webMethods ebXML Module Installation and User's Guide

Version 7.1 SP1

January 2017

WEBMETHODS

This document applies to webMethods ebXML Module 7.1 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: ESTD-EBXML-IUG-71 SP1-20200827

Table of Contents

About this Guide **5**
Document Conventions 6
Online Information and Support 7
Data Protection 7

1 Concepts **9**
Introduction 10
webMethods ebXML Module Features 11
webMethods ebXML Module Architecture 12
Getting Started 15

2 Installing, Upgrading, and Uninstalling webMethods ebXML Module **17**
Overview 18
Requirements 18
Installing ebXML Module 7.1 SP1 18
Installing the ebXML Module Samples and Security Packages 19
Upgrading to ebXML Module 7.1 SP1 19
Uninstalling ebXML Module 7.1 SP1 22

3 Configuring webMethods ebXML Module **23**
Configuring ebXML Module for Message Processing 24
Configuring the Trading Networks Database 31
Improving Performance 32

4 Working with CPAs, TN Document Types, and Processing Rules **33**
What Is a Trading Partner Profile? 34
What Is a Trading Partner Agreement (TPA)? 34
About Collaboration Protocol Agreements (CPA) 35
Exchanging Business Documents: User Scenarios 36
Generating Trading Partner Profiles and a TPA from a CPA 38
Defining Your Enterprise Profile 38
Defining Your Trading Partners' Profiles 39
Manually Creating a TPA 41
Default TN Document Types Used by ebXML Module 43
Defining Processing Rules 44

5 Working with ebXML Messages **47**
Supported Communication Protocols 48
Initiating the ebXML Module Handshake 49
Sending ebXML Messages 53
Receiving ebXML Messages 55
Checking the Status of ebXML Messages 57

Viewing Transactions....59

6 ebXML Module Messaging Features....61
Reliable Messaging....62
Message Ordering....64
Multi-hop Messaging....66
Large Business Document Handling....70
Payload Compression....71
MIME Encoding Payloads....72
Message Exchange Using a Proxy Server....74
Using the ebXML Module in a Clustered Environment....74

7 ebXML Module Security Features....77
Overview....78
Configuring Certificates for Secure Messaging....79
Configuring XML Signature Support....79
Configuring S/MIME Support....82
Combining Compression and S/MIME Support....88

A Built-In Services....91
Summary of Elements....91
CPA Folder (wm.ip.ebxml.cpa)....92
MSH Folder (wm.ip.ebxml.MSH)....94
TN Folder (wm.ip.ebxml.TN)....103
util Folder (wm.ip.ebxml.util)....105

B Trading Partner Agreement Parameters Version 1. 0....107
Overview....107
TPA Parameters for ebXML Message Service Version 1.0 Specification....107

C Trading Partner Agreement Parameters Version 2. 0....119
Overview....119
TPA Parameters for ebXML Message Service Version 2.0 Specification....119

About this Guide

- Document Conventions 6
- Online Information and Support 7
- Data Protection 7

This guide describes the webMethods ebXML Module and how to implement the ebXML Message Service protocol.

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

- webMethods Integration Server and Integration Server Administrator, and understand the concepts and procedures described in the *webMethods Integration Server Administrator's Guide*.
- Software AG Designer and Developer, and understand the concepts and procedures described in the *Software AG Designer Online Help* and the *webmethods Developer User's Guide*.
- webMethods Trading Networks, and understand the concepts and procedures described in the various webMethods Trading Networks guides.
- My webMethods Server and its interface My webMethods, and understand the concepts and procedures described in the *Administering My webMethods Server*.
- ebXML terminology. For more information, see http://www.ebxml.org/specs/.

