyond those of its host Integration Server. If you plan to store a large number of envelopes, however, you will need more storage space than Integration Server requires.

Installing OnRamp for Commerce One MarketSite 3.5 SP1

Before You Begin

1. If you are going to install webMethods OnRamp for Commerce One MarketSite on an already installed Integration Server, shut down the Integration Server.

2. Download the Software AG Installer from the Empower Product Support Web site at http://empower.softwareag.com.

Install C1 OnRamp 3.5 SP1

To install C1 OnRamp 3.5 SP1

1. Start Software AG Installer.

2. Choose the webMethods release that includes the Integration Server on which to install the adapter. For example, if you want to install the module on Integration Server 8.2, choose the 8.2 release.

3. Specify the installation directory as follows:

   - If you are installing webMethods OnRamp for Commerce One MarketSite on an existing Integration Server, specify the Software AG installation directory that contains the host Integration Server.

- If you are installing both the host Integration Server and the adapter, specify the installation directory to use.

4. In the product selection list, select **Adapters > webMethods CommerceOne OnRamp 3.5.1**.

   Note:
   The **WmMarketconnectExample 3.5 SP1** and **WmMarketconnectTN 3.5 SP1** packages are optional. If you do not want to install the **WmMarketconnectExample 3.5 SP1** and **WmMarketconnectTN 3.5 SP1** packages, unselect the packages. For more information about the **WmMarketconnectTN 3.5 SP1** package, see "Using C1 OnRamp with webMethods Trading Networks " on page 83.

5. Install any required webMethods components you have not installed, such as Integration Server and Trading Networks. For a list of products and their versions required by OnRamp for Commerce One MarketSite, see *webMethods Adapters System Requirements* .

6. To install documentation for OnRamp for Commerce One MarketSite, on the Documentation panel in Installer, select **Adapter Readmes and Documentation**. Alternatively, you can download the documentation at a later time from the Software AG Documentation Web page.

7. After the Installer completes the installation, close the Installer.

Complete the Installation

To install the Commerce One XDK

You must extract the jar files for the Commerce One XDK on a Windows system. After you have extracted the jar files, you can copy them to the system on which you installed C1 OnRamp.

1. Go to this URL:

   ```
   http://www.xcbl.org/xdk/distribution.html
   ```

2. Download the xdk101r1.zip file.

3. Navigate to the directory into which you downloaded the zip file and extract the jar files by running the xdk101r1.exe file.

4. Navigate to the directory into which you extracted the jar files and copy the file (ccs_dir.jar, ccs_event.jar, ccs_util.jar, sax.jar, xdk_dev.jar, and xt.jar) to the *Integration Server_directory* /packages/WmMarketConnect/code/jars directory.

5. Make sure the *Integration Server_directory* /packages/WmMarketConnect/code/jars directory contains these files:

   activation.jar

ccs_dir.jar

ccs_event.jar

ccs_util.jar

collections.jar

jndi.jar

mail.jar

sax.jar

xdk_dev.jar

xt.jar

In addition, make sure the directory contains a subdirectory named static.

6. If you do not need to modify B2B Server startup scripts, restart Integration Server.

Uninstalling OnRamp for Commerce One MarketSite 3.5 SP1

To uninstall C1 OnRamp 3.5 SP1

1. Shut down the Integration Server that hosts webMethods OnRamp for Commerce One MarketSite.

2. Start Software AG Uninstaller, selecting the webMethods installation directory that contains the host Integration Server. In the product selection list, select **Adapters > webMethods CommerceOne OnRamp 3.5.1** and any other products and items you want to uninstall.

3. Restart the Integration Server from which you uninstalled webMethods OnRamp for Commerce One MarketSite.

4. Uninstaller does not delete files that you have created or configuration files associated with webMethods OnRamp for Commerce One MarketSite 3.5 SP1, nor does it delete the directory structure that contains those files. If you do not want to save those files, go to the *Integration Server_directory* \packages directory and delete the C1 OnRamp package directories.

3 Configuring OnRamp for Commerce One MarketSite

Identify Service to Receive Envelopes from MarketSite

When you obtain a MarketSite account, you are asked for the destination to which you want MarketSite to send all envelopes.

The recommended destination is the receiveEnvelope service. The URL for the `receiveEnvelope` service is as follows, where alias is the alias for the MarketSite that is sending the envelopes:

https://*host*:*port*/invoke/pub.marketconnect.transport/receiveEnvelope?marketConnectAlias=*alias*

You can use a custom service as the destination instead of the `receiveEnvelope` service. The custom service must adhere to the envelope handler service specification (see "EnvelopeHandlerSpecification" on page 66).

Define and Test a MarketSite Alias

To define and test a MarketSite alias

1. Open Integration Server Administrator.
2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.
3. In the C1 OnRamp navigation area, click **Aliases**. The system displays the MarketSite Aliases page.
4. Click **Add Alias**. The system displays the Add HTTPS Alias page.
5. Complete the boxes on the page as follows:

Box	Value
Alias	Alias for the MarketSite with which you want to communicate.
MarketSite Authorizing Entity/TPID	Your trading partner ID (TPID) for this MarketSite.
MarketSite System Account ID	Your user ID for this MarketSite.
MarketSite System Account Password	Your password for this MarketSite.
MarketSite URL	Destination for all envelopes you send to this MarketSite.
MarketSite Timeout (seconds)	Maximum period of time Integration Server is to wait for a synchronous send to this MarketSite to complete before giving up and ending the connection.

6. Click **Submit Changes**. The system returns to the MarketSite Aliases page and lists the alias you just defined.

7. Test your synchronous connection to the alias by clicking ▶ in the **Test Sync**column for the alias. The system generates an envelope containing a Ping document and sends the envelope synchronously to the MarketSite destination you specified in the **MarketSite URL**box. If you quickly receive a reply envelope containing a Pong document from MarketSite, the connection is working. If you receive a reply envelope containing an error or if you receive error messages, the connection is not working.

8. Test your asynchronous connection to the alias by clicking ▶ in the **Test Async** column for the alias. The system generates an envelope containing a Ping document and sends the envelope asynchronously to the MarketSite destination you specified in the **MarketSite URL**box. If you quickly receive a reply envelope containing an acknowledgement and later receive an envelope containing a Pong document from MarketSite, the connection is working. If you receive a reply envelope containing an error, if you receive error messages, or if you do not receive an envelope containing a Pong document within 30 seconds, the connection is not working.

Create a DDID Mapping

To create a DDID mapping

1. In the C1 OnRamp navigation area, click **DDIDs**. The system displays the first DDIDs page.

2. To add a mapping for a trading partner application on your local system, click **Edit** in the cell at the intersection of the **Local Partner ID** column and the row for the appropriate MarketSite. The system displays the second DDIDs page.

3. Click **Add DDID**. The system displays the Add DDID page.

4. In the **Document Destination ID** box, type the DDID for the trading partner application.

5. In the **Local Partner ID** box, type the partner ID for the trading partner application.

6. In the **xCBL Version** list, click the version of the xCBL documents you will exchange with the trading partner application.

7. Click **Submit Changes**. The system returns to the second DDIDs page and lists the mapping you just defined.

8. Follow a similar procedure to add a mapping for a trading partner application on a remote system.

Specify Default Document Validation

To specify default document validation

1. In the C1 OnRamp navigation area, click **xCBL Validation**. The system displays the Validation page.

2. In the **Schema Path** box, type the path to the SOX schemas against which you want to validate documents. The schemas must be installed local to Integration Server. The default is **packages/WmxCBL/schema**, the path to the directory that contains all the SOX schemas for xCBL version 3.5, 3.0, and 2.0 r3, provided with C1 OnRamp.

3. In the **Validate Outgoing Documents** list, indicate whether you want to validate all xCBL documents you try to send to MarketSite. The service you use to create the envelope performs the validation; if the document is invalid, the service fails.

   Note:
   Validate all outgoing envelopes only during development, for debugging purposes. In a production environment, if you are sending a large number of envelopes, validating all outgoing envelopes will negatively affect performance.

4. In the **Validate Incoming Documents** list, indicate whether you want to validate all xCBL documents you receive from MarketSite. The "receiveEnvelope" on page 67 service performs the validation; if the document is invalid, the service fails and C1 OnRamp stores the envelope in the C1 OnRamp Errors Log. If you are logging incoming envelopes, C1 OnRamp also stores the envelope in the C1 OnRamp Envelope Log.

5. Click **Submit Changes**.

4 Document Exchange Processing

Receiving an xCBL Request Document from MarketSite

This section explains how C1 OnRamp receives and processes an envelope containing an xCBL request document from MarketSite.

1. MarketSite transmits an envelope containing a buyer's xCBL request document and, optionally, attachments to the C1 OnRamp receiveEnvelope service.
2. The receiveEnvelope service forwards the envelope to the appropriate envelope handler service.
3. The envelope handler service calls C1 OnRamp services that:
   a. Get the envelope header information.
   b. Get the xCBL request document's header information.
   c. Get the xCBL request document and, optionally, validate the document.
   d. Get any attachments from the envelope.

   C1 OnRamp automatically calls a C1 OnRamp service that looks up the DDIDs in the envelope header and returns the corresponding sender and receiver IDs.

   The envelope handler service interacts with your back-end system to process the retrieved xCBL request document and attachments.
4. The next step depends on the request mode of the incoming document and on whether errors occur during processing.
   - If MarketSite sent the envelope synchronously and no processing errors occur, your back-end system provides data to the appropriate webMethods adapter. The adapter receives the data and calls custom services you developed. The custom services map the data to an xCBL response document and return the document to the envelope handler service. The envelope handler service then calls C1 OnRamp services that:
     1. Optionally, validate the xCBL response document.
     2. If your back-end system does not handle DDIDs, look up the sender and recipient IDs provided by your back-end system and return the corresponding DDIDs.
     3. Create a reply envelope that contains the xCBL response document.
     4. Create any necessary attachments and add them to the envelope.

     The envelope handler service then returns the reply envelope to the `receiveEnvelope` service, which returns the reply envelope to MarketSite.
   - If MarketSite sent the envelope asynchronously and no processing errors occur, C1 OnRamp immediately returns a reply envelope containing a default acknowledgement to MarketSite. (If you want to return a custom acknowledgement instead, you can call a C1 OnRamp service that creates a reply envelope containing the custom acknowledgement and passes the reply envelope to the `receiveEnvelope` service to return to MarketSite.)

Later, the back-end system provides data to the appropriate webMethods adapter. The adapter receives the data and calls custom services you developed. The custom services map the data to an xCBL response document and pass the document to a C1 OnRamp send service. The send service calls C1 OnRamp services that:

1. Optionally, validate the xCBL response document.
2. If your back-end system does not handle DDIDs, look up the sender and recipient IDs provided by your back-end system and return the corresponding DDIDs.
3. Create an envelope that contains the xCBL response document.
4. Create any necessary attachments and add them to the envelope.
5. Send the envelope to MarketSite (sendEnvelope service).

- For either request mode, if errors occur during processing, C1 OnRamp immediately returns a reply envelope containing a default error to MarketSite. (If you want to return a custom error instead, you can call a C1 OnRamp service that creates a reply envelope containing the custom error and passes the reply envelope to the `receiveEnvelope` service to return to MarketSite.)

Sending an xCBL Request Document to MarketSite

This section explains how C1 OnRamp sends an xCBL request document to MarketSite.

1. Your back-end system or a scheduled service generates data and sends it to the Integration Server via the appropriate webMethods adapter.
2. The adapter receives the data and calls custom services you developed. The custom services map the data to an xCBL request document and pass the document to a C1 OnRamp send service.
3. The send service calls C1 OnRamp services that:
   a. Optionally, validate the xCBL document.
   b. If your back-end system does not handle DDIDs, look up the sender and recipient IDs provided by your back-end system and return the corresponding DDIDs.
   c. Create an envelope containing the xCBL request document.
   d. Create any necessary attachments and add them to the envelope.
   e. Send the envelope to MarketSite (sendEnvelope service).
4. MarketSite returns a reply envelope to the `sendEnvelope` service. The contents of the reply envelope depend on the request mode of the envelope you sent and on whether errors occur during processing, as follows:
   - If you sent the envelope synchrononously and no processing errors occur, MarketSite's reply envelope contains an xCBL response document and any attachments.

- If you sent the envelope asynchronously and no processing errors occur, MarketSite's reply envelope contains an acknowledgement. (MarketSite sends an envelope containing an xCBL response document and any attachments to the C1 OnRamp receiveEnvelope service later.)
- For either request mode, if errors occur during processing, the envelope that MarketSite returns to the sendEnvelope service contains an error instead of an acknowledgement or xCBL response document.

5 Setting Up Secure Communication with MarketSite

Communicate as a Server

When a MarketSite client tries to connect to Integration Server via SSL, Integration Server submits its SSL certificate chain to the client. For the MarketSite client to establish the connection, the certificate chain Integration Server submits must be complete.

If Integration Server's certificate was signed by a root CA, the certificate chain Integration Server submits is complete. If Integration Server's certificate was signed by an intermediary CA, however, the certificate chain Integration Server submits is not complete. In such cases, you must modify your Integration Server configuration.

Determine whether you need to modify the Integration Server configuration by finding out whether an intermediary CA signed Integration Server's certificate. You can ask your CA or you can consult the documentation you received with your certificate to determine whether your certificate chain includes the signature of an intermediary CA. If you are using a Windows system, you can see your certificate chain by double-clicking your certificate file to open the Certificate interface, then clicking the Certification Path tab.

This section describes how to modify your Integration Server configuration to have Integration Server submit a complete certificate chain. The section also describes additional changes you must make to your Integration Server configuration if the MarketSite client you are communicating is at a Commerce One.net marketplace.

Note:
This section assumes you have already configured Integration Server for SSL communication using instructions in the *webMethods Integration Server Administrator's Guide* for your release. If you edit your HTTPS configuration from Integration Server Administrator after making the changes in this section, the changes will be lost and you will have to make them again.

Configure SSL Communication with Any MarketSite Client

To configure Integration Server to communicate with a MarketSite client, you create an HTTPS listener, obtain a certificate file for each entity in the certificate chain, and instruct Integration Server to send the complete certificate chain to clients that connect to Integration Server via SSL.

To configure SSL communication with any MarketSite client

1. Open Integration Server Administrator and create a listener.

   - If you are communicating with a MarketSite version earlier than 4.0, create a webMethods/HTTPSForMarketConnect listener.
   - If you are communicating with a MarketSite version 4.0 or later, create a webMethods/HTTPS listener.

     For instructions on creating listeners, see the *webMethods Integration Server Administrator's Guide* for your release. Make note of the package with which you associated the listener.

2. For each entity in the Integration Server certificate chain, obtain a certificate file in the DER format used by Integration Server and place the files in a directory that is accessible to Integration Server (for example, the *Integration Server_directory* \config directory).

   If you are not using a Windows system, ask your CA how to obtain the files. If you are using a Windows 2000 system, you can use a wizard to obtain the files, as follows:

   a. Double-click your certificate file. The system opens the Certificate interface. Click the **Certification Path** tab to display Integration Server's certificate chain.

   b. Double-click a CA entity in the chain. The system opens another Certificate interface. Click the **Details** tab.

   c. Click **Copy to File** to open the wizard. Export the file to **DER encoded binary X.509**.

   d. Repeat these steps for each CA entity in the chain.

3. Shut down Integration Server.

4. Navigate to the *Integration Server_directory**package*\config directory, where *package* is the package with which you associated the HTTPSForMarketConnect listener. Open the listener.cnf file in a text editor. The file is an XML file and contains information about each listener that is associated with the package. Instruct Integration Server to send its complete certificate chain to clients that connect to Integration Server via SSL by editing the file as follows:

   a. Locate the record for the HTTPSForMarketConnect listener.

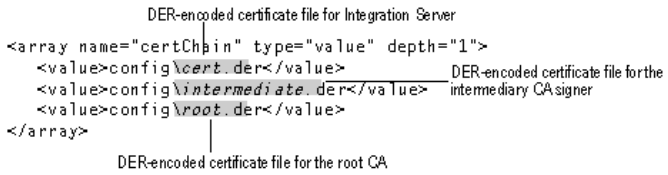
   b. Delete these lines:

```
<value name="signedCert"></value>
<value name="caCert"></value>
```

   c. Locate these lines:

```
<array name="certChain" type="value" depth="1">
   <value></value>
   <value></value>
</array>
```

   d. Edit the lines to list the entities in the certificate chain as shown below, where *cert*.der is the DER-encoded certificate file for Integration Server, *intermediate*.der is the DER-encoded certificate file for the intermediary CA signer, and *root*.der is the DER-encoded certificate file for the root CA:



5. Restart Integration Server.

Configure SSL Communication with a Commerce One.net MarketSite Client

To configure Integration Server to communicate with a MarketSite client at a Commerce One.net marketplace, you obtain a special certificate file from Commerce One, convert the file to the format used by Integration Server, and associate the converted certificate with an Integration Server user that can call the C1 OnRamp receiveEnvelope service.

To configure SSL communication with a Commerce One.net MarketSite client

1. Open Integration Server Administrator and set the webMethods/HTTPSForMarketConnect listener to request certificates from clients trying to connect to Integration Server via SSL (For instructions, see the *webMethods Integration Server Administrator's Guide* for your release.

2. Obtain a certificate file named marketsite.p7m from Commerce One. The file contains a MarketSite certificate chain in PKCS #7 format. Install and configure the file as instructed by Commerce One.

3. Convert the certificate file from PKCS #7 file format to the DER format used by Integration Server as follows:

   - If you are using a Windows 2000 system, you can use a wizard to export the file to DER format. Rename the certificate file marketsite.p7c, then double-click the file. The system opens the Certificate interface. Click the **Details** tab, then click **Copy to File** to open the wizard. Export the file to **DER encoded binary X.509**.
   - If you are using a Windows NT system, you can use the Certificate Manager application to convert the file.
   - If you are using a non-Windows system, use a commercial certificate manipulation tool to convert the file.

4. Associate the certificate you just converted with an Integration Server user that can call the C1 OnRamp "receiveEnvelope" on page 67 service. To complete the following steps use the instructions in the *webMethods Integration Server Administrator's Guide* for your release.

   a. In Integration Server Administrator, add a user that represents the MarketSite to the ACL that is authorized to call the "receiveEnvelope" on page 67 service.

b. Import the certificate and associate it with the user.

Communicate as a Client

MarketSite only accepts SSL certificates from clients when MarketSite requests them. You can choose to not submit any certificates, even when MarketSite requests them, or you can choose to submit certificates at MarketSite's request.

Note:
This section assumes you have already configured Integration Server for SSL communication using instructions in the *webMethods Integration Server Administrator's Guide* for your release. If you edit your HTTPS configuration from Integration Server Administrator after making the changes in this section, the changes will be lost and you will have to make them again.

Do Not Send Certificates to MarketSite

If you do not want to send certificates to MarketSite, even when requested, you can either:

- Configure Integration Server to submit certificates in all outbound communications by default and override the default for MarketSite communications.
- Configure Integration Server to not submit certificates in any outbound communication by default and override the default for non-MarketSite communications.

Submit Certificates by Default and Override for MarketSite Communications

To configure Integration Server so that it submits certificates

1. Specify Integration Server's global outbound certificate settings. (For instructions, see the *webMethods Integration Server Administrator's Guide* for your release.)

2. If an intermediary CA signed Integration Server's certificate, make sure the Integration Server certification path includes the complete certificate chain (see "Viewing C1 OnRamp Logs" on page 43 for more information).

3. Contact Software AG Global Support for the rest of the procedure.

Do Not Submit Certificates by Default and Override for Non-MarketSite Communications

To configure Integration Server so that it does not submit certificates

To complete the steps below, use the instructions in the *webMethods Integration Server Administrator's Guide* for your release.

1. Remove Integration Server's global outbound certificate settings.

2. In any non-MarketSite communication that Integration Server initiates for which you want to submit certificates, call the Integration Server `pub.security:setKeyAndChain` service and pass the service the certificates to use. (For information on the `pub.security:clearKeyAndChain` service, see the *webMethods Integration Server Built-In Services Reference* for your release.)

3. Open Integration Server Administrator and, for each non-MarketSite HTTPS listener, specify the certificates to use.

Send Certificates at MarketSite's Request

If you want to send certificates to MarketSite when MarketSite requests them, you can either:

- Configure Integration Server to submit certificates in all outbound communications by default.
- Configure Integration Server to not submit certificates in any outbound communication by default and override the default for MarketSite communications.

Submit Certificates by Default

To configure Integration Server so that it does not submit certificates

- Specify Integration Server's global outbound certificate settings. (For instructions, see the *webMethods Integration Server Administrator's Guide* for your release.) If an intermediary CA signed Integration Server's certificate, make sure the Integration Server certification path includes the complete certificate chain (see "Viewing C1 OnRamp Logs" on page 43 for instructions).

Do Not Submit Certificates by Default and Override for MarketSite Communications

To configure Integration Server so that it does not submit certificates

1. Remove Integration Server's global outbound certificate settings. (For instructions, see the *webMethods Integration Server Administrator's Guide* for your release.)

2. Make sure you have the DER-encoded file for every entity in Integration Server's certificate chain. If you do not, obtain them (see "Configure SSL Communication with Any MarketSite Client" on page 26 for instructions).

3. In any C1 OnRamp send service you create, call the Integration Server service pub.security:setKeyAndChain before you call the C1 OnRamp "sendEnvelope" on page 68 service. Pass to the service the Integration Server's complete certificate chain. (For information on the `pub.security:setKeyAndChain` service, see the *webMethods Integration Server Built-In Services Reference* for your release.)

6 Receiving and Sending Envelopes

Build an Envelope Handler Service

The WmMarketConnectExample package contains sample envelope handler services.

To build an envelope handler service

1. Open Designer and create a new package. Declare that the package is dependent on the Integration Server package WmMarketConnect. For more information, see the *webMethods Service Development Help* for your release.

2. In the new package, create a service that represents an envelope handler service.

3. If the handler service is to process both incoming envelopes sent synchronously and incoming envelopes sent asynchronously, but you want to process the envelopes differently based on their request mode, have the handler service call the "getRequestMode" on page 63 service. Then use a branching flow statement to set up separate logic for processing synchronous and asynchronous requests.

4. In the handler service, call services to get information from each incoming envelope, as follows:

   a. Call the "getEnvelopeHeader" on page 62 service to get the envelope header information.

   b. Call the "getDocumentHeader" on page 62 service to get the xCBL request document's header information.

   c. Call a service such as "getDocumentAsNode" on page 60 to get the xCBL request document.

   d. In your C1 OnRamp configuration, you specified whether to validate all incoming xCBL documents. If you want to override that default setting, specify the `validate` parameter on the getDocument service you call.

   e. Call a service such as "getAttachmentList" on page 59 to get attachments, if any.

   You can also have the handler service call custom services to process the document and attachments. (For instructions on writing custom services, see the *webMethods Service Development Help* for your release.)

5. Handle the reply to each incoming envelope. This step varies based on the incoming envelope's request mode.

   - If the incoming envelope's request mode is synchronous, your custom services return an xCBL response document to the handler service. In the handler service, call services to create a reply envelope to return to MarketSite, as follows:

     1. In your C1 OnRamp configuration, you specified whether to validate all outgoing xCBL documents. If you want to override that default setting, call the parseString service.

2. If your back-end system cannot handle DDIDs, call the getDDID service to get the DDIDs that correspond to the sender ID and recipient IDs provided by your back-end system. You must call the service twice, once to get the DDID for the sender ID and once to get the DDID for the recipient ID.
3. Call a service such as the createEnvelopeFromRecord service to create a reply envelope that contains the xCBL response document).
4. In the header of the reply envelope, use the correlationId from the header of the incoming envelope. If you want to specify non-standard fields in the reply envelope header, insert the name/value pairs in the customProperties envelope header record.
5. Call a service such as the createBinaryAttachment service to create any necessary attachments. If you want to create an attachment from a string, use the Integration Server service pub.string:bytesToString.
6. Call the addAttachmentToEnvelope service to add the attachments to the envelope.
7. Return the reply envelope to the receiveEnvelope service to return to MarketSite.

- If the incoming envelope's request mode is asynchronous, the handler service typically does not have to create a reply envelope. By default, when no errors occur during processing, C1 OnRamp returns a reply envelope to MarketSite that contains a default acknowledgement. If you want to create a custom acknowledgement, call the createAcknowledgement service; this service creates a reply envelope containing your custom acknowledgement and returns the reply envelope to the receiveEnvelope service to return to MarketSite. In the header of the reply envelope, use the correlationId from the header of the incoming envelope.

For either request mode, when errors occur during processing, C1 OnRamp by default returns a reply envelope to MarketSite that contains a default error. If you want to create a custom error, call the createError service; this service creates a reply envelope containing your custom error and returns the reply envelope to the receiveEnvelope service to return to MarketSite. In the header of the reply envelope, use the correlationId from the header of the incoming envelope.

You can also have the handler service call custom services to create reply envelopes. (For instructions on writing custom services, see the *webMethods Service Development Help* for your release.)

Build a Send Service

The WmMarketConnectExample package contains sample send services.

To build a send service

1. Open Designer.
2. Create a service that represents the send service in either of two locations, depending on the purpose of the send service:

- If the send service is to create and send envelopes containing xCBL response documents to MarketSite in reply to asynchronous requests, create the service in the package containing the envelope handler that processes the asynchronous requests.
- If the send service is to create and send envelopes containing xCBL request documents to MarketSite, create a new package, declare that the package is dependent on the Integration Server package WmMarketConnect, and create the send service in that package.

3. In the send service, call services that create and send envelopes, as follows:

   a. In your C1 OnRamp configuration, you specified whether to validate all outgoing xCBL documents. If you want to override that default setting, call the "parseString" on page 64 service.

   b. If your back-end system cannot handle DDIDs, call the "getDDID" on page 58 service to get the DDIDs that correspond to the sender ID and recipient IDs provided by your back-end system. You must call the service twice, once to get the DDID for the sender ID and once to get the DDID for the recipient ID.

   c. Call a service such as the "createEnvelopeFromRecord" on page 56 service to create an envelope that contains the xCBL document.

   d. If you want to specify non-standard fields in the envelope header, insert the name/value pairs in the customProperties envelope header record.

   e. If you are creating a reply envelope, use the correlationId from the header of the incoming envelope in the header of the reply envelope.

   f. Call a service such as the "createBinaryAttachment" on page 51 service to create any necessary attachments. If you want to create an attachment from a string, use the Integration Server service pub.string:bytesToString.

   g. Call the "addAttachmentToEnvelope" on page 56 service to add the attachments to the envelope.

   h. If necessary, call the pub.security:clearKeyAndChain service or the pub.security:setKeyAndChain service (see "Communicate as a Client" on page 29 for more information).

   i. Call the "sendEnvelope" on page 68 service to send the envelope to MarketSite.

   You can also have the send service call custom services to create envelopes. (For instructions on writing custom services, see the *webMethods Service Development Help* for your release.)

Build a Startup Service

When the C1 OnRamp receiveEnvelope service receives an envelope from MarketSite, the service uses a lookup table to route the envelope to the proper registered envelope handler service. You

must create a startup service that runs each time you start Integration Server to register the envelope handler services you want to use in the lookup table.

The WmMarketConnectExample package contains a sample startup service.

To build a startup service

1. Open Designer.
2. In any C1 OnRamp package, create a startup service.
3. In the startup service, call the "setDocumentHandler" on page 68 service once for each handler service you want to register in the envelope handler service lookup table.

7 Implementing RoundTrip

Build a RoundTrip Response Service

Important: This section assumes you understand the RoundTrip protocol and have read Commerce One's RoundTrip documentation.

To build a roundtrip response service

1. Open Designer and create a new package. Declare that the package is dependent on the Integration Server package WmMarketConnect. (For more information, see the *webMethods Service Development Help* for your release.)

2. In the new package, create a service that represents the RoundTrip response service.

3. In the RoundTrip response service, call services that convert a buyer's shopping cart into a RoundTrip response, as follows:

   a. Call the "createRoundTripResponse" on page 73 service to create an empty RoundTrip response.

   b. Call the "addCatalogToRoundTripResponse" on page 72 service to add a catalog to the RoundTrip response.

   c. Call the "addProductToRoundTripCatalog" on page 73 service to add products to the catalog in the RoundTrip response. Call the service once for each product in the buyer's shopping cart.

4. Call a service that converts the RoundTrip response into an HTML string, an XML string, or a record you can convert into an HTML string.

   Note:
   If you want to convert the RoundTrip response into XML, first make sure the versions of BuySite that your buyers use support that vocabulary.

   - To convert to an HTML string, call the getRoundTripResponseAsHTML service.
   - To convert to an XML string, call the getRoundTripResponseAsXML service. See Commerce One documentation for detail information on supported XML tags.
   - If you want to convert to an HTML string but you would like to customize the HTML's appearance, call the getRoundTripResponseAsRecord service to convert the RoundTrip response into a record. Build a template that converts the record into an HTML string using instructions in Dynamic Server Pages and Output Templates Developer's Guide , then use Integration Server services in the pub.report folder of the WmPublic package to apply the templates. (For instructions, see the *webMethods Service Development Help* for your release.)

     You can now imbed the HTML or XML string in the HTML forms you display on buyers' Web browsers. See Commerce One's RoundTrip documentation for instructions.

8 Testing Connections and Services and Validating Documents

Test a Connection

To test a connection

1. Open Integration Server Administrator.
2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.
3. In the **Test** area, click **Test Connection**. The system displays the Test Connection page.
4. Test your synchronous connection to an alias by clicking ▶ in the **Test Sync**column for the alias. The system generates an envelope containing a Ping document and sends the envelope synchronously to the MarketSite destination you specified in the **MarketSite URL**box. If you receive a reply envelope containing a Pong document from MarketSite, the connection is working. If you receive a reply envelope containing an error or if you receive error messages, the connection is not working.
5. Test your asynchronous connection to the alias by clicking ▶ in the **Test Async** column for the alias. The system generates an envelope containing a Ping document and sends the envelope asynchronously to the MarketSite destination you specified in the **MarketSite URL**box.

   The **Asynchronous Test Connection Results** area lists the envelopes in the test; click next to an envelope to see its contents. If you received a reply envelope containing an acknowledgement and later received an envelope containing a Pong document from MarketSite, the connection is working. If you received a reply envelope containing an error, the connection is not working. Click the link in the **Status** column to view the errors that occurred.

Test a Service that Sends, Receives, or Processes Envelopes

To test a service

1. Open Integration Server Administrator.
2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.
3. In the **Test** area, click **Test Send/Receive**. The system displays the Test Send/Receive page.
4. In the **Service** area, indicate the type of test you want to perform.
   - If you want to test receiving an envelope using the receiveEnvelope service and processing the envelope, click **Receive Envelope**. (You will actually receive the envelope from your Integration Server.)

- If you want to test sending an envelope to a MarketSite using the sendEnvelope service, click **Send Envelope**.
- If you want to test a service you built (for example, an envelope handler service), click **Invoke service specified below** and type the path to the service in the box below.

5. Complete the envelope header boxes as follows:

Box	Value
sender	MarketSite TPID of the trading partner you want to be the envelope sender.
receiver	MarketSite TPID of the trading partner you want to be the envelope receiver.
msgType	msgType of the xCBL document you are going to add to the envelope to send to MarketSite.
requestMode	Request mode you want to use to send the envelope.
MarketSite	Alias for the MarketSite to which you want to send the envelope.

6. In the **Document** box, paste the xCBL document you want to add to the envelope.

7. Click **Submit**. If the test succeeds, the system displays the contents of your pipeline in HTML. If the test fails, the system displays the errors that occurred.

Validate an xCBL Document

To validate an xCBL document

1. Open Integration Server Administrator.

2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.

3. In the **Test** area, click **Validate Document**. The system displays the Validate Document page.

4. In the **Schema Path** box, type the path to the SOX schema against which you want to validate the document. The default is the schema path specified in the C1 OnRamp configuration, which by default is packages/WmxCBL/schema (see "Specify Default Document Validation" on page 20).

5. In the **Document** box, paste the document you want to validate.

6. Click **Submit**. If the document is valid, the system displays a message to that effect. If the document is invalid, the system displays the errors from the CXP parser.

9 Viewing C1 OnRamp Logs

View the C1 OnRamp Log

To view the C1 OnRamp log

1. Open Integration Server Administrator.

2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.

3. Click **view** next to the **C1 OnRamp Log** label. The system displays the log, in XML format.

4. If you want to increase or decrease the amount of detail the log is to contain from this point, follow these steps:

   a. Return to the C1 OnRamp Home page.

   b. In the **Logs** area, click **change** next to the **C1 OnRamp Log Level** label. The system displays the C1 OnRamp Log Level page.

   c. In the **C1 OnRamp Log Level** list, click the level of detail you want the log to contain. A typical level for production systems is 4 or 5; at this level, C1 OnRamp logs warnings in addition to the information logged at the lower levels. At level 7, C1 OnRamp logs the MML headers of every envelope sent or received, while at level 9, C1 OnRamp logs the entire contents of every envelope sent or received. For a description of each logging level, see the *webMethods Integration Server Administrator's Guide* for your release.

   d. Click **Submit Changes**. The system returns to the C1 OnRamp Home page and displays the new log level.