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

- Introduction 10
- webMethods ebXML Module Features 11
- webMethods ebXML Module Architecture 12
- Getting Started 15

- “Introduction” on page 10
- “ webMethods ebXML Module Features” on page 11
- “ webMethods ebXML Module Architecture” on page 12
- “Getting Started” on page 15

Introduction

webMethods ebXML Module is based on the electronic business XML (ebXML) specification that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, organizations have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms, and define and register business processes.

webMethods ebXML Module 7.1 SP1 is an implementation of the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. webMethods ebXML Module runs on webMethods Integration Server and webMethods Trading Networks. webMethods Trading Networks enables enterprises to link with other organizations to enable a secure business-to-business trading network.

The *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification* provide a reliable and secure delivery infrastructure for exchanging electronic business documents of any format that is independent of your communications protocol.

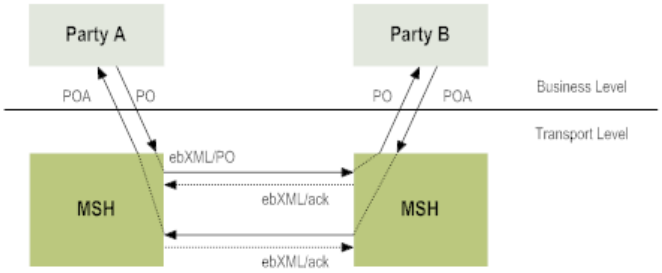
A typical example of an ebXML exchange might be two trading partners (Party A and Party B) exchanging a purchase order (PO) and purchase order acknowledgment (POA) via the ebXML Message Service. This illustration refers to a message receiver or sender at the transport level as a Message Service Handler (MSH).

Party A submits a PO to its MSH. The MSH for Party A wraps the PO within an ebXML envelope and sends it to Party B. The MSH for Party B receives the message and sends back a transport level acknowledgment to the MSH for Party A. Then, the MSH for Party B forwards the PO to Party B.

After processing the PO, Party B creates a POA and submits it to its MSH. The MSH for Party B wraps the POA within an ebXML envelope and sends it over to Party A. The MSH for Party A receives the message and sends back a transport level acknowledgment to the MSH for Party B. Then, the MSH for Party A forwards the POA to Party A.

As illustrated in the following figure, a clear separation exists between the business level and the transport level. The ebXML Message Service is involved in transport-level communication.

Figure 1. Example exchange between two trading partners using the ebXML Message Service Protocol



webMethods ebXML Module Features

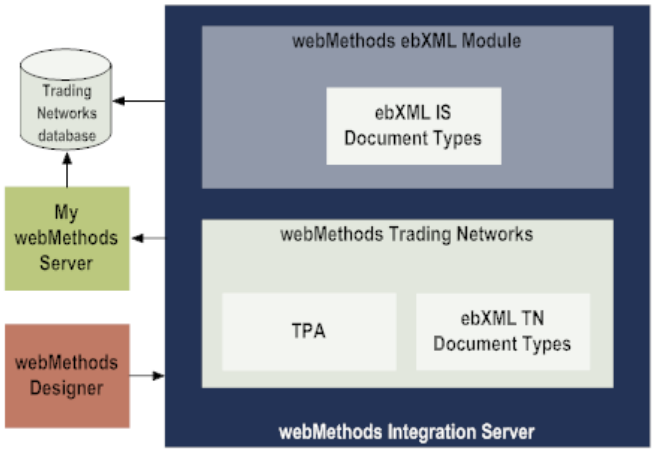
webMethods ebXML Module provides the basic implementation of all major features in the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. The webMethods ebXML Module is built on the webMethods Integration Server and webMethods Trading Networks components and provides the following features:

- **Header Processing.** When creating the ebXML messages, ebXML Module creates the header elements based on information from the following sources:
  - Application that is passed through the Message Service Interface
  - Information that governs the message received from the Collaboration Protocol Agreement
  - Generated information, such as digital signature, timestamps, and unique identifiers
- **Header Parsing.** ebXML Module supports extracting or transforming the information from the ebXML Header element into a form that the Message Service Handler (MSH) requires for processing the header.
- **Message Packaging.** ebXML Module supports the compilation of an ebXML Message envelope, comprising an ebXML envelope, header, and payload, into the SOAP Messages with Attachments container. For more information about SOAP messages, see the *SOAP Developer's Guide*.
- **Security Features.** ebXML Module provides digital signature and verification, encryption, and authentication. Other components of the MSH can use these services, including the header processing and header parsing components.
- **Error Handling.** ebXML Module reports errors encountered during MSH processing or processing of a message.
- **Synchronous and Asynchronous Messaging.** With ebXML Module, you can employ either synchronous or asynchronous communications to better meet the time requirements of your trading partner transactions. For those messages that require immediate responses over the same Internet connections, you can use synchronous communications. For transactions with longer duration, you can use asynchronous communications.
- **Message Persistence.** ebXML Module always saves all incoming and outgoing messages in the Trading Networks database.

- **Reliable Messaging.** The Reliable Messaging feature defines an interoperable protocol for two MSHs to reliably exchange messages. That is, when an MSH sends a message, the recipient MSH receives the message *once and only once*. Reliable Messaging also handles the delivery and acknowledgment of ebXML Messages. The feature includes message handling for persistence, retry, error, notification, duplicate detection, and acknowledgment of messages that require reliable messaging. Reliable Messaging is achieved when a receiver MSH responds to a message with an acknowledgment. For more information, see "HTTP/S Support" on page 48.
- **Message Ordering.** The Message Ordering feature allows the presenting of messages to the receiver in a specific order. The receiver MSH forwards the messages in the same specific order to the receiver application for processing the messages. The Message Ordering feature requires the use of the Reliable Messaging feature. For more information, see "Message Ordering" on page 64.
- **Multi-Hop Messaging.** The Multi-Hop Messaging feature allows messages to be sent from one MSH to another MSH via one or more intermediate parties. This feature allows the next destination of the message to be an intermediary MSH other than the receiver MSH that the original sending MSH has identified. For more information, see "Multi-hop Messaging" on page 66.
- **Large Business Document Handling.** The Large Business Document Handling allows ebXML Module to send, receive, and process large ebXML messages. For more information, see "Large Business Document Handling" on page 70.
- **Payload Compression.** The Payload Compression feature allows ebXML Module to compress and uncompress any type of payload in ebXML messages, especially for large business documents. For more information, see "Payload Compression" on page 71.
- **HTTP/S and SMTP support.** ebXML Module supports HTTPS and SMTP communication protocols along with the HTTP communication protocol. For more information, see "Supported Communication Protocols" on page 48.

webMethods ebXML Module Architecture

ebXML Module exists in conjunction with webMethods Trading Networks and is deployed as a webMethods Integration Server package named WmebXML. This package contains the common components and interfaces used across the ebXML implementation. Most of the components that this document describes belong to this package. The following shows the architecture of ebXML Module:



Component	Description
webMethods ebXML Module	webMethods ebXML Module supports secure business-to-business communications based on the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. ebXML Module contains the core WmebXML package that includes the services for implementing the ebXML functionality. It creates, parses, and validates the ebXML messages as specified in the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. During design time, to transmit the message from the sender to the receiver, ebXML Module requires sender, receiver, and TPA information, along with the payload information and any processing information. At run time, ebXML Module receives a business document from a back-end system or trading partner. It invokes a Trading Networks service to: ■ Obtain the input information. ■ Create the outbound message. ■ Transport the outbound message.
webMethods Trading Networks	Trading Networks enables your enterprise to link with trading partners with whom you want to exchange business documents, thereby forming a business-to-business trading network. During design time, you define your trading partner profiles using My webMethods. The profiles contain the information that Trading Networks requires to exchange business documents with your trading partners. In addition to defining trading partner profiles during design time, you also create and customize the TPAs using the Trading Networks Console. You can create the TPAs and the trading partner profiles manually, or you can import them using the CPA. The TN document types are