Start Envelope Logging and View the C1 OnRamp Envelope Log

To start envelope logging and view the log

For performance reasons, C1 OnRamp does not log envelopes by default. You must turn envelope logging on.

1. Open Integration Server Administrator.

2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.

3. Click **view** next to the **C1 OnRamp Envelope Log** label. The system displays the Envelope Logging page.

4. Click the number next to one of the **Maximum Number of Envelopes Logged** labels. The system displays the Envelope Logging Settings page.

5. By default, the maximum number of sent envelopes to log is set to 0 (no logging). If you want C1 OnRamp to log sent envelopes, type a number greater than 0 in the **Maximum Number of Sent Envelopes Logged** box. If you want C1 OnRamp to store all sent envelopes, type `None`.

   By default, the maximum number of received envelopes to log is set to 0 (no logging). If you want C1 OnRamp to log received envelopes, type a number greater than 0 in the **Maximum Number of Received Envelopes Logged** box. If you want C1 OnRamp to store all received envelopes, type `None`.

6. Click **Submit Changes**. The system returns to the Envelope Logging page and displays the new maximums.

7. If you want to see the contents of an envelope, click the envelope's message ID. Scroll down to see the associated reply envelope and errors, if any (envelopes with associated errors are shown in red). The envelope format is Commerce One's proprietary format, MML.

View the C1 OnRamp Errors Log

To view the C1 OnRamp Errors log

1. Open Integration Server Administrator.

2. In the navigation area, under the **Adapters** menu, click **C1 OnRamp**. The system displays the C1 OnRamp Home page.

3. Click **view** next to the **C1 OnRamp Errors** label. The system displays the log.

4. If you want to see detailed information about an error, click the error message. If you want to see the information in HTML format, click **alternate view** for that message.

A Public Records, Specifications, and Services

Service Summary

The table below briefly describes the public specifications and services in each C1 OnRamp package. The sections that follow describe each C1 OnRamp record, specification, and service in detail.

Service	Function
pub.marketconnect.attachment	
createAttachment	Creates an attachment from a binary or document object.
createBinaryAttachment	Creates an attachment from a binary object.
createDocumentAttachment	Creates an attachment from a document object.
getAttachmentAsBytes	Gets an attachment from an envelope and returns it as a byte array.
getAttachmentId	Gets the identity attribute for an attachment.
getAttachmentType	Gets the type attribute for an attachment.
pub.marketconnect.cbl	
dateToString	Converts a java.util.Date object into a string that contains a date in a format that is compliant with xCBL documents.
getXCBLVersion	Gets the version of an xCBL document.
stringToDate	Converts a string that contains a date in a format that is compliant with xCBL documents into a java.util.Date object.
pub.marketconnect.envelope	
addAttachmentToEnvelope	Adds an attachment to an envelope.
createEnvelopeFromRecord	Creates an envelope containing an xCBL document from a record that represents a complete xCBL document.
createEnvelopeFromString	Creates an envelope containing an xCBL document from a string that contains a complete xCBL document.
getAttachmentList	Gets all attachments from an envelope.
getAttachmentName	Gets the name of an envelope attachment.
getDDID	Looks up the sender or recipient ID in the appropriate DDID mapping and returns the corresponding DDID.
getDDIDMapping	Looks up a DDID in the appropriate DDID mapping and returns a record containing the corresponding sender or recipient ID.

Service	Function
getDocumentAsNode	Gets the xCBL document from an envelope and returns a webMethods document known as a node.
getDocumentAsRecord	Gets the xCBL document from an envelope and returns a record in the form of a webMethods object called a boundNode.
getDocumentAsString	Gets the xCBL document from an envelope and returns a string.
getDocumentHeader	Gets header information for the xCBL document that is contained in an envelope and returns the information as a record.
getEnvelopeHeader	Gets header information for an envelope and returns the information as a record.
getNamedAttachment	Gets a specific attachment from an envelope.
getRequestMode	Gets the request mode of an envelope.
pub.marketconnect.parse	
parseString	Validates or does not validate an xCBL document.
pub.marketconnect.reply	
createAcknowledgement	Creates a reply envelope that contains a custom acknowledgement for an asynchronous request and passes the reply envelope to the receiveEnvelope service to return to MarketSite.
createError	Creates a reply envelope that contains a custom error for a request and passes the reply envelope to the receiveEnvelope service to return to MarketSite.
pub.marketconnect.transport	
EnvelopeHandlerSpecification	Specification for envelope handler services.
receiveEnvelope	Receives incoming envelopes from MarketSite, gets the document type from each envelope, and forwards the envelope to the appropriate envelope handler service.
sendEnvelope	Sends an envelope to MarketSite.
setDocumentHandler	Registers an envelope handler service and the document type it handles in the envelope handler service lookup table.
setHandler	Registers your default envelope handler service with C1 OnRamp.

Service	Function
pub.marketconnect.roundtrip.response	
addCatalogToRoundTripResponse	Adds a catalog to a RoundTrip response.
addProductToRoundTripCatalog	Adds a product to a catalog in a RoundTrip response.
createRoundTripResponse	Creates an empty RoundTrip response to which you can add catalogs and products.
getRoundTripResponseAsHTML	Converts a RoundTrip response into an HTML string.
getRoundTripResponseAsRecord	Converts a RoundTrip response into a record.
getRoundTripResponseAsXML	Converts a RoundTrip response into an XML string.
wm.PartnerMgr.gateway.transport.B2B	
InboundProcess	Submits a document to the Partner Manager.
OutboundProcess	Invokes a service on a local or remote webMethods Integration Server.
wm.PartnerMgr.gateway.transport.EmailTransport	
OutboundProcess	Sends a document as an e-mail attachment to specified recipients
wm.PartnerMgr.gateway.transport.FTPTransport	
OutboundProcess	FTPs a document to a specified host.
wm.PartnerMgr.xtn.Sweeper	
sweepTRX	Purges transactions in the message store when invoked.
wm.marketconnect.xcbl	
xCBL4_GenerateRecords	Creates DocTypes for xCBL 4.0 document.
xCBL4_RegisterTNDocTypes	Registers the DocTypes in for xCBL 4.0 document in Trading Networks.
xCBL4_DeleteTNDocType	Deletes the registered DocTypes for xCBL 4.0 document from Trading Networks.

pub.marketconnect.attachment

createAttachment

Creates an attachment from a binary object such as a PDF file or a file that is not plain text or from a document object such as an XML string.

Note:
This service has been superseded by the createBinaryAttachment and createDocumentAttachment services.

Package

WmMarketConnect

Input Parameters

bytes	**Object** Object from which to create the attachment, as a byte array.
encoding (optional)	**String** The encoding attribute for the attachment. If you do not specify this parameter, the service does not include the encoding attribute in the attachment.
identity (optional)	**String** The identity attribute for the attachment. If you do not specify this parameter, the service does not include the identity attribute in the attachment.
type (optional)	**String** The MIME type attribute for the attachment. If you do not specify this parameter, or if you specify the type `application/xml` or a type that starts with `text/`, the service creates a text (document) attachment. If you specify any other type, the service creates a binary attachment.

Output Parameters

attachment	Resulting attachment.

createBinaryAttachment

Creates an attachment from a binary object such as a PDF file or a file that is not plain text.

Package

WmMarketConnect

Input Parameters

bytes	**Object** Object from which to create the attachment, as a byte array.
type	**String** The MIME type attribute for the attachment.

Output Parameters

attachment	Resulting attachment.

createDocumentAttachment

Creates an attachment from a document object such as an XML string.

Package

WmMarketConnect

Input Parameters

bytes	**Object** Object from which to create the attachment, as a byte array.
encoding (optional)	**String** The encoding attribute for the attachment.
identity (optional)	**String** The identity attribute for the attachment.
soxType (optional)	**String** The SOX type attribute for the attachment.
documentType (optional)	**String** The document type attribute for the attachment.
contentType (optional)	**String** The MIME type attribute for the attachment. Values are `application/xml` or a type that starts with `text/`.
type (optional)	This parameter is identical to and has been superseded by the *contentType* parameter.

Output Parameters

attachment	Resulting attachment.

getAttachmentAsBytes

Gets an attachment from an envelope and returns it as a byte array.

Package

WmMarketConnect

Input Parameters

attachment	Attachment to get.

Output Parameters

bytes	Resulting byte array.

getAttachmentId

Gets the identity attribute for an attachment.

Package

WmMarketConnect

Input Parameters

attachment	Attachment whose identity attribute to get.

Output Parameters

attachmentId	**String** The identity attribute.

getAttachmentType

Gets the MIME type attribute for an attachment.

Package

WmMarketConnect

Input Parameters

attachment	Attachment whose type attribute to get.

Output Parameters

attachmentType	**String** The type attribute.

pub.marketconnect.cbl

dateToString

Converts a java.util.Date object into a string that contains a date in a format that is compliant with xCBL documents.

Package

WmMarketConnect

Input Parameters

date	**java.util.Date** Object to convert.
format (optional)	**String** The format of the date in the resulting string. ■ If you want the string to contain the year, month, and day, specify the format `dateOnly`. ■ If you want the string to contain the year, month, day, hour, minute, second, and time zone offset, do not specify this parameter or specify the format `dateAndTime`.

Output Parameters

dateString	Resulting string.

getXCBLVersion

Gets the version of an xCBL document.

Package

WmMarketConnect

Input Parameters

xmldata	xCBL document whose version to get.

Output Parameters

version	Version of the xCBL document.

stringToDate

Converts a string that contains a date in a format that is compliant with xCBL documents into a java.util.Date object.

Package

WmMarketConnect

Input Parameters

dateString	**String** The date to convert.

Output Parameters

date	Resulting java.util.Date object.

pub.marketconnect.envelope

envelopeHeader

The envelopeHeader record contains the fields below.

Field	Value
senderId	DDID of the envelope sender.
recipientId	DDID of the envelope receiver.
correlationId (optional)	Unique identifier used to associate the envelope with related envelopes (for example, an envelope containing a PurchaseOrder document and an envelope containing a PurchaseOrderResponse document). The correlationID is assigned by the sender of the first envelope in the series. If you do not specify this parameter, no correlation ID is used.

Field	Value
documentType	Type of the xCBL document contained in the envelope.
requestMode	Indicates whether the envelope was sent synchronously or asynchronously. Values are `sync`, `async`, `peer-peer`, and `oneway`.
messageId (optional)	Unique identifier for the envelope. If you do not specify this parameter, no message ID is used.

Package

WmMarketConnect

addAttachmentToEnvelope

Adds an attachment to an envelope.

Package

WmMarketConnect

Input Parameters

envelope	Envelope to which to add the attachment.
attachment	Attachment to add.
attachmentUri (optional)	**String** The name of the attachment to add.

Output Parameters

None.

createEnvelopeFromRecord

Creates an envelope containing an xCBL document from a record that represents a complete xCBL document.

Package

WmMarketConnect

Input Parameters

envelopeHeader	**Record** The header attributes of the envelope to create.
boundNode	**Record** The top-level tag of the xCBL document. For example, if the document type is PurchaseOrder, specify a record named PurchaseOrder.
documentHeader (optional)	**String** The xCBL document header. If you do not specify this parameter, the services uses the default xCBL document header `<soxtype urn:x-commerceone: document:com:commerceone:CBL:CBL.sox>`.
recordName (optional)	**String** The record type of the record specified by the boundNode parameter. If you do not specify this parameter, the service uses the xCBL record prefix and document type.

Output Parameters

envelope	Resulting envelope.

createEnvelopeFromString

Creates an envelope containing an xCBL document from a string that contains a complete xCBL document.

Package

WmMarketConnect

Input Parameters

envelopeHeader	**Record** The header attributes of the envelope to create.
documentString	**String** The entire xCBL document, including SOX processing instructions.

Output Parameters

envelope	Resulting envelope.

getAttachmentName

Gets the name of an envelope attachment.

Package

WmMarketConnect

Input Parameters

attachment	Attachment whose type attribute to get.
envelope	Envelope that contains the attachment.

Output Parameters

attachmentName	**String** The attachment name.

getDDID

Looks up the sender or recipient ID in the appropriate DDID mapping and returns the corresponding DDID.

Package

WmMarketConnect

Input Parameters

marketConnectAlias	**String** The alias of the MarketSite to which you are sending the envelope. This string is provided by the Software AG Partner Manager.
type	**String** The type of ID to look up. Values are `local` (sender ID) and `remote` (recipient ID).
partnerID	**String** The sender ID or recipient ID. This parameter works with the *type* parameter.

Output Parameters

ddID	Corresponding DDID.

getDDIDMapping

Looks up a sender or recipient DDID in the appropriate DDID mapping and returns a record containing the corresponding sender or recipient ID.

Package

WmMarketConnect

Input Parameters

marketConnectAlias	**String** The alias of the MarketSite from which you are receiving the envelope.
type	**String** The type of ID to look up. Values are `local` (sender DDID) and `remote` (recipient DDID).
DDID	**String** The sender DDID or recipient DDID. This parameter works with the *type* parameter.

Output Parameters

ddRecord (DDRecord)	**Record** The DDID specified in the *DDID* input parameter, the corresponding sender or recipient ID, and the xCBL version specified in the mapping.

getAttachmentList

Gets all attachments from an envelope.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose attachments to get.

Output Parameters

attachmentCount	**String** The number of attachments found in the envelope. If no attachments are found, this value is 0.
attachmentList	**Object** list that contains an array of attachments.

getDocumentAsNode

Gets the xCBL document from an envelope and returns a webMethods document known as a node.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose xCBL document to get.
encoding (optional)	**String** The encoding to use when parsing the xCBL document. If you do not specify this parameter, the XML parser uses the encoding indicated by the encoding attribute in the xCBL document header.
validate (optional)	**String** Whether to validate the xCBL document. The service uses the Commerce One CXP parsing engine and the SOX schemas in the schema path specified in the C1 OnRamp configuration to validate documents. Values are `true` and `false`; if you set this parameter to `true` and the document is invalid, the service fails. If you do not specify this parameter, C1 OnRamp uses the default validation setting for incoming documents in the C1 OnRamp configuration (see Specify Default Document Validation).

Output Parameters

node	**Document** Document or node.
encoding (optional)	Encoding used by the XML parser to parse the xCBL document.

getDocumentAsRecord

Gets the xCBL document from an envelope and returns a record in the form of a webMethods object called a boundNode.

C1 OnRamp determines the version of the xCBL document and uses the appropriate Software AG record definition for that version.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose xCBL document to get.
recordName (optional)	**String** The type of record to return. If you do not specify this parameter, C1 OnRamp determines the version of the xCBL document and uses the appropriate Software AG record definition for that version. Specify this parameter only when you want the service to use a custom record definition that C1 OnRamp does not supply. Specify the fully qualified name of the record. Use the name format that is expected by the Integration Server service documentToRecord in the WmPublic package. (For information, see the *webMethods Integration Server Built-In Services Reference* for your release.)
validate (optional)	**String** Whether to validate the xCBL document. The service uses the Commerce One CXP parsing engine and the SOX schemas in the schema path specified in the C1 OnRamp configuration to validate documents. Values are `true` and `false`; if you set this parameter to `true` and the document is invalid, the service fails. If you do not specify this parameter, C1 OnRamp uses the default validation setting for incoming documents in the C1 OnRamp configuration (see Specify Default Document Validation).

Output Parameters

boundNode	**Record** The top-level tag of the retrieved xCBL document. For example, if the document is a PurchaseOrder, boundNode contains a record called PurchaseOrder.

getDocumentAsString

Gets the xCBL document from an envelope and returns a string.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose xCBL document to get.
validate (optional)	**String** Whether to validate the xCBL document. The service uses the Commerce One CXP parsing engine and the SOX schemas in the schema path specified in the C1 OnRamp configuration to validate documents.

	Values are `true` and `false`; if you set this parameter to `true` and the document is invalid, the service fails. If you do not specify this parameter, C1 OnRamp uses the default validation setting for incoming documents in the C1 OnRamp configuration (see Specify Default Document Validation).

Output Parameters

documentString	**String** The entire xCBL document, including SOX processing instructions.

getDocumentHeader

Gets header information for the xCBL document that is contained in an envelope and returns the information as a record.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose xCBL document header information to get.

Output Parameters

documentHeader	**Record** The xCBL document header information.

getEnvelopeHeader

Gets header information for an envelope and returns the information as a record.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose header information to get.

Output Parameters

envelopeHeader	**Record** The header attributes of the envelope.

getNamedAttachment

Gets a specific attachment from an envelope.

Package

WmMarketConnect

Input Parameters

envelope	Envelope from which to get the attachment.
attachmentName	**String** The name of the attachment to get.

Output Parameters

attachment	Specified attachment.

getRequestMode

Gets the request mode of an envelope from the envelope header.

Package

WmMarketConnect

Input Parameters

envelope	Envelope whose request mode to get.

Output Parameters

requestMode	**String** The envelope's request mode.

pub.marketconnect.parse

parseConfiguration

The parseConfiguration record contains these fields:

Field	Contents
schemaPath	**String** The path to the SOX schema against which to validate an xCBL document.
validate	**String** Whether you want to validate the xCBL document. Values are `true` or `false`; the default is `false`.

Package

WmMarketConnect

parseString

Validates or does not validate an xCBL document.

This service lets you override the default validation setting for outgoing documents in the C1 OnRamp configuration.

If the document is valid, the service succeeds. If the document is not valid, the service fails.

Package

WmMarketConnect

Input Parameters

documentString	**String** The xCBL document to validate.
parseConfiguration (optional)	**Record** The path to the SOX schema against which to validate the document and that indicates whether to validate the document. If you do not specify this parameter, the service uses the schema path and the default validation setting for outgoing documents in the C1 OnRamp configuration (see "Specify Default Document Validation" on page 20).

Output Parameters

None.

pub.marketconnect.reply

createAcknowledgement

Creates a reply envelope that contains a custom acknowledgement of an asynchronous request and passes the reply envelope to the receiveEnvelope service to return to MarketSite.

C1 OnRamp automatically returns a reply envelope containing a default acknowledgement to any asynchronous request for which there were no processing errors. Use this service if you want to send a custom acknowledgement instead.

Package

WmMarketConnect

Input Parameters

messageId	**String** The message ID of the acknowledgement. Specify the message ID of the envelope you are acknowledging.
recipientId	**String** The recipient of the acknowledgment. Specify the sender ID of the envelope you are acknowledging.
correlationId	**String** The correlation ID of the acknowledgment. Specify the correlation ID of the envelope you are acknowledging.
senderId (optional)	**String** The sender of the acknowledgement. If you do not specify this parameter, the service uses your MarketSite TPID.

Output Parameters

replyEnvelope	Reply envelope that contains the custom acknowledgement.

createError

Creates a reply envelope that contains a custom error for a request and passes the reply envelope to the receiveEnvelope service to return to MarketSite.

C1 OnRamp automatically returns a reply envelope containing a default error to any request for which there were processing errors. Use this service if you want to send a custom error instead.

Package

WmMarketConnect

Input Parameters

recipientId (optional)	**String** The recipient of the error. If you do not specify this parameter, the service uses the sender ID of the envelope for which the error is being returned.
errorCode (optional)	**String** The error code for the error.
errorMessage	**String** The error message for the error.
errorLanguage (optional)	**String** The language (for example, English) of the error.
correlationId (optional)	**String** The correlation ID of the error. If you do not specify this parameter, the service uses the correlation ID of the envelope for which the error is being returned.
errorSeverity (optional)	**String** The severity of the error.
senderId (optional)	**String** The sender of the error. If you do not specify this parameter, the service uses your MarketSite TPID.

Output Parameters

replyEnvelope	Reply envelope that contains the custom error.

pub.marketconnect.transport

EnvelopeHandlerSpecification

Specification for envelope handler services. All envelope handler services must adhere to this specification.

For synchronous requests that were processed with no errors, the handler service must create a reply envelope that contains the appropriate xCBL response document and any necessary attachments. In the reply envelope header, the handler service must use the correlationId that was used in the header of the incoming envelope. The handler service must return the reply envelope to the receiveEnvelope service to return to MarketSite.

For asynchronous requests that were processed with no errors, C1 OnRamp automatically returns a reply envelope containing a default acknowledgement to MarketSite. For these requests, then, the handler service does not have to create or return a reply envelope.

For synchronous and asynchronous requests that had processing errors, C1 OnRamp automatically returns a reply envelope containing a default error to MarketSite. For these requests, then, the handler service does not have to create or return a reply envelope.

Package

WmMarketConnect

Input Parameters

envelope	Envelope received from the receiveEnvelope service.

Output Parameters

replyEnvelope	Reply envelope to return to the receiveEnvelope service.

receiveEnvelope

Receives incoming envelopes from MarketSite, gets the document type from each envelope, and forwards the envelope to the appropriate envelope handler service. The receiveEnvelope service also receives reply envelopes and returns them to MarketSite in these cases:

- When a synchronous request was processed with no errors, the receiveEnvelope service receives a reply envelope containing an xCBL response document and any attachments from the envelope handler service and returns the reply envelope to MarketSite.
- When an asynchronous request was processed with no errors, if you have created a custom acknowledgement, the receiveEnvelope service receives a reply envelope containing the custom acknowledgement from the createAcknowledgement service and returns the reply envelope to MarketSite.
- When a synchronous or asynchronous request had processing errors, if you have created a custom error, the receiveEnvelope service receives a reply envelope containing the custom acknowledgement from the createError service and returns the reply envelope to MarketSite.

Package

WmMarketConnect

Input Parameters

envelope	Envelope received from MarketSite.

Output Parameters

replyEnvelope	Reply envelope to return to MarketSite.

sendEnvelope

Sends an envelope to MarketSite.

Package

WmMarketConnect

Input Parameters

envelope	Envelope to send to MarketSite.
marketConnectAlias	**String** The alias for the MarketSite to which you are sending the envelope. This string is provided by the Software AG Partner Manager.
failOnError (optional)	**String** Whether this service fails if MarketSite returns an error. Value are `true` and `false`. If you do not specify this parameter, the service fails if MarketSite returns an error. If you want the service to continue executing even if MarketSite returns an error, specify `false`. You will receive the error in the replyEnvelope parameter, and the errorOccurred parameter will contain `true`. (This service might still fail for other reasons; for example, it will fail if it cannot reach MarketSite.)

Output Parameters

replyEnvelope	Reply envelope you receive from MarketSite. Depending on the document type and request mode of the envelope you sent, the reply envelope might contain an acknowledgement, an error, or a synchronous response.
errorOccurred	When the failOnError input parameter is set to `false`, string that indicates whether MarketSite returned an error. Values are `true` and `false`.

setDocumentHandler

Registers an envelope handler service and the document type it handles in a lookup table. The receiveEnvelope service uses the lookup table to find the appropriate envelope handler service for each incoming xCBL document.

Package

WmMarketConnect

Input Parameters

handlerService	**String** The fully qualified path to the envelope handler service to register.
documentType	**String** The document type the envelope handler service handles (for example, PurchaseOrder).

Output Parameters

None.

setHandler

Registers your default envelope handler service with C1 OnRamp.

This service has been superseded by the setDocumentHandler service. The setDocument handler service registers an envelope handler service and the document type it handles in a lookup table. The receiveEnvelope service uses the lookup table to find the appropriate envelope handler service for each incoming xCBL document. You can call the setDocumentHandler service any number of times to register any number of handler services, and you can name handler services anything you like.

The setHandler service, conversely, registers only a default envelope handler service with C1 OnRamp, and the receiveEnvelope can only find other handler services if they are named using a rigid naming convention-each handler service must have the same name as the default handler service, with the document type appended to that name. In addition, you cannot use the setHandler service to register a different handler service (for example, from a different package such as the WmMarketConnectExample package) during a C1 OnRamp session; if you call the service more than once, the service fails. If you want to register a different handler service, you have to reload the WmMarketConnect package, then call the setHandler service.

If you use both services, the receiveEnvelope refers first to the lookup table to find the appropriate envelope handler service for each incoming xCBL document, and only uses the handler services registered by the setHandler service when no other handler services apply.

Package

WmMarketConnect

Input Parameters

handlerService	**String** that specifies the fully qualified path to the envelope handler service to register.

Output Parameters

None.

pub.marketconnect.roundtrip.request

RoundTripRequest

Defines the structure of RoundTrip requests you receive from buyers.

See Commerce One's RoundTrip documentation for detailed information on the fields in this record.

Package

WmMarketConnect

pub.marketconnect.roundtrip.response

RoundTripResponse

Defines the structure of RoundTrip responses you build from shopping carts.

See Commerce One's RoundTrip documentation for detailed information on the fields in this record.

Package

WmMarketConnect

catalogInfo

The catalogInfo record contains these top-level items:

Item	Structure
CatalogID	**Record**
Description (optional)	**Record**
Text (optional)	**Record List**
Version (optional)	**Record**
ValidFrom (optional)	**Record**
ValidUntil (optional)	**Record**

Item	Structure
Category (optional)	**Record List**

Each record contains several strings, and each record list contains strings and nested record lists, records, and strings.

See Commerce One's RoundTrip documentation for detailed information on the fields in this record.

Package

WmMarketConnect

productInfo

The productInfo record contains these top-level items:

Item	Structure
ProductType (optional)	**String**
ProductID (optional)	**Record**
CatalogKey (optional)	**Record**
ParentCategoryID (optional)	**Record List**
BasicUnitOfMeasure (optional)	**Record**
ValidFrom (optional)	**Record**
ValidUntil (optional)	**Record**
Description	**Record**
CatalogText (optional)	**Record List**
Thumbnail	**Record List**
Picture (optional)	**Record**
Attachment	**Record List**
Attribute (optional)	**Record List**
ShoppingBasketItem (optional)	**Record**
ManufacturerDescription (optional)	**Record**
VendorDescription (optional)	**Record List**

Each record contains several strings, and each record list contains strings and nested record lists, records, and strings. See Commerce One's RoundTrip documentation for detailed information on the fields in this record.

Package

WmMarketConnect

catalogHeader

A catalogHeader record contains these top-level items:

Item	Structure
DataSource (optional)	**Record**
TradingPartners (optional)	**Record**

Each record contains strings and records. See Commerce One's RoundTrip documentation for detailed information on the fields in this record.

Package

WmMarketConnect

addCatalogToRoundTripResponse

Adds a catalog to a RoundTrip response.

This service uses the specification wm.marketconnect.roundtrip:addCatalogToRoundTripResponseSpecification.

Package

WmMarketConnect

Input Parameters

roundTripResponse	RoundTrip response to which to add a catalog.
catalogInfo	**Record** The RoundTrip catalog information.

Output Parameters

roundTripCatalog	RoundTrip catalog.

addProductToRoundTripCatalog

Adds a product to a catalog in a RoundTrip response.

This service uses the specification wm.marketconnect.roundtrip:addProductToRoundTripResponseSpecification.

Package

WmMarketConnect

Input Parameters

roundTripCatalog	RoundTrip catalog to which to add the product.
productInfo	**Record** The RoundTrip product information.

Output Parameters

None.

createRoundTripResponse

Creates an empty RoundTrip response to which you can add catalogs and products.

This service uses the specification wm.marketconnect.roundtrip:createRoundTripResponseSpecification.

Package

WmMarketConnect

Input Parameters

catalogHeader	**Record** Header information for a RoundTrip catalog.

Output Parameters

roundTripResponse	Empty RoundTrip response.

getRoundTripResponseAsHTML

Converts a RoundTrip response into an HTML string.

Package

WmMarketConnect

Input Parameters

roundTripResponse	RoundTrip response to convert.

Output Parameters

roundTripResponseHTML	Resulting HTML string.

getRoundTripResponseAsRecord

Converts a RoundTrip response into a record.

Package

WmMarketConnect

Input Parameters

roundTripResponse	RoundTrip response to convert.

Output Parameters

roundTripResponseRecord	Resulting record.

getRoundTripResponseAsXML

Converts a RoundTrip response into an XML string.

Package

WmMarketConnect

Input Parameters

roundTripResponse	RoundTrip response.

Output Parameters

roundTripResponseXML Resulting XML string.

wm.PartnerMgr.gateway.transport.B2B

InboundProcess

Submits a document to the Partner Manager.

Package

WmPartners

Input Parameters

sender	**String** Specifies the sender of the message. The Partner Manager matches this string with the Sender parameter in its routing rules to decide how to route the document.
receiver	**String** Specifies the receiver of the message. The Partner Manager matches this string with the Receiver parameter in its routing rules to decide how to route the document.
msgType	**String** Specifies the type of document you are sending. The Partner Manager matches this string with the Message Type parameter in its routing rules to decide how to route the document.
$routeOnly	**String** Optional. Specifies whether you want Partner Manager to log the document in its message store. Valid values are: ■ `true` Route the document without logging it in the message store. The *$routeOnly* parameter must exist in the pipeline for this to take effect. ■ `false` Default. Log the document in the message store.
$tid	**String** Optional. Specifies the transaction ID that you want Partner Manager to assign to the document when it puts it into the message store. Used only when *$routeOnly* is false. If you do not specify a transaction ID and *$routeOnly* is false, the Partner Manager automatically generates a transaction ID for the document.

$action	**String** Optional. Specifies one of the following action codes, which represents the state of the document. Used only when *$routeOnly* is false. Set the value to assign a state as follows: ■ `0` Create ■ `1` Execute ■ `2` Rollback ■ `3` Commit ■ `4` Confirm

Output Parameters

None.

OutboundProcess

Invokes a service on a local or remote webMethods Integration Server. This is the transport service that the Partner Manager invokes for routing rules that deliver their documents via the IS Service transport. When this transport executes, it passes the entire pipeline to the target service.

Package

WmPartners

Input Parameters

$tid	**String** Optional. Represents a transaction ID. If you want to invoke the service using guaranteed deliver, specify a transaction ID. If you are not using guaranteed delivery, do not specify a transaction ID.
$action	**String** Optional. Specifies one of the following action codes. Set the value to assign a state as follows: ■ `0` Create ■ `1` Execute ■ `2` Rollback ■ `3` Commit ■ `4` Confirm

transportParams An IS Document (an IData object) containing the following elements.

- *serverAlias* **String** Specifies the alias (the named set of connection parameters) defined for the webMethods server on which the target service resides.

  The default is `local`. The default specifies that the target service resides on the machine where Partner Manager is running, and that the target service is not password protected. (If the service is password protected, you must define an alias for the local server and specify that alias in *serverAlias*.)

- *servicePath* **String** Specifies the name of the folder in which the target service resides.
- *service* **String** Specifies the name of the target service.
- *valueScope* **String** Optional. Specifies how you want to maintain the routing service's connection information when the document is routed to a service on a remote server.

  Valid values are:

  - `SESSION` Maintain connection information within the routing service's session. This preserves the session information for the exclusive use of the routing service while it is connected to the remote server. Use this option if the routing service will perform stateful interactions with the remote server.
  - `GLOBAL` Maintain connection information in a shared session pool, which makes it available for use by other services that may want to connect to the same server. Since the data is not protected from other services, you should only use this option if the routing service's interaction with the remote server is stateless (i.e., it is not dependent on the availability of session information after a connection has been established).

Output Parameters

The results of the target service.

wm.PartnerMgr.gateway.transport.EmailTransport

OutboundProcess

Sends a document as an e-mail attachment to specified recipients. This is the transport service that the Partner Manager invokes for routing rules that deliver their documents via the Email transport. When this transport executes, it e-mails the contents of *data*/*content* to the specified recipients.

Important:
This service is for use by the Partner Manager only. If you want to send a message directly to an e-mail address, use pub.client:smtp, instead.

Note:
Internally, this service invokes pub.client:smtp, which is in the WmPublic package.

Package

WmPartners

Input Parameters

data	An IS Document (an IData object) containing the following element: *content* An object that contains the data that you want the transport to send. This object must be one of the following types: String, byte[ ], or InputStream.
transportParams	An IS Document (an IData object) containing the following elements: ■ *contenttype* **String** Specifies the content type of the document (e.g., MIME type). ■ *to* **String** Specifies the e-mail address to which you want the transport to send the document in *data*/*content*. ■ *cc* **String** Specifies the e-mail addresses where you want the transport to send copies of the document in *data*/*content*. To specify multiple recipients, separate each address with a comma. ■ *bcc* **String** Specifies the e-mail addresses where you want the transport to send blind copies of the document in *data*/*content*. To specify multiple recipients, separate each address with a comma. ■ *subject* **String** Specifies the text that will appear in the subject line of the e-mail message. ■ *from* **String** Specifies the sender of the email message. ■ *mailhost* **String** Specifies the host name of the SMTP server to which you want the document delivered.