Component	Description
	automatically created when you load the WmebXML package, and you can view them using the Trading Networks Console.
webMethods Trading Networks (continued)	At run time, ebXML Module uses Trading Networks services and TN document types to: ■ Recognize ebXML messages it receives. ■ Create BizDocEnvelopes. ■ Save BizDocEnvelopes to the Trading Networks database. ebXML Module uses the trading partner profiles in Trading Networks to determine, for example, the methods by which to send business documents to its trading partners. ebXML Module uses TPAs in Trading Networks to determine information such as whether an outbound business document should be signed or whether the Service Header of the business document (along with any attachments) should be encrypted. For more information about Trading Networks, trading partner profiles, TN document types, and TPAs, see the *webMethods Trading Networks Administrator's Guide* for your release. You can also find information about trading partner profiles in "About Collaboration Protocol Agreements (CPA)" on page 35 and information about TPAs in "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.
Trading NetworksDatabase	Trading Networks saves trading partner profiles, TN document types, trading partner profiles, and TPA information in its database and retrieves this information when needed. It can also save information about the documents (that is, ebXML messages) that it processes, including the content of the documents.
My webMethods Server	My webMethods Server is the underlying server that manages the My webMethods user interface. During design time, you use My webMethods to create trading partner profiles. At run time, you can use My webMethods to view the message transaction details. For more information about My webMethods Server and its My webMethods interface, see *Working with My webMethods* and the *webMethods Integration Server Administrator's Guide* for your release.
webMethods Integration Server	Integration Server contains the IS document types and services that you use when creating your ebXML Module services.
Software AG Designer	At design time, use Designer to edit ebXML Module services and create customized solutions. Designer also provides tools for testing and debugging the solutions you create.

Getting Started

The following procedure outlines the steps you take to get started with webMethods ebXML Module.

1. Install webMethods ebXML Module. For more information about this step, see "Installing ebXML Module 7.1 SP1" on page 18.

2. Configure webMethods ebXML Module. After completing the installation, set the configuration properties. For details, see "Configuring ebXML Module for Message Processing" on page 24.

3. Configure the Trading Networks Database. For connecting to the Trading Networks database, you create a JDBC pool alias. When you create the JDBC pool alias, specify the name of the connection database that is required to connect to the Trading Networks database. For more information about creating a JDBC pool alias and associating it to a JDBC functional alias, see "Configuring the Trading Networks Database" on page 31.

4. Create trading partner profiles and Trading Partner Agreements. To use ebXML Module, you should have a formal agreement, called a Trading Partner Agreement (TPA), with your trading partners. For more information about creating trading partner profiles and Trading Partner Agreements, see "Working with CPAs, TN Document Types, and Processing Rules" on page 33.

5. Send and receive messages. After creating the trading partner profile and Trading Partner Agreement, you are ready to send and receive the messages, with payloads, to your trading partners. For more information about the parameters to be set for sending and receiving different messages, see "Working with ebXML Messages" on page 47.

6. View the transactions. After sending and receiving messages, you can view the status of the messages and the list of transactions with a specific partner in My webMethods. For more information about viewing the ebXML transactions, see "Viewing Transactions" on page 59.

2 Installing, Upgrading, and Uninstalling webMethods ebXML Module

- Overview 18
- Requirements 18
- Installing ebXML Module 7.1 SP1 18
- Installing the ebXML Module Samples and Security Packages 19
- Upgrading to ebXML Module 7.1 SP1 19
- Uninstalling ebXML Module 7.1 SP1 22

- "Overview" on page 18
- "Requirements" on page 18
- "Installing ebXML Module 7.1 SP1" on page 18
- "Installing the ebXML Module Samples and Security Packages" on page 19
- "Upgrading to ebXML Module 7.1 SP1" on page 19
- "Uninstalling ebXML Module 7.1 SP1" on page 22

Overview

This chapter explains how to use the Software AG Installer and Uninstaller wizards to install, upgrade, and uninstall webMethods ebXML Module 7.1. For complete information about other installation methods or installing other Software AG Installer products, see the *Installing webMethods Products On Premises* for your release.