Output Parameters

None.

wm.PartnerMgr.gateway.transport.FTPTransport

OutboundProcess

FTPs a document to a specified host. This is the transport service that the Partner Manager invokes for routing rules that deliver their documents via the FTP transport. When this transport executes, it FTPs the contents of *data/content* to the specified file location.

Package

WmPartners

Input Parameters

data	An IS Document (an IData object) containing the following element: *content* An object that contains the data that you want the transport to send. This object must be one of the following types: String, byte[ ], or InputStream.
transportParams	An IS Document (an IData object) containing the following elements: ■ *serverhost* **String** Specifies the host name of the FTP server to which you want the transport to send the document. ■ *serverport* **String** Specifies the port number on which the FTP server listens for incoming requests. The default is 21. ■ *username* **String** Specifies the user ID that you want the transport to use when it connects to the FTP server. ■ *password* **String** Specifies the password that you want the transport to use when it connects to the FTP server. ■ *dirpath* **String** Specifies the directory into which you want the document copied. **Note:** You do not need to specify a file name, because the transport automatically assigns a name to the documents it sends. This name has the format, "ftpfile*xx*.data", where *xx* is an ID that the FTP transport defines.

Output Parameters

None.

wm.PartnerMgr.xtn.Sweeper

sweepTRX

Purges transactions in the message store when invoked. It also invokes the System finalizer and the garbage collector. You can schedule this service so that it purges the message store periodically.

Package

WmPartners

Input Parameters

onState	The transaction state. For example, "Confirmed".
elapsedTime	The number of minutes that the transaction has been in *onState*.
maxTrxCount	The maximum number of transactions to delete each time that sweepTRX is invoked.

Output Parameters

None.

wm.marketconnect.xcbl

The following services are added to support xCBL 4.0 documents:

- wm.marketconnect.xcbl:xCBL4_GenerateRecords
- wm.marketconnect.xcbl:xCBL4_RegisterTNDocTypes
- wm.marketconnect.xcbl: xCBL4_DeleteTNDocType

xCBL4_GenerateRecords

Creates DocTypes for xCBL 4.0 document. The core and external schemas(dgs) are shipped with the package.

Input Parameters

useDefaultLocation	If set to *true*, then the DocTypes are created in the default location. The default location is, `WmxCBL.org.xcbl.schemas.xcbl.v4_0.` If set to *false*, then provide the *packageName* and the *ifcName*.

packageName	Specify the package name to store the DocTypes if the *useDefaultLocation* is *false*.
ifcName	Specify the interface name to store the DocTypes if the *useDefaultLocation* is *false*
messagesGroupType	DocType category. For example, financial group type.
generateALL	Specifies whether to generate all the root elements defined in the messagesGroupType xsd.
messageTypes	If the *generateALL* is set to false, then provide the root element names of *messagesGroupType* to generate the DocTypes it. For example : FXRateRequest, Invoice for Financial messageGroupType.

Output Parameters

success	Specifies the result of creating the DocTypes. It can either be *true* or *false*.
errorMessages	Specifies if the error message while creating the DocTypes.

xCBL4_RegisterTNDocTypes

This service registers DocTypes xCBL 4.0 document in Trading Networks.

Input Parameters

messageDocTypes	Specifies fully qualified namespace for the DocTypes which needs to be registered with Trading Networks

Output Parameters

success	Specifies the result of registering the DocTypes. It can either be *true* or *false*.
errorMessages	Specifies if the error message while creating the DocTypes.

xCBL4_DeleteTNDocType

This service deletes the registered DocTypes for xCBL 4.0 document from Trading Networks.

Input Parameters

docType	Specify only the DocType name that needs to be deleted from Trading Networks

Output Parameters

success	Specifies the result of deleting the registered DocTypes. It can either be *true* or *false*.

B Using C1 OnRamp with webMethods Trading Networks

Overview

The WmMarketConnectTN package enhances webMethods OnRamp for Commerce One MarketSite by using certain features of webMethods Trading Networks.

Important:
This chapter assumes you are familiar with Trading Networks.

My webMethods is a web-based administration and monitoring user interface for managing your webMethods components. For Trading Networks functions, you can use it to perform the following tasks:

- Define partner profiles, partner groups, trading partner agreements (TPAs), custom attributes, TN document types, processing rules, and queues.
- Perform configuration tasks.
- Monitor Trading Networks transactions, service execution tasks, delivery tasks, and activity logs.

For information about the tasks you can perform with My webMethods, see the *webMethods Trading Networks Administrator's Guide* for your release.

Note:
Beginning with Trading Networks 8.2, Trading Networks Console is deprecated. All functionality has been migrated to My webMethods and all Trading Networks Console-specific information has been removed from the guides. If you need information about Trading Networks Console features, see an earlier version of the Trading Networks documentation.

Features

The WmMarketConnectTN package provides a delivery service for reliable delivery of envelopes. The package adds two attributes to the Trading Networks Server:

Attribute	Used to . . .
C1DeliveryAlias	Deliver documents
requestMode	Send and receive documents

In the Trading Networks user interface, these attributes appear on the Document Attributes window, accessible from the **Document Types** task.

The WmMarketConnectTN package adds an external ID type called **TPID** to the external identifier list for profiles. In the Trading Networks user interface, the TPID appears on the **Corporate** tab.

The WmMarketConnectTN package also makes it easier to receive and process Commerce One envelopes.

If you are using Trading Networks 7.1 or later, the WmMarketConnectTN package supports Trading Networks conversation scripts generated from Designer process models.

Configure the WmMarketConnectTN Package

To configure the WmMarketConnectTN package, you must do the following:

Configure Routing of Incoming Envelopes

The WmMarketConnectTN package registers a single, default envelope handler service with C1 OnRamp using the C1 OnRamp setHandler service. The WmMarketConnectTN envelope handler service forwards all envelopes you receive to Trading Networks to be handled by the Trading Networks processing engine.

Important:
In your MarketSite accounts, the destination for your incoming envelopes is the C1 OnRamp receiveEnvelope service (https://*host:port*/invoke/pub.marketconnect.transport/receiveEnvelope?marketConnectAlias=*alias*). Do not change this destination to the Trading Networks receive service; let the WmMarketConnectTN envelope handler service forward your incoming envelopes to Trading Networks.

If there are any document types that you do not want Trading Networks to process, create document type-specific handler services in C1 OnRamp and register them using the C1 OnRamp setDocumentHandler service. Envelope handler services registered using the setDocumentHandler service take precedence over the default envelope handler service registered by the setHandler service.

If you have another C1 OnRamp package installed that registers a default envelope handler service using the setHandler service (for example, the WmMarketConnectExample package), disable that package and restart Integration Server to register the WmMarketConnectTN envelope handler service.

Define C1 OnRamp Aliases

For each trading partner, you must define at least one alias in C1 OnRamp, as follows:

- Define an alias for each MarketSite with which you want to communicate.
- Define an alias for each trading partner with which you want to communicate directly (that is, without going through any marketplace).
- If you are operating as a hub and are using Trading Networks to connect your partners through marketplaces, you can define one or both of two aliases for each partner-an alias for communicating with the marketplace associated with the partner and an alias for communicating with the partner directly.

- An alias called "default" that specifies the URL for a service that receives all envelopes that do not get sent to other aliases. For example, the default alias could point to a service that captures all envelopes for which a destination could not be detected and thus could not be delivered.

Define Trading Networks Partner Profiles

To define Trading Networks partner profiles

1. In Trading Networks, define a partner profile for each trading partner. The WmMarketConnectTN package adds an external ID type called **TPID** for profiles to the Trading Networks Server. The TPID appears on the **Corporate** tab.

2. In each profile, set the TPID to the TPID you specified in the alias for the trading partner in C1 OnRamp (that is, in the **MarketSite Authorizing Entity/TPID** box on the Add HTTPS Alias page).

3. For each trading partner with which you are exchanging Commerce One envelopes, identify the partner's C1 OnRamp aliases. You specify the aliases on the **Extended Fields** tab.

4. In the **MarketSite Alias** box, specify the alias you defined in C1 OnRamp for the MarketSite to which the trading partner belongs. In the **Direct Alias** box, specify the alias you defined in C1 OnRamp to communicate with the trading partner directly.

Import Document Type Definitions into Trading Networks

Trading Networks uses document type definitions to recognize documents and route them to the appropriate processing rules. The WmMarketConnectTN package provides definitions for every xCBL Version 3.5, 3.0, and 2.0 r3 document type; each definition is based on the root tag of the document. You can import the document type definition for each xCBL document you are going to send to Trading Networks for processing.

The document type definitions are located in the *Integration Server_directory* \packages\WmMarketConnectTN\config\xCBL35, xCBL30, and xCBL20 directories and in the files xCBL35.tnf, xCBL30.tnf, and xCBL20r3.tnf. You can import all document type definitions for a particular version, or you can import only the document type definitions you need. See the Trading Networks documentation for instructions on importing document types.

If you want Trading Networks to determine the version of an incoming xCBL document, you can set up the document type definition to extract the first root node in the document (that is, the document type header or the SOX header).

The WmMarketConnectTN package associates each document type definition with the corresponding record structure in the WmxCBL package. When Trading Networks calls a service such as itswm.tn.doc.xml:bizdocToRecord to get the xCBL document from an envelope and return it as a record, you do not have to identify the record structure for the service.

Use the C1 OnRamp Test Send/Receive utility to make sure Trading Networks can recognize xCBL documents using the document type definitions you imported. (See "Test a Service that Sends, Receives, or Processes Envelopes" on page 40 for instructions.)

Create a Processing Rule for Sending Envelopes

To create a processing rule for sending envelopes

1. In xCBL documents, the sender and receiver are not identified in the document. If you want to send xCBL documents, create a wrapping XML document that contains an xCBL document and identify the document sender and receiver in the wrapping XML document so Trading Networks can extract that information.

2. In Trading Networks, create a processing rule for the document type you want to send and define the criteria for the processing rule.

   If you created a larger XML document around the relevant document type definition that identifies the document sender and receiver, strip out the xCBL document, then call the Trading Networks service wm.tn.doc:persist to persist the xCBL document to the Trading Networks database as the xmldata content part. If you want to also save a copy of the larger XML document for auditing purposes, call the Trading Networks service wm.tn.doc:addContentPart to persist the larger XML document to the database as another content part (for example, the source content part).

3. You use the Trading Networks reliable delivery service to send documents to a MarketSite or trading partner. In the processing rule, choose reliable delivery by clicking the **C1 Envelope HTTP(S) service** option provided by the WmMarketConnectTN package in the delivery method list.

   Note:
   If you are communicating with a MarketSite over HTTPS, and the MarketSite requires you to not submit certificates for authentication, but you have certificates in Integration Server that are submitted by default, contact Software AG Global Support for a workaround.

4. If you are sending an xCBL response document, in the document type definition, set the **ConversationID** system attribute to the **correlationID** field in the document's envelope header of the corresponding request document. The WmMarketConnectTN package maps the **ConversationID** system attribute in the document type definition to the **correlationID** field in the document's envelope header so you can easily perform conversational transactions with MarketSite. In addition, you can also use theTrading Networks user interface to find related documents or Designer to model processes and generate corresponding conversations.

5. Specify attributes for the delivery job, as follows:

   - Call the appropriate Trading Networks service to set the **requestMode** attribute for the outgoing envelopes. Values for this attribute are **sync**, **oneway**, and **peer-peer**.

     Important:

You must set this attribute or the reliable delivery service will fail.

- If you do not want to send the envelopes to the C1 OnRamp default alias, specify the alias to which to send the envelopes. Call the appropriate Trading Networks service to set the **C1DeliveryAlias** attribute for outgoing envelopes. Values for this attribute are:

Value	Use the Destination Specified in the:
`Sender Market`	**C1 OnRamp MarketSite Alias**box in the Trading Networks profile for the sender.
`Receiver Market`	**C1 OnRamp MarketSite Alias**box in the Trading Networks profile for the receiver.
`Direct`	**C1 OnRamp Direct Alias**box in the Trading Networks profile for the receiver.
alias	Any C1 OnRamp alias.
none (empty)	C1 OnRamp alias named **default**.

6. If you want to create a complex envelope that contains more than an xCBL document (for example, contains attachments), call C1 OnRamp services to create the envelope and add the attachments. Leave the envelope object in your pipeline under the name **envelope** so the reliable delivery service can detect it.

   If you are not including attachments in the envelope and the document you are sending includes an xmldata content part, provide the document you want to send as input to the delivery job and let the delivery service create the envelope.

7. When you send an envelope to MarketSite using the **C1 Envelope HTTP(S)** delivery service provided by the WmMarketConnectTN package, MarketSite returns one of three replies, as follows:

Reply	Processing
Error	The delivery service persists the error to the Trading Networks database as a content part of the document you sent. The content part is called **Error Reply Envelope**.
Acknowledgment	The delivery service persists the acknowledgment as a content part of the document you sent. The content part is called **Acknowledgment Reply Envelope**.
Synchronous	The delivery service routes the reply envelope containing the response document through Trading Networks automatically and the delivery service automatically associates the two documents.

8. For reference and auditing purposes, the delivery service persists a copy of each envelope you send to the Trading Networks database as a content part of the document you sent. The content part is called **C1 Envelope(outgoing)**.

9. The delivery service looks up the TPID specified in the alias to which the envelope is to be delivered and sets the sender TPID in the envelope to that TPID.

Create a Processing Rule for Receiving Envelopes

When Trading Networks receives an envelope from the WmMarketConnectTN package envelope handler service, the WmMarketConnectTN package detects the sender and receiver in the envelope and uses that information to associate the document with the Trading Networks partner profiles for the sender and receiver. By default, Trading Networks extracts the **messageId** and **correlationId** fields from the envelope header and persists them to its database as the **DocumentID** and **ConversationID** system attributes for the envelope, respectively.

If you have a processing rule for a document from which Trading Networks extracts the **ConversationID** system attribute, but you do not want Trading Networks to perform conversation management on the document, use the **Execute a service** processing action to call the Trading Networks service wm.tn.cm:disableDocument (or to call another service that calls the wm.tn.cm:disableDocument service).

In addition, the WmMarketConnectTN package calls the C1 OnRamp service getRequestMode to extract the request mode from the incoming envelope and persists the request mode to the Trading Networks database as the **requestMode** attribute for the envelope.

Trading Networks also persists the incoming envelope to the database as the **C1EnvBytes** content part, for auditing purposes.

To create a processing rule for receiving envelopes

1. Build a C1 OnRamp envelope handler service. Have the envelope handler service first call the WmMarketConnectTN package pub.marketconnect.tn.util:getMarketConnectEnvelopeFromBizDocservice. This service extracts incoming envelopes from the Trading Networks database, translates them into the format expected by C1 OnRamp's services, and puts the translated object into your pipeline). Have the envelope handler service then call other typical C1 OnRamp services used in envelope handler services, such as "getDocumentAsRecord" on page 60.

   Important:
   You must call the pub.marketconnect.tn.util:getMarketConnectEnvelopeFromBizDoc service or you will not be able to reprocess a document in Trading Networks, nor will you be able to work with documents using the Trading Networks conversation manager or the Business Integrator.

   Do not register the envelope handler service you just created with C1 OnRamp.

2. In a document type definition for an xCBL document you are receiving, you can set the **DocumentID** and **ConversationID** system attributes to information you extract from the document. For example, if you receive an envelope containing a PurchaseOrder, you can have Trading Networks extract the PO number and set the **ConversationID** system attribute to it.

   If you want to set the **DocumentID** system attribute to the messageId from the document's envelope header, do not set the **DocumentID** system attribute or set it to an empty string (""). Likewise, if you want to set the **ConversationID** system attribute to the correlationId from the document's envelope header, do not set the **ConversationID** system attribute or set it to an empty string (""). If you want Trading Networks to associate response documents you receive in reply to an asynchronous request document, you must set the **GroupID**systemattribute of each document to the same value, set the **ConversationID** system attribute of each document to the same value, or both.

3. If you want Trading Networks to determine the version of an incoming xCBL document, set up the document type definition to extract the first root node in the document (that is, the document type header or the SOX header). In the identification section of the definition, define a query that specifies `/pi()[0]/source()` as the query and the SOX header as the value (for example, define the value as `<?soxtype urn:x-commerceone:document:com:commerceone:CBL:CBL.sox$1.0?>`.

4. In Trading Networks, create a processing rule for the document type you are receiving and define the criteria for the processing rule.