Requirements

For a list of the operating systems and webMethods products supported by ebXML Module 7.1 SP1, see the *webMethods eStandards Modules System Requirements* .

ebXML Module 7.1 SP1 has no hardware requirements beyond those of its host Integration Server.

Installing ebXML Module 7.1 SP1

To install ebXML Module 7.1 SP1

1. If you are installing the module on an existing Integration Server, shut down the Integration Server.

2. Download the Software AG Installer from the Empower Product Support website at https://empower.softwareag.com.

3. Start the Software AG Installer wizard.

   - Choose the webMethods release that includes the Integration Server on which to install the module. For example, if you want to install the module on Integration Server 7.1, choose the 7.1 release.
   - If you are installing on an existing Integration Server, specify the *Software AG_directory* that contains the host Integration Server. If you are installing both the host Integration Server and the module, specify the installation directory to use. Installer will install the module in the *Integration Server_directory* \packages directory.

- In the product selection list, select **eStandards > webMethods ebXML Module7.1 SP1 > Program Files**. You can also choose to install documentation and any required products indicated in the *webMethods eStandards Modules System Requirements* .

4. After installation is complete, start the host Integration Server.

5. Verify the ebXML Module installation as follows:

   a. On the Integration Server host machine, make sure the *Integration Server_directory* \packages\WmebXML directory exists. If it does not, uninstall and re-install the ebXML Module.

   b. Make sure the WmebXML.jar and WmebXML601.jar files exist in the *Integration Server_directory* \packages\WmTN\jars directory.

   Note:
   The WmebXML601.jar file is for backward compatibility with ebXML Module 6.0.1 document types.

   c. Start Integration Server Administrator and go to the **Packages > Management** page. Make sure the WmebXML package is loaded and enabled. If it is not, open the error log file and check for any dependencies. Fix the dependencies, save the package, and reload the package.

Installing the ebXML Module Samples and Security Packages

ebXML Module 7.1 SP1 offers a samples package named WmebXMLSample and a security package named WmebXMLSecurity. The security package contains the services you need to implement XML encryption features.

These packages are not installed with ebXML Module 7.1 SP1. To download the packages, the installation procedure, and the *webMethods ebXML Module Sample Package User's Guide*, go to webMethods Community page on the Software AG Developer Community at http://communities.softwareag.com/ecosystem/communities/public/Developer/webmethods/products/esb/ and see the Code Samples.

Upgrading to ebXML Module 7.1 SP1

You can upgrade to ebXML Module 7.1 SP1 from ebXML Module 6.0.1 or from ebXML Module 7.1.

Upgrading from ebXML Module 6.0.1

To upgrade and migrate from ebXML Module 6.0.1 to ebXML Module 7.1 SP1

1. Back up your existing ebXML Module 6.0.1 installation and all custom packages that are used by ebXML Module 6.0.1.

2. Export the ebXML Module 6.0.1 Trading Networks profiles, custom ebXML Document Types, custom processing rules, and TPAs, from Trading Networks 6.5. For instructions, see the *webMethods Trading Networks User's Guide* for your release.

3. If you are running earlier, unsupported version of Integration Server and Trading Networks, upgrade to supported versions. For instructions, see the *Upgrading Software AG Products* for your release.

4. Start Integration Server.

5. Note the actions defined for each ebXML Module 6.0.1 processing rules. Using Trading Networks, remove all ebXML Module 6.0.1 processing rules manually.

6. Copy the ebXML Module 6.0.1 custom packages and all other packages that depend on the ebXML Module 6.0.1 from the backup directory to the *Integration Server_directory* \packages directory.

   Important:
   Do not copy the WmebXML package or the WmebXMLSample package from ebXML Module 6.0.1, because these packages will not work with the Integration Server versions that ebXML Module 7.1 SP1 supports.