5. In the processing rule, call the envelope handler service you just created.

6. If the XML in the incoming envelopes contain a SOX header and you do not want the SOX header included in the objects returned by the envelope handler service, have the processing rule call the WmMarketConnectTN package service pub.marketconnect.tn.util:removeSOXHeader to remove the SOX header from the documents.

C Using the Partner Manager

Overview

The Partner Manager is a OnRamp for Commerce One MarketSite package (the WmPartners package) that manages the routing of documents based on a set of routing rules that you specify.

The Partner Manager:

- Accepts documents that you submit to via a special service
- Determines how the document is supposed to be routed
- Routes the document to the appropriate destination
- Tracks documents in its message store

How Does the Partner Manager Route a Document?

The Partner Manager determines how to route a document based on routing rules that you establish on your webMethods Integration Server. When the Partner Manager receives a document, it looks for a routing rule that matches the following attributes of the incoming document:

- Sender
- Receiver
- Message Type

When Partner Manager locates the routing rule that matches these criteria, it executes the routing action specified by that rule. Depending on how you define the rule, the routing action might deliver the document to:

- A service on the local machine or a remote machine
- An FTP location
- An SMTP mailbox

Basic Steps to Use the Partner Manager

Before your site begins using the Partner Manager, you must decide where you want to maintain the Partner Manager's *message store*—the place where the Partner Manager maintains an audit trail of documents that it processes. You can choose to maintain the message store in a log file on webMethods Integration Server or in a JDBC-compatible database that you are already using at your site. For information about setting up the message store, "Using the Message Store" on page 111.

After you configure the message store, you can begin using the Partner Manager to automatically route documents based on their attributes. To do this, you need to take the following general steps for each type of document that you want the Partner Manager to route:

1. Build a routing rule that defines the document's **Sender**, ***Receiver*** , and ***Message Type*** attributes and specifies what action you want the Partner Manager to take when it receives documents matching those attributes. For information about building routing rules, see "Building and Maintaining the Routing Rules" on page 11"Building and Maintaining the Routing Rules" on page 93.

2. Build the service that will be used to submit the document to the Partner Manager. For information about handing documents to the Partner Manager, see "Submitting Documents to Partner Manager" on page 107.

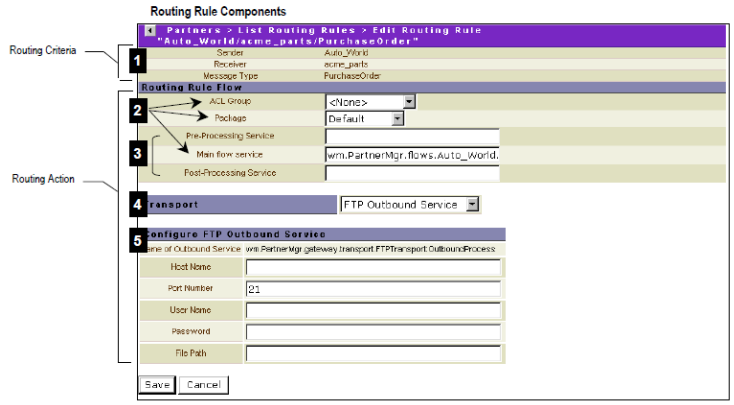
Building and Maintaining the Routing Rules

Routing rules tell the Partner Manager what to do with documents it receives. A routing rule has two basic elements: routing criteria and routing action.

- The routing criteria specify the sender, receiver, and message type values that cause the routing action to occur.
- The routing action is encompassed in a flow service (called the *routing service*). This service performs any pre-processing actions that you specify and then invokes the outbound transport, which delivers the document to its intended destination.

Components of a Routing Rule

You use the Integration Server Administrator to create, edit, and view routing rules. Each routing rule has the following basic components.



Number	Description
1	**The routing criteria**—the sender, receiver, and message type values that trigger the action specified by the rule.
2	**The ACL, package, and name of the routing service** associated with the rule. This is the actual service that the Partner Manager invokes when it receives a document matching the rule's routing criteria. The Partner Manager

Number	Description
	generates this service based on the action parameters you specify when you create a routing rule. You do not build this service manually.
3	**The names of the pre- and post-processing services**, if any, that the Partner Manager will execute before and after delivering the document to its destination. If either of these services fail, the transaction status will change to reflect the service failure. A pre-processing service is a service that you build to perform any work that must be executed immediately before the Partner Manager sends a document out. For example, you might use a pre-processing service to affix a digital signature to a document or mark it with a special postmark. A post-processing service is a service that you build to perform any work that must be executed immediately after the Partner Manager sends a document out. The use of these services is purely optional.
4	**The transport** that Partner Manager will use to deliver the document to its destination.
5	**Addressing parameters** that describe the document's specific destination. These parameters vary depending on the transport you select. For example, when you choose the Email transport, these parameters specify a host system and a particular mailbox; when you use the IS Service transport, they identify a particular service on a webMethods Integration Server.

The Routing Criteria

The routing criteria—the **Sender**, **Receiver**, and **Message Type** values—determine which rule is selected at run time. They uniquely identify a routing rule and act as the trigger for the rule's routing action. You can also use a wildcard character (*) to match all **Sender**, **Receiver**, or **Message Type** values.

This parameter...	Specifies...
Sender	An arbitrary string that identifies who sent the document, or a wildcard character (*).
Receiver	An arbitrary string that identifies the document's destination, or a wildcard character (*).
Message Type	An arbitrary string that specifies what type of information is in the document (e.g., a purchase order, a credit memo, an invoice, and so forth), or a wildcard character (*).

As noted previously, the values that you use as routing criteria are arbitrary or they can be wildcards (*). They do not need to specify a physical address or document. The routing criteria should match the *sender*, *receiver*, and *msgType* values that are submitted with the document. For example, if your trading partner submits invoices under a *sender* value of `MtnPaintPurch003`, a *receiver* value of `BboardsAP`, and a *msgType* of `Invoice`, the **Sender**, **Receiver**, and **Message Type** values in your routing rule must match these strings letter for letter. (These values are case sensitive).

Using Wildcards as Routing Criteria

You can use a single asterisk character (*), known as a wildcard, for the **Sender**, **Receiver**, and **Message Type** values in a routing rule. A wildcard matches any value that may be submitted with an incoming document. Some uses for wildcards include:

- Use a wildcard for the Sender parameter to serve as a "catch all" routing rule for a specific company. For example, suppose that you want to execute a particular routing rule for all documents from XYZ Company. You don't necessarily want to define routing rules for each message type that might be sent from XYZ Company, so you use the wildcard character (*) for the **Message Type**.
- Use a wildcard for all parameters (**Sender**, **Receiver**, and **Message Type)** to serve as a "last resort" routing rule. This routing rule will be executed when all other routing rules do not match the incoming document.

Routing Rule Precedence

A routing rule is executed when its **Sender**, **Receiver**, and **Message Type** values match an incoming document's values. When one or more of those values is a wildcard (*), then the routing rule is matched and executed based on its value precedence. See the following table for the default match order. The most specific rule is matched (and executed) first.

A rule with these values is matched...	Sender	Receiver	msgType
First	*arbitrary string*	*arbitrary string*	*arbitrary string*
Second	*	*arbitrary string*	*arbitrary string*
Third	*arbitrary string*	*	*arbitrary string*
Fourth	*arbitrary string*	*arbitrary string*	*
Fifth	*	*	*arbitrary string*
Sixth	*	*arbitrary string*	*
Seventh	*arbitrary string*	*	*
Eight (last)	*	*	*

Note:
If rule matching is performed by an external service via a getRule service, then the value precedence is dependent on your custom implementation. A custom getRule service uses

wm.PartnerMgr.gateway.getRule:getRuleSpecification to bypass the default routing rule functionality and returns the routing rule that you want invoked. The getRule service must be specified in the server.cnf file on the watt.WmPartners.getRule parameter. Any kind of wildcards can be implemented within the getRule service.

Example 1

Suppose that you have a routing rule with the following values:

Sender=`snd`

Receiver=`receiver`

msgType=`msg1`

If a message arrives with the following values:

Sender=`snd1`

Receiver=`receiver`

msgType=`msg1`

and a routing rule does not exist with those exact values, then the message will be matched to a routing rule with the following pattern:

Sender=`*`

Receiver= *`arbitrary string`*

msgType= *`arbitrary string`*

Example 2

Suppose that you have a routing rule with the following values:

Sender=`snd`

Receiver=`receiver`

msgType=`msg1`

If a message arrives with the following values:

Sender=`snd1`

Receiver=`receiver`

msgType=`msg2`

and a routing rule does not exist with those exact values, then the message will be matched to a routing rule with the following pattern:

Sender=`*`

Receiver= *arbitrary string*

msgType=*

To override the default match order, see "Overriding Routing Rule Match Order" on page 104.

When an Inbound Message Contains a Wildcard

If an inbound message contains an asterisk (*) for its *receiver* or *msgType* parameters, then the Partner Manager creates a mailbox directory on the server, replacing "*" with "WildCard". This is necessary because file systems do not allow special characters for directory or file names.

For example, suppose that you have a routing rule with the following values:

Sender=*

Receiver=`Rcv`

msgType=*

If a message arrives with the following values:

Sender=`sender`

Receiver=`Rcv`

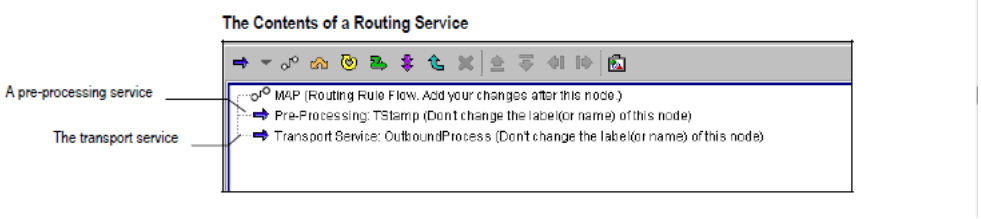
msgType=*

then the corresponding .values file is stored in the mailbox at:
\..\packages\WmPartners\pub\mailbox\Rcv\WildCard

The Routing Service

All of the action associated with a particular routing rule is encompassed within its routing service. The routing service executes the pre-processing service specified by the rule (if any), invokes the transport service, which delivers the document to a specified destination, and then executes the post-processing service specified by the rule (if any).

The following example shows the contents of a routing service. In this example, the routing service invokes a pre-processing service (`TStamp`) and then submits the document to the transport service (`OutboundProcess`).



The Partner Manager automatically generates a routing service when you create a routing rule. You do not build routing services manually, although you can use Designer to edit the services that the Partner Manager builds.

With respect to routing services, keep the following points in mind:

- You can use Designer to view the contents of a routing service.
- You can use Designer to incorporate additional pre- or post-processing steps into the routing service; however, you must insert your changes after the MAP step that appears at the top of the flow. For additional information about adding steps to a routing service, see "Editing a Routing Service" on page 106.
- You must not rename a routing service after you create a routing rule. Doing so will cause the routing rule to fail at run time because the Partner Manager will not be able to locate the service named in the routing rule.

Note:
If the routing rule contains a wildcard character for the Receiver or Message Type, the routing service name will have "WildCard" in the name, instead of "*".

Pre-Processing Services

When you define a routing rule, you can optionally instruct the Partner Manager to execute a pre-processing service before routing a document to its destination. For example, you might use a pre-processing service to obtain a special timestamp or affix a digital signature to the document. Because the document exists in the pipeline when the Partner Manager invokes the pre-processing service, you can even use this service to extract information from, or write information to, the document itself.

When you specify the name of a pre-processing service in the routing rule, the Partner Manager executes that service before it invokes the transport—that is, in the routing service, the invocation step for the pre-processing service appears before the invocation step for the transport service.

The Transport Service

The transport service (usually referred to simply as, the *transport*) performs the actual work of delivering a document to a specified destination.

When you create a routing rule, you specify which transport you want the Partner Manager to use to deliver the document. You also specify addressing information that the transport needs to deliver the document to the appropriate destination (e.g., a mailbox, file location, or service) at run time.

The following table identifies the transports you can use to deliver a document.

You use this transport...	To pass the document to...
IS Service	A specified service on the local Integration Server or on a remote Integration Server.
Email	A specified mailbox as an e-mail attachment using the Simple Mail Transfer Protocol (SMTP).
FTP	A specified file location using the File Transfer Protocol (FTP).

Post-Processing Services

When you define a routing rule, you can optionally instruct the Partner Manager to execute a post-processing service after routing a document to its destination.

When you specify the name of a post-processing service in the routing rule, the Partner Manager executes that service after the Outbound Process is complete in the routing service.

When a Routing Rule Does Not Exist

If the Partner Manager receives a document for which a routing rule does not exist (that is, it cannot locate a rule matching the document's *sender*, *receiver* and *msgType* values), the Partner Manager creates an *incomplete routing rule*—a rule with no action associated with it—for that document.

Viewing Incomplete Rules Created by the Partner Manager

When you view the list of routing rules on your Integration Server, incomplete routing rules appear with a red **Status** icon in the **View Routing Rules** screen.

An incomplete rule contains routing criteria (which was derived directly from the submitted document) but no routing action.

You use Integration Server Administrator to edit an incomplete rule and define its routing action (i.e., make it a complete, and therefore valid, routing rule). This will allow the Partner Manager to successfully route subsequent documents whose attributes match the rules' **Sender**, **Receiver**, and **Message Type** values. For information about editing an incomplete routing rule, see "Activating an Incomplete Routing Rule" on page 103.

Note:
There may be times when you want to deliberately trigger the Partner Manager's incomplete rule-generating capability. For instance, if a document's **Sender**, **Receiver**, and **Message Type** values are not known, you can quickly and accurately capture that information in a routing rule by simply submitting the document to the Partner Manager. An alternative is to use wildcard characters in the routing rule.

Creating a Routing Rule

Use the following procedure create a routing rule.

Important:
You must have administrator privileges on Integration Server to execute the following procedure.

This procedure requires you to select the package in which you want the rule's routing service registered. If this package does not already exist, create it before you begin. If you need procedures for creating a package, see *webMethods Integration Server Administrator's Guide*, or the *webMethods Service Development Help* for your release.

To create a routing rule

1. Start Integration Server Administrator.

2. On the Adapters menu, click **Routing**.

3. On the View Routing Rules screen, specify the routing criteria in the three text boxes under the **Sender**, **Receiver**, and **MsgType** columns, and click **Add Rule**.

   Remember that these values should match the *sender*, *receiver*, and **msgType** values that will be submitted with the document, or a wildcard character (*). These values are case sensitive.

4. On the Edit Routing Rule screen, set the following **Routing Rule Flow** parameters to specify where you want the routing service created.

In this field...	Do the following...
ACL Group	Select the ACL group to which you want to limit the routing service access. Only the users that are in groups that are in the selected ACL list will be able to use this service.
Package	Select the package in which you want the Partner Manager to register the routing service it generates.
Main flow service	Accept the default service name suggested by the Partner Manager or type a new, fully qualified name for the routing service. (This is the name that the Partner Manager will assign to the flow service it generates to perform the routing action. By default, the Partner Manager suggests a name that incorporates the sender, receiver, and message type information.) **Important:** If **Sender**, **Receiver**, or **Message Type** contains characters that your file system does not support, be sure to modify the suggested name to eliminate those characters. For example, if the message type were "XML\PurchaseOrder", you would eliminate the \ from the suggested service name. If **Receiver** or **Message Type** contains a wildcard character (*), then the Partner Manager automatically replaces it with "WildCard" in the service name.

5. If you want the routing service to invoke a pre-processing or post-processing service, type the fully qualified name of that service in the **Pre-Processing Service** field and **Post-Processing Service** field.

6. Select the transport that you want to use to deliver the document.

Select this transport...	To...
Service	Pass the document to a specified service on a local or remote Integration Server.
Email Outbound Service	Send the document to a specified mailbox as an attachment to an e-mail message using the Simple Mail Transfer Protocol (SMTP).
FTP Outbound Service	Send the document to a specified file location using the File Transfer Protocol (FTP).

7. Configure the transport as described in the following sections, then click **Save**.

To configure the...	See this section...
IS Service transport	"Configuring the IS Service Transport" on page 101
Email transport	"Configuring the Email Transport" on page 102
FTP transport	"Configuring the FTP Transport" on page 103

The routing rule appears in the View Routing Rules screen. By default, it is disabled. You can enable it by clicking **No** in the **Enabled?** column. Once enabled, it changes to **Yes**.

Configuring the IS Service Transport

Use the following procedure to complete the configuration parameters on the Edit Routing Rule screen if you are creating a routing rule that uses the IS Service transport. The IS Service transport passes the document to a specified service on a local or remote Integration Server.