7. Shut down the Integration Server.

8. Install webMethods ebXML Module 7.1 SP1. For instructions, see "Installing ebXML Module 7.1 SP1" on page 18.

9. Update the ebXML Module 7.1 SP1 config.cnf file in the WmebXML\config directory with applicable settings from the equivalent ebXML Module 6.0.1 file. Some of the properties in the ebXML Module 6.0.1 configuration file are not applicable for ebXML Module 7.1 SP1. For information on the revised properties, see the *webMethods ebXML Module*7.1 SP1*readme*.

10. Start the Integration Server.

11. Verify the migration of the Trading Networks database from Trading Networks 6.5 to the supported version. To do so, start the Trading Networks Console and make sure the ebXML TN document types, ebXML processing rules (including the custom processing rules) and TPAs were migrated successfully. If the migration was not successful, import the ebXML Trading Networks profiles, document types, processing rules, and TPAs from the export file created in step 2 into Trading Networks. For instructions, see the *webMethods Trading Networks Administrator's Guide* for your release.

12. Verify the ebXML Module 7.1 installation, as described in "Installing ebXML Module 7.1 SP1" on page 18.

13. Update the following, if applicable:

- Custom services that were dependent on theebXML Module 6.0.1 built-in services with the ebXML Module 7.1 SP1 built-in services.
- ebXML Module 7.1 processing rules with the actions, as specified in step 5.
- TN document types that were dependent on the ebXML Module 6.0.1 TN document types (for example, custom services, process models) with ebXML Module 7.1 TN document types.
- ebXML Module 6.0.1 TPAs to ebXML Module 7.1 TPAs by invoking the wm.ip.ebxml.util:migrateTPA service. For information about the parameters for this service, see "Built-In Services" on page 91.

14. Make sure the ebXML Module 7.1 SP1 can successfully exchange a ping message with its Trading Partners. For instructions, see "Initiating the ebXML Module Handshake" on page 49.

15. Update ebXML Module 6.0.1 services that use the ping, route, send, and status services to use the services provided in ebXML Module 7.1, as follows:

Update your ebXML Module 6.0.1 services that use....	To use these ebXML Module 7.1 services
ping	pingUsingTPA
route	routeUsingTPA
send	sendUsingTPA
status	statusUsingTPA

16. For information about ebXML Module 7.1 SP1 services, see "Built-In Services" on page 91.

17. Verify the services that theebXML Module provides by exchanging the ebXML messages with the trading partners. For instructions, see "Working with ebXML Messages" on page 47.

Note:
The ebXML Module can exchange messages with any MSH that uses previous versions of the ebXML Module. This includes exchanging messages using the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

Upgrading from ebXML Module 7.1

To upgrade and migrate from ebXML Module 7.1 to ebXML Module 7.1 SP1

1. Back up your existing ebXML Module 7.1 installation and all custom packages.

2. If you have customized the configuration file in *Integration Server_directory* \packages\WmebXML\config\config.cnf, back up this file.

3. Export the ebXML Module 7.1 Trading Networks profiles, custom ebXML Document Types, custom processing rules, and TPAs, from Trading Networks as a backup if needed. For instructions, see the *webMethods Trading Networks User's Guide* for your release. These elements can be used with ebXML Module 7.1 SP1.

4. Shut down Integration Server.

5. Uninstall the ebXML Module 7.1 installation.

6. Delete the *Integration Server_directory* \packages\WmebXML folder.

7. Install ebXML Module 7.1 SP1 as described in "Installing ebXML Module 7.1 SP1" on page 18.

8. Replace the configuration file in *Integration Server_directory* \packages\WmebXML\config\ with the config.cnf file you backed up in step 2\.

9. Restart Integration Server.

10. Verify the services that ebXML Module provides by exchanging ebXML messages with the trading partners. For instructions, see "Working with ebXML Messages" on page 47.