To configure the IS Service transport, in the Configure IS Service section of the Edit Routing Rule screen, complete the following fields:

In this field...	Do the following...
Server Alias	Select the Integration Server on which the service resides, or select (local) if the service resides on the same server as the Partner Manager. (If the Integration Server you need does not appear in this list, you must create an alias for it. For information about defining aliases for remote Integration Servers, refer to the *webMethods Integration Server Administrator's Guide* for your release.)
Folder	Type the name of the folder in which the service resides.
Service	Type the name of the service to which you want to pass the document.
Scope	Specify how you want to maintain the routing service's connection information in cases where the document is routed to a service on a remote server. ■ Select **SESSION** if you want to maintain connection information within the routing service's session. This preserves this information for the routing

In this field...	Do the following...
	service's exclusive use while it is connected to the remote server. Use this option if the routing service will perform stateful interactions with the remote server. ■ Select **GLOBAL** if you want to maintain connection information in a shared session pool, making it available for use by other services that may need to connect to the same server. Because the data in this session is not protected from other services, you should only use this option if the routing service's interaction with the remote server is stateless (i.e., it is not dependent on the availability of session information after a connection has been established).

Configuring the Email Transport

Use the following procedure to complete the configuration parameters on the Edit Routing Rule screen if you are creating a routing rule that uses the Email transport. The Email transport sends the document to a specified mailbox as an attachment to an e-mail message.

To configure the Email Transport, in the Configure Email Outbound Service section of the Edit Routing Rule screen, complete the following fields.

In this field...	Do the following...
SMTP Host	Type the host name of the SMTP server to which you want the document delivered.
Content Type	Type the string that will be used to specify the content type of the e-mail message. For example: `application/x-edi-message`
From	Type the e-mail address that you want to use as the sender of the e-mail message. For example: `orderingsystem@buyer.com`
To	Type the address of the mailbox where you want the document delivered. For multiple mailboxes, separate each address with a comma. For example: `orders@seller.com,purch@buyer.com`
CC	If you want to send a copy of the message to another mailbox, type the address of that mailbox. For multiple mailboxes, separate each address with a comma. For example: `acctpayable@buyer.com,receiving@buyer.com.`
BCC	If you want to send a blind copy of the message to another mailbox, type the address of that mailbox. For multiple mailboxes, separate each address with a comma. For example: `manager@buyer.com,log@buyer.com`

In this field...	Do the following...
Subject	Type the string that you want to appear as the subject line in the e-mail message.

Configuring the FTP Transport

Use the following procedure to complete the configuration parameters on the Edit Routing Rule screen if you are creating a routing rule that uses the FTP transport. The FTP transport sends the document to a specified file location on an FTP server.

To configure the FTP transport, in the Configure FTP Outbound Service section of the Edit Routing Rule screen, complete the following fields.

In this field...	Do the following...
Host Name	Type the host name of the remote FTP server.
Port Number	Type the port number on which the remote FTP server listens for FTP requests.
User Name	Type the user ID that the Partner Manager must submit in order to connect to the FTP server.
Password	Type the password that the Partner Manager must submit in order to connect to the FTP server.
File Path	Type the directory in which you want the document transferred. The transport will automatically assign a file name to the file it puts in this directory. The name will have the format, "ftpfile*xx*.data", where *xx* is an ID that the FTP transport assigns.

Activating an Incomplete Routing Rule

Use the following procedure to complete the definition of an incomplete routing rule. An incomplete routing rule is a rule that the Partner Manager automatically generates from a document whose *sender*, *receiver*, and *msgType* values do not match any of the existing rules. (For more information about what causes Partner Manager to generate incomplete routing rules, see "When a Routing Rule Does Not Exist" on page 99.) To complete the rule, you must specify the routing action associated with the routing criteria.

To complete an incomplete routing rule

1. Start Integration Server Administrator.
2. On the Adapters menu, click **Routing**.
3. On the View Routing Rules screen, locate the incomplete rule that you want to edit. (Incomplete rules have a red indicator in the **Status** column.)

4. Click the **Edit** icon for the rule.

5. On the Edit Routing Rule screen, set the Routing Rule Flow and Transport parameters as described in "Creating a Routing Rule" on page 99.

6. Click **Save**.

Viewing the Routing Rules

Perform the following procedure to view the list of existing routing rules.

To view routing rules

1. Start Integration Server Administrator.

2. On the Adapters menu, click **Routing**. The View Routing Rules screen appears.

3. If you want to sort the list, click the up or down arrow icons in the column headings.

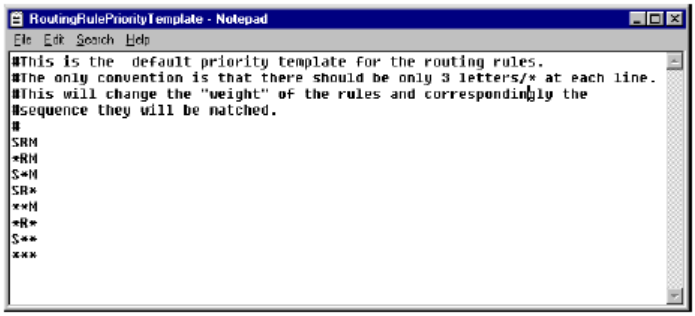
Overriding Routing Rule Match Order

When routing rules contain wildcards for the **Sender**, **Receiver**, or **Message Type**, there is a default order in which the routing rule is matched to the incoming document's values. (For details, see "Routing Rule Precedence" on page 95.) You can override this default order by modifying the RoutingRulePriorityTemplate.cnf file in your WmPartners package. See the following procedure.

To override routing rule match order

1. In a text editor such as Notepad, open the following file: *Integration Server_directory* \packages\WmPartners\config\RoutingRulePriorityTemplate.cnf

RoutingRulePriorityTemplate.cnf



2. Edit the file to reflect the match order priority that you want for rules containing wildcards. See the following table.

This character in the file...	Represents...
#	A comment. Put at the beginning of a line that contains a comment.
S	The *sender* value.
R	The *receiver* value.
M	The *msgType* value.
*	A wildcard. It will match any value.

3. Save the file.

4. In Integration Server, under **Packages**, click **Management**.

5. Next to **WmPartners**, click the **Reload** icon. The new routing rule match order is now in effect.

Note:
If rule matching is performed by an external service via a getRule service, then the value precedence is dependent on your custom implementation. A custom getRule service uses wm.PartnerMgr.gateway.getRule:getRuleSpecification to bypass the default routing rule functionality and returns the routing rule that you want invoked. The getRule service must be specified in the server.cnf file on the watt.WmPartners.getRule parameter. Any kind of wildcards can be implemented within the getRule service.

Editing a Routing Rule

You can use the following procedure to edit a routing rule and change its routing action.

Note that once you create a routing rule, you cannot modify its routing criteria. If you need to change the rule's criteria, you must delete the existing rule and create a new rule based on the new criteria.

Note:
When you modify a rule's routing action, the Partner Manager updates the rule's routing service. However, the changes you make only affect the first step in this service. If you added your own steps to the routing service, they will not be altered. For information about editing the routing service directly, see "Editing a Routing Service" on page 106.

To edit a routing rule

1. Start Integration Server Administrator.

2. On the Adapters menu, click **Routing**. The View Routing Rules screen appears.

3. Locate the rule that you want to edit. Click the **Edit** icon for the rule.

4. Edit the Routing Rule Flow and Transport parameters as necessary. If you need additional information about these parameters, see "Creating a Routing Rule" on page 99.

5. Click **Save**.

Disabling Routing Rules

Use the following procedure to disable a routing rule. When you disable a routing rule, all routing service and rule information is preserved until you re-enable it.

To disable a routing rule

1. Start Integration Server Administrator.

2. On the Adapters menu, click **Routing**. The View Routing Rules screen appears.

3. Locate the rule that you want to disable. Under **Enabled?**, click **Yes** until it turns to **No**.

Deleting Routing Rules

Use the following procedure to delete a routing rule.

Important:
When you delete a routing rule, the Partner Manager deletes the routing service associated with that rule.

To delete a routing rule

1. Start Integration Server Administrator.

2. On the Adapters menu, click **Routing**. The View Routing Rules screen appears.

3. Locate the rule that you want to delete. Click the **Delete** icon for the rule.

Editing a Routing Service

When necessary, you can edit a routing service with Designer to incorporate additional steps before or after the transport delivers the document. For example, you might want to insert flow-control steps (for example, LOOP or BRANCH steps) around the pre- or post-processing services, or you might want to include some error-handling steps immediately after the transport service.

Guidelines to Follow When Editing a Routing Service

When you edit a routing service, keep the following points in mind:

- Do not add any flow steps before the first MAP step.
- Do not modify the pre-processing service's label property (if a pre-processing service exists in the flow).
- Do not modify the transport service's label property.

Submitting Documents to Partner Manager

Submitting a Document to the Partner Manager

To submit a document to the Partner Manager, you create a flow service that invokes the following service:

wm.PartnerMgr.gateway.transport.B2B:InboundProcess

Your flow service must pass the document to this service, along with the appropriate sender, receiver, and message type values. The InboundProcess service invokes the Partner Manager, which inspects the document's sender, receiver, and message type values and selects the routing rule matching those values.

Passing a Document to the Transport Service

When you invoke InboundProcess, the Partner Manager receives the entire contents of the pipeline, including your document. However, you need to pass your document to the Partner Manager where the transport service expects it.

The following table describes where the Partner Manager expects the document for each transport type.

This Transport...	Sends…
IS Service Transport	The entire pipeline to the specified service. When a document is routed with this transport, it doesn't make any difference where you put it in the pipeline, because the entire pipeline is transmitted to the destination service.
Email Transport	The contents of *data*/*content* to the specified email address, where *data* is an IS Document (an IData object) and *content* is an object element of *data*. If your document will be routed through the email transport: ■ It must be in the form of a String, a byte [ ], or an InputStream. ■ You must map the document to *data*/*content* before invoking the Partner Manager.
FTP Transport	The contents of *data*/*content* to the specified file location, where *data* is an IS Document (an IData object) and *content* is an object element of *data*. If your document will be routed through the FTP transport:

This Transport...	Sends…
	■ It must be in the form of a String, a byte [ ], or an InputStream. ■ You must map the document to *data/content* before invoking the Partner Manager.

Building a Service that Invokes the Partner Manager

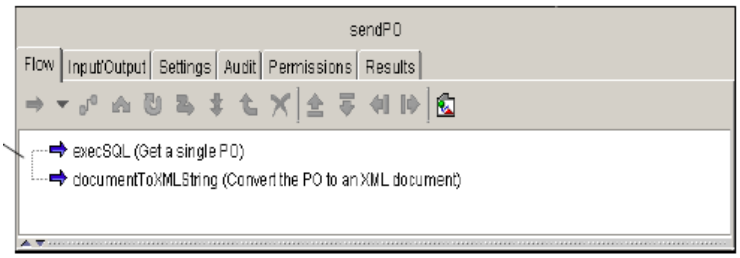
To build a flow service that submits a document to the Partner Manager

1. Use Designer to create a new flow service.

2. Insert the flow steps that your service will use to generate or retrieve the document that you want to submit to the Partner Manager. For example, if the document will originate from an ERP system such as Baan or PeopleSoft or from a database, build the invoke and data-transformation steps that will retrieve and generate the document at run time.

   The following example shows a flow service that extracts a single document from the database (in this case, a PO) and converts it to an XML document. (Although an XML document is generated in this example, you can submit any type of data to the Partner Manager. It does not have to be an XML document.)

   Example of Steps Used to Generate a Document

3. If you will use the Email or FTP transport to route the document to its destination, do the following:

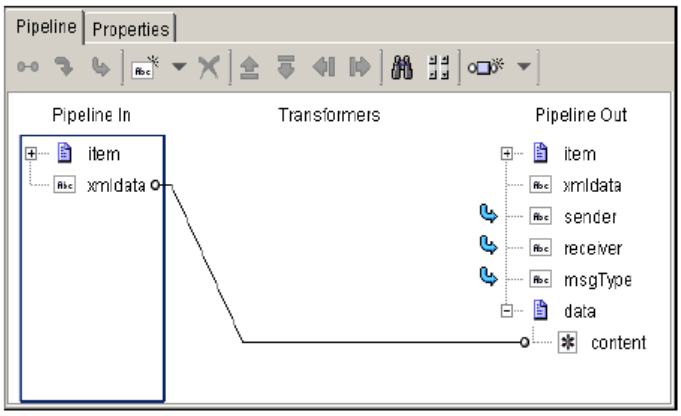
   a. Add a map step that creates an IS Document (IData object) called `data`.

   b. Within `data`, create an object called `content`.

   c. Map your document (which must be a String, or an object that contains a byte [ ] or an InputStream) to `data/content`.

   The following shows how the MAP step is used to map the XML document generated by documentToXMLNode from the preceding example, to *data/content*.

Example of a Document Mapped to *data/content*



4. Insert a step that invokes wm.PartnerMgr.gateway.transport.B2B:InboundProcess

5. Use the Pipeline Editor to set (or map) the following InboundProcess input values.

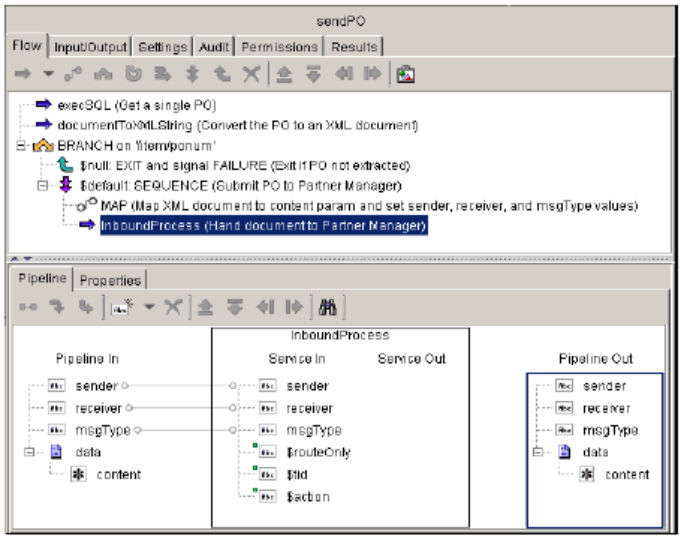
In this parameter…	Specify…
sender	The string that you are using to represent the sender. This value is what the Partner Manager will use to match the **Sender** value in its routing rules.
receiver	The string that you are using to represent the receiver of this document. This value is what the Partner Manager will use to match the **Receiver** value in its routing rules.
msgType	The string that you are using to represent the type of document you are submitting to the Partner Manager. This value is what the Partner Manager will use to match to the **Message Type**value in its routing rules.
$routeOnly	Optional. Whether you want the Partner Manager to log the document in its message store. Valid values are: ■ `true` Route the document without logging it in the message store. The *$routeOnly* parameter must exist in the pipeline for this to take effect. ■ `false` Log the document in the message store to the message store when it is routed.This is the default value. ■ `no value` Log the document in the message store to the message store when it is routed.
$tid	Optional. A string that specifies the transaction ID that you want Partner Manager to the document when it puts it in the message store. Used only when *$routeOnly* is false or doesn't exist in the pipeline.

In this parameter…	Specify…
	If you do not specify a transaction ID and *$routeOnly* is false or doesn't exist, the Partner Manager automatically generates a transaction ID for the transaction.
$action	Optional. One of the following action codes that sets the state of the document when it is put in the message store. Used only when *$routeOnly* is false or doesn't exist. Set the value to assign a state as follows: ■ `0` Create ■ `1` Execute ■ `2` Rollback ■ `3` Commit ■ `4` Confirm

Note:
You might want to drop any unused variables from the pipeline before invoking InboundProcess, especially if the document will be routed to another service on a remote server. This will prevent Partner Manager from unnecessarily sending data across the network.

The figure that follows shows a finished flow service that generates and submits a document to the Partner Manager. (In this example, the *sender*, *receiver*, and *msgType* parameters already exist in the pipeline, so they are implicitly mapped to those parameters in InboundProcess.)

Example of a flow service that submits a document to Partner Manager



Using the Message Store

What Is the Message Store?

When you submit a document to the Partner Manager, you can optionally instruct the Partner Manager to log a copy of that document in its *message store*. The message store is a facility that maintains a document of transactions that the Partner Manager processes. Each transaction document in the message store contains a copy of the document that was submitted to the Partner Manager, along with its routing parameters and an audit trail describing how the document was processed.