Uninstalling ebXML Module 7.1 SP1

To uninstall ebXML Module 7.1 SP1

1. Shut down the Integration Server that hosts ebXML Module 7.1 SP1.

2. Start the Software AG Uninstaller, selecting the *Software AG_directory* that contains the host Integration Server. In the product selection list, select **eStandards > webMethods ebXML Module 7.1 SP1 > Program Files** and any other products and items you want to uninstall.

   Note:
   Processing rules are not deleted automatically; instead, you must delete the rules manually. For instructions, see the *webMethods Trading Networks User's Guide* for your release.

3. Restart the host Integration Server.

4. Uninstaller removes all ebXML Module 7.1 SP1 related files that were installed into the *Software AG_directory* \packages directory. However, Uninstaller does not delete files created after you installed the module (for example, user-created or configuration files), nor does it delete the module directory structure. You can go to the *Software AG_directory* \packages directory and delete the WmebXML directory.

3 Configuring webMethods ebXML Module

- Configuring ebXML Module for Message Processing 24
- Configuring the Trading Networks Database 31
- Improving Performance 32

- "Configuring ebXML Module for Message Processing" on page 24
- "Configuring the Trading Networks Database" on page 31
- "Improving Performance" on page 32

Configuring ebXML Module for Message Processing

You can control many functions of the ebXML Module by specifying values for the properties in the configuration file, config.cnf. This file contains the properties for caching, message processing, payload size, message timestamps, and other settings.

To edit the configuration properties

1. Open the file config.cnf, available at *Integration Server_directory* \packages\WmebXML\config.
2. Specify the values for the properties as defined below.
3. When you are finished, save the file.
4. Reload the WmebXML package for the changes to take effect.
5. The following table lists the definitions of the configuration properties in the ebXML Module config.cnf file.

Configuration Property	Definition
wm.ebxml.autoAckSend	Enables or disables the sending of automatic acknowledgments when messages are processed successfully. ■ Set this property to `false` to disable the sending of automatic acknowledgments when messages are processed successfully. You can explicitly send acknowledgments using the wm.ip.ebxml.MSH:sendAck service. ■ Set this property to `true` to enable ebXML Module to automatically send acknowledgments to the message sender if the messages were processed successfully. This is the default setting. Automatic acknowledgments can also be controlled using the TPA parameter AutoAckSend. For more information, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

Configuration Property	Definition
wm.ebxml.bypassRoutingRule	Specifies whether you want to bypass using Trading Networks processing rules to enhance the performance of the ebXML Module. ■ Set this property to `true` to bypass the Trading Networks processing rules. This is the default setting. ■ Set this property to `false` to use the Trading Networks processing rules. For information on bypassing the routing rule and enhancing the performance, see "Bypassing the Use of Trading Networks Processing Rules" on page 32.
wm.ebxml.caching	Specifies whether to enable or disable memory caching. ■ Set this property to `true` to enable caching for better performance. This setting is recommended only for production servers. ■ Set this property to `false` to disable caching. This is the default setting.
wm.ebxml.cachingScheme	Specifies the caching mechanism that is used. **Note:** These values are considered only if the wm.ebxml.caching property is set to `true`. ■ Set this property to `Local` cache, which is an in-process extension of java.util.Map. Its contents are not visible outside of the JVM in which it runs. This is the default setting. ■ Set this property to `Distributed` cache, for all the cache members to have access to the entire cache, but no single member actually retains a complete copy of the cache contents. ■ Set this property to `Replicated` cache, where each and every application connected to the cache (cache members) retains its own copy of the entire contents of the cache. Any change in the value of this property requires you to restart the Integration Server for the changes to take effect.