You use Integration Server Administrator to view the contents of the message store.

By viewing the message store, you can determine if and when the Partner Manager handled a particular document. You can also determine whether the Partner Manager processed a particular document successfully or returned an error.

Location of the Message Store

By default, the Partner Manager maintains the message store in the xtn.log file, which resides in the *Integration Server_directory* \packages\WmPartners\config directory on your webMethods Integration Server. However, you can optionally configure the Partner Manager to maintain this information in a JDBC-compliant database, such as Oracle 6.0 (or higher), Informix 7.0 (or higher), or MS SQL Server 6.50.201 (or higher). For information about how to configure your system to use a database for the message store, see "Using a Database as the Message Store" on page 112.

Transaction IDs Identify Documents in the Message Store

Individual documents in the message store are uniquely identified by their *transaction IDs (tid)*. A transaction ID is an arbitrary string that you can assign to the document before submitting it to the Partner Manager or you can allow the Partner Manager to assign to a document automatically. For information about assigning a transaction ID to a document, see the *$tid* parameter under "Submitting a Document to the Partner Manager" on page 107.

Logging a Document in the Message Store

When you submit a document to the Partner Manager, you specify whether or not you want that document recorded in the message store by setting the *$routeOnly* parameter. The Partner Manager only records documents whose *$routeOnly* parameter is set to "false" or when the *$routeOnly* parameter doesn't exist in the pipeline. If you submit a document with *$routeOnly* set to "true" (i.e., *$routeOnly* exists in the pipeline), the Partner Manager processes the document without logging it into the message store.

For additional information about setting the *$routeOnly* flag when submitting a document to the Partner Manager, see the *$routeOnly* parameter under "Submitting a Document to the Partner Manager" on page 107.

Note:

Once a document is written to the message store, the Partner Manager never removes it. The message store will grow continually unless you periodically purge the documents that are no longer needed. For information about deleting transactions from the message store, see "Deleting a Single Transaction from the Message Store" on page 119 and "Purging All Transactions from the Message Store" on page 119.

Configuring the Location of the Message Store

You can configure the Partner Manager to maintain the message store in a log file (xtn.log) on webMethods Integration Server or in a JDBC-compliant database that you have installed at your site.

- If you want the Partner Manager to maintain its message store in the xtn.log file, you do not need to perform any special configuration steps to set up the message store. The Partner Manager will automatically use xtn.log as the message store unless you specifically configure it to use a database.
- If you want the Partner Manager to maintain its message store in a JDBC-compliant database, you must configure the database server and webMethods Integration Server as described in the following sections.

Important:
To maintain the message store in a JDBC-compliant database, you must have the WmDB package installed and enabled on your Integration Server.

Using a Database as the Message Store

If you want to maintain the message store in a JDBC-compliant database that is installed at your site, you must take the following general steps to identify the location of that database. For more information about specific steps for Microsoft SQL Server 7 and Oracle 8i, see "Configuring the Message Store on Microsoft SQL Server 7.x" on page 113 and "Configuring the Message Store on Oracle 8i" on page 116.

Important:
Do *not* use a database with either the Microsoft or SUN built-in JDBC-ODBC bridge because these bridges can cause problems with webMethods Integration Server.

Note:
If Integration Server cannot establish a connection to the database for any reason, the message store will be automatically switched to the file system.

Configuring the database server

On the database server

1. Establish a user account that Partner Manager can use to access the database.

2. Have your database administrator create the following tables in the database. Make sure that the user account you created in the previous step has read and write access to these tables.

Table	Columns	Type	Length
WMTRANSACTION_Log	tid	VARCHAR	50
	creationdate	DATE	
	message	VARCHAR	255
WMTransactions	tid	VARCHAR	50
	sender	VARCHAR	50
	receiver	VARCHAR	50
	msgType	VARCHAR	50
	package	VARCHAR	50
	docURI	VARCHAR	50
	flow	VARCHAR	50
	lastError	VARCHAR	50
	state	INTEGER	
	wmTime	DATE	

Configuring webMethods Integration Server

On webMethods Integration Server

1. Install a pure JDBC driver (for example the weblogic.jdbc.mssqlserver4.Driver from WebLogic or a comparable driver from a vendor such as Merant or Imprise) that webMethods Integration Server can use to access the database server. (Make sure to add this driver to the server's classpath.)

2. Check that the WmDB package is installed and enabled on your Integration Server. Then use Integration Server Administrator to define an alias for the database server.

3. Add the following statement to the server.cnf file:

   watt.PartnerMgr.xtn.store=db

Configuring the Message Store on Microsoft SQL Server 7.x

Use the following procedures to maintain the Partner Manager's message store in a SQL Server 7.x database.

Configuring the SQL Server

To use an SQL 7.x database as the message store, configure the SQL Server as follows

1. Create a user account that the Partner Manager can use to access the database.
2. Create a user-defined data type named `Date` that has the data type "datetime."
3. Create the following tables and give the Partner Manager's account read/write access to them.

Table	Columns	Type	Length
WMTRANSACTION_Log	tid	VARCHAR	50
	creationdate	DATE	
	message	VARCHAR	255
WMTransactions	tid	VARCHAR	50
	sender	VARCHAR	50
	receiver	VARCHAR	50
	msgType	VARCHAR	50
	package	VARCHAR	50
	docURI	VARCHAR	50
	flow	VARCHAR	50
	lastError	VARCHAR	50
	state	INTEGER	
	wmTime	DATE	

Note:
If you grant the Partner Manager's user account permission to create tables, the Partner Manager will automatically create these tables for you the first time it connects to the database.

Configuring webMethods Integration Server

On webMethods Integration Server

1. Obtain a pure JDBC driver (for example, the weblogic.jdbc.mssqlserver4.Driver from WebLogic or a comparable driver from a vendor such as Merant or Imprise).

   Important:
   The Microsoft SQL Server 7.x ODBC driver is *not* compatible with JDBC.

2. Install the driver according to the vendor's installation instructions.

3. Edit the server.bat file (under Windows) or the server.sh file (under Unix) and add the JDBC driver to the classpath statement. (These files reside in the *Integration Server_directory* \bin directory.)

4. Start Integration Server Administrator and use the **Database** command on the Adapter menu to define an alias (a named set of connection parameters) called "transactions" for the SQL Server database. When you add this alias, specify the following information in the **Details** section of the New DB Alias screen. For more information about adding an alias for a database server, see the *WmDB User's Guide*.

In this field...	Specify...
Alias	The following string: `transactions` Type this string *exactly* as it appears above, using only lowercase letters.
DB URL	The URL for the SQL Server database. **Example:** `jdbc:weblogic:mssqlserver4:edi:1433.`
DB Username	The user ID that you created for the Partner Manager in step 1 of "Configuring the SQL Server" on page 114. This is the user ID that the Partner Manager will use to connect to the SQL Server database. **Note:** When the Partner Manager connects to the database, it attempts to create the message store tables if they do not already exist. Before using the Partner Manager with SQL Server for the first time, make sure that the tables listed in step 3 of "Configuring the SQL Server" on page 114 exist on the SQL server or that the user ID specified in **DB URL** is authorized to create tables.
DB Password	The password associated with the user account specified in **DB Username**.
Minimum Connections	The minimum number of connections to pool.

In this field...	Specify...
Maximum Connections	The maximum number of connections to have in the pool. When the maximum number of connections are open, then the next call to get a connection from the pool will wait until a connection is available (that is, re-added to the pool).
Expiration Time (ms)	The amount of time to wait before discarding a cached connection from the pool.
Loaded Drivers	The class name of the JDBC driver. Example: `weblogic.jdbc.mssqlserver4.Driver`

5. Edit the server configuration file to enable database logging. To do this, open *Integration Server_directory* \config\server.cnf with a text editor and add the following statement to it:

   watt.PartnerMgr.xtn.store=db

Configuring the Message Store on Oracle 8i

Use the following procedures to maintain the Partner Manager's message store in an Oracle 8i database.

Configuring the Oracle Database Server

To use an Oracle 8i database for the message store, configure the Oracle database server as follows

1. Create a user account that the Partner Manager can use to access the database.

2. Create the following tables and give the Partner Manager's account read/write access to them.

Table	Column	Type	Length
WMTRANSACTION_Log	tid	VARCHAR	50
	creationdate	DATE	
	message	VARCHAR	255
WMTransactions	tid	VARCHAR	50
	sender	VARCHAR	50
	receiver	VARCHAR	50

Table	Column	Type	Length
	msgType	VARCHAR	50
	package	VARCHAR	50
	docURI	VARCHAR	50
	flow	VARCHAR	50
	lastError	VARCHAR	50
	state	DECIMAL	
	wmTime	TIMESTAMP	

Note:
If you grant the Partner Manager's user account the permission to create tables, the Partner Manager will automatically create these tables for you the first time it connects to the database.

Configuring webMethods Integration Server

On webMethods Integration Server

1. Install the Oracle 8i thin driver classes111.zip file.

2. Edit the server.bat file (under Windows) or the server.sh file (under Unix) and add the JDBC driver to the classpath statement. (These files reside in the *Integration Server_directory* \bin directory.)

3. Start Integration Server Administrator and use the **Database** command on the Adapter menu to define an alias (a named set of connection parameters) called "transactions" for the Oracle database. When you add this alias, specify the following information in the **Details** section of the New DB Alias screen. For more information about adding an alias for a database server, see the *WmDB User's Guide*.

In this field...	Specify...
Alias	The following string: `transactions` Type this string *exactly* as it appears above, using only lowercase letters.
DB URL	The URL for the Oracle database, which is usually specified in the following format: `jdbc:oracle:thin:username/password@hostname:portnumber:DatabaseName`

In this field...	Specify...
	where: ■ *username* and *password* are the username and password you want to use to log on to the server. These are generally not needed, since the database framework will use the **DBUsername** and the **DBPassword** fields. ■ *hostname* is the host name of the Oracle server. ■ *portnumber* identifies the port on which the Oracle server listens for incoming requests. ■ *DatabaseName* is the name of the database.
DB Username	The user ID that you created for the Partner Manager in step 1 of "Configuring the Oracle Database Server" on page 116. This is the user ID that the Partner Manager will use to connect to the Oracle database. **Note:** When the Partner Manager connects to the database, it attempts to create the message store tables if they do not exist. Before using the Partner Manager with Oracle 8i for the first time, make sure that the tables listed in step 2 of "Configuring the Oracle Database Server" on page 116 exist on the Oracle server or that the user ID specified in **DB URL** is authorized to create tables.
DB Password	The password associated with the user account specified in **DB Username**.
Minimum Connections	The minimum number of connections to pool.
Maximum Connections	The maximum number of connections to have in the pool. When the maximum number of connections are open, then the next call to get a connection from the pool will wait until a connection is available (i.e., re-added to the pool).
Expiration Time (ms)	The amount of time to wait before discarding a cached connection from the pool.
Loaded Drivers	The class name of the JDBC driver. Example: `oracle.jdbc.driver.OracleDriver`

4. Edit the server configuration file to enable database logging. To do this, open *Integration Server_directory* \config\server.cnf with a text editor and add the following statement to it:

   watt.PartnerMgr.xtn.store=db

Viewing the Message Store

Use the following procedure to view a list of the documents that the Partner Manager has logged into the message store and to examine the details (including the sender, receiver, message type, and error information) for a particular document.

To view the transactions in the message store

1. Start Integration Server Administrator.
2. On the Adapters menu, click **Routing**.
3. Click **Transactions**. The contents of the message store appears. Documents are identified by transaction ID.
4. If you want to sort the list, click the up or down arrow icons in the column headings.
5. If you want to view details about a particular document, click its transaction ID (in the TID column).
6. If you want to view the body of the document (in addition to additional transaction information that was present at run time), click one of the following options in the **View** field:
   - To view the transaction information in webMethods standard HTML-encoded format, click **As XML** or **As Values**.
   - To view the transaction information in a special HTML format that displays the body of the document in a text box, click **As HTML**.

Deleting a Single Transaction from the Message Store

Use the following procedure to delete a transaction from the message store.

To delete a transaction from the message store

1. Start Integration Server Administrator.
2. On the Adapters menu, click **Routing**.
3. Click **Transactions**. Locate the transaction that you want to delete.
4. Click the **Delete** icon for the transaction.

Purging All Transactions from the Message Store

There are two procedures to periodically purge the message store. The following procedure, which is recommended, uses the wm.PartnerMgr.xtn.Sweeper:sweepTRX service to periodically purge the message store of transactions.

Purging All Transactions from the Message Store Periodically

To purge all transactions from the message store periodically

1. In Designer, create a flow service that invokes the wm.PartnerMgr.xtn.Sweeper:sweepTRX service.

2. Pass the following parameters to the sweepTRX service.

For this parameter...	Specify
onState	The transaction state
elapsedTime	The number of minutes that the transaction has been in onState
maxTrxCount	The maximum number of transactions to delete each time that sweepTRX is invoked

   When the sweepTRX service is invoked, the System finalizer is invoked and the garbage collector is invoked.

3. Schedule the flow service to run periodically to purge the message store.

Example

If you want to periodically purge all transactions which have been in the Confirmed state for longer than 30 minutes, you can create a service called purgeConfirmedTRX with those state and time parameters. You also specify how many transactions every invocation of the service will purge. This is so you can control the pace of your maintenance jobs without increasing the amount of overhead processing.

The following procedure deletes all transactions manually from the message store. Use this procedure only to delete a small number of documents. Otherwise, server performance can degrade substantially.

Purging All Transactions from the Message Store Manually

To purge all transactions from the message store manually

1. Start Integration Server Administrator.

2. On the Adapters menu, click **Routing**.

3. Click **Transactions**.

4. In the Delete column heading, click **Delete All**.

D Troubleshooting

Cannot Send Envelopes to MarketSite

Problem

You cannot send envelopes to MarketSite. When you test sending envelopes to MarketSite using the Test Send/Receive utility, the HTML representation of your pipeline shows error messages with Java exceptions. When you send envelopes in your production environment, you receive reply envelopes containing errors such as MarketSite cannot verify your user ID or password or your TPID is unknown.

Causes

- Integration Server cannot establish a network connection to MarketSite.
- Your firewall or MarketSite's firewall is preventing the connection from being established.
- Your alias definition for MarketSite is incorrect (for example, the URL you specified as the destination for envelopes you send to MarketSite is invalid).
- At error occurred at MarketSite's end.

Resolutions

- Contact your network administrator and make sure you can establish a TCP/IP connection to MarketSite.
- Contact your network administrator and make sure your firewall allows you to open an outbound communication that reaches the MarketSite host on the specified port. Contact the MarketSite network administrator and make sure MarketSite's firewall allows you to open a connection at the specified host and port.
- Correct your alias definition for MarketSite. Check the URL you specified as the destination for envelopes you send to MarketSite; make sure you are using the correct protocol (HTTP or HTTPS) and have specified the correct host and port. Also check the rest of the URL for accuracy. Make sure you spelled the alias correctly in all relevant C1 OnRamp services.
- Open the C1 OnRamp envelope log and view the contents of the reply envelope to determine the problem.

Not Receiving Envelopes from MarketSite

Problem

You are not receiving envelopes from MarketSite.

Causes

- MarketSite is using the wrong URL as the destination for envelopes you receive.

- MarketSite might be using the wrong user ID and password to access the Integration Server.
- Your firewall or MarketSite's firewall is preventing the connection from being established.

Resolutions

- Check whether MarketSite is using the wrong URL as the destination for envelopes you receive. Set the Integration Server log to 9 or 10. Ask MarketSite to send you an envelope. Open the Integration Serverlog; if you find an error message indicating an incoming request that refers to a resource such as /xcc, MarketSite is using the wrong URL (for example, http[s]://*server*:*port*:/xcc). Ask the MarketSite network administrator to change your destination URL to the correct URL (for example, http[s]://*server*:*port*/invoke/pub.marketconnect.transport/receiveEnvelope).
- Contact the MarketSite network administrator to make sure MarketSite is using the correct user ID and password to access the Integration Server.
- Contact the MarketSite network administrator and make sure MarketSite's firewall allows it to open an outbound communication that reaches Integration Server on the specified port. Contact your network administrator and make sure your firewall allows MarketSite to open a connection at the specified host and port.

MarketSite Acknowledges a Sent Envelope But Recipient Does Not Receive It

Problem

MarketSite returns a reply envelope containing an acknowledgement for an envelope you sent, but the recipient never receives the envelope.

Cause

MarketSite sometimes redirects envelopes to its lost and found directory.

Resolution

Work with the MarketSite network administrator to fix this problem.