Configuration Property	Definition
	For more details on cache schemes, see the *webMethods Integration Server Administrator's Guide* for your release.
wm.ebxml.defaultDoctypeOnly	Specifies whether to bypass using Trading Networks document types to recognize an ebXML message. ■ Set this property to `true` to bypass using TN document types to recognize ebXML messages. **Note:** You cannot use this setting if you have customized ebXML document types. ■ Set this property to `false` to use TN document types to recognize ebXML messages. This is the default setting.
wm.ebxml.defaultDocType.conversationID	Specifies how to map the conversation ID of the default document type. ■ Set this property to `true` to map the ebXML conversation ID to the ebXML default document type's conversationID. ■ Set this property to `false` to leave the ebXML default document type's conversation ID empty. This is the default setting.
wm.ebxml.docType.groupID	Specifies how to set the group ID for a message that matches an ebXML document type. ■ Set this property to `conversationID` if you want the group ID to match the conversation ID of the message. This is the default setting. ■ Set this property to `cpaID` if you want the group ID to match the CPA ID of the ebXML message.
wm.ebxml.getAgreementIDSvc	Specifies the qualified name of the service that returns the Agreement ID for the TPA. ebXML Module uses the *From*, *To*, and *CPAId* information specified as inputs to the wm.ip.ebxml.MSH services or available in the MessageHeader section of an ebXML message to retrieve a TPA from Trading Networks. If the CPA ID does not directly translate to the Agreement ID of the TPA, a custom service must be implemented to provide the Agreement ID for the TPA to ebXML Module.

Configuration Property	Definition
	The input and output for the service is defined by the specification, wm.ip.ebxml.rec:getAgreementID, while the implementation details are left to the implementer of the solution. ■ If the getAgreementIDSvc property is not defined, the behavior is unchanged (that is, the CPA ID is interpreted as the agreement ID of the TPA). ■ If this property is defined, but the specified service does not exist or the service fails during invocation, ebXML Module processing fails. An error is logged in the server logs to report the reason the service could not be invoked. ■ If this property is defined with a valid service that executes without failing, but does not retrieve an agreement ID (null or empty string) for the TPA, then ebXML Module logs an error in the server logs and the processing fails. The sendUsingTPA and similar services return the same error as above in the *errorMsg* structure. ■ If a non-null and non-empty agreement ID is returned, then processing and logging occurs as normal from there on.
wm.ebxml.message.useMultipartAlways	Specifies the format in which ebXML Module should generate messages when the messages do not contain payload(s) or attachments (for example, Ping or Status Request messages). ■ Set this property to `true` to generate messages without payload(s) as MIME multipart messages. Use this setting if you are using a multi-hop scenario where the sender MSH, intermediate MSH, or receiver MSH are running ebXML Module 6.0.1. ■ Set this property to `false` to generate messages without payload(s) as SOAP-formatted XML documents. This is the default setting.
wm.ebxml.oldContentIDFormat	Indicates how ebXML Module formats the content ID and reference URI. ■ Set this property to `true` to have ebXML Module use the format for the content ID without angle brackets and reference.

Configuration Property	Definition
	Note: Setting this property to `true` implements a content ID format and reference URI that does not comply with the *ebXML Message Service Version 2.0 Specification*. ■ Set this property to `false` to have ebXML Module follow the format as per the *ebXML Message Service Version 2.0 Specification*. This is the default setting.
wm.ebxml.payloadProcessSvc	Specifies the qualified name of the service to process all payloads. You must provide an implementation of the service that has the signature given by the IS document type specification wm.ip.ebxml.rec:payloadProcess for ebMS 1.0, and wm.ip.ebxml.rec:payloadProcess_v2 for ebMS 2.0. By default, the ebXML Module will invoke a webMethods service referenced by the *service element* in the ebXML envelope if the service type="webMethods".
wm.ebxml.StoreUnprocessed	Specifies whether you want to save the unprocessed ebXML message to the Trading Networks database before the decryption or decompression process. ■ Set this property to `true` to have the ebXML Module save the unprocessed ebXML message before decrypting or decompressing it. When the ebXML Module saves the ebXML message, it sets the User Status to Persisted. You can view the user status on My webMethods. The ebXML Module also saves the message after decrypting/decompressing it. ■ Set this property to `false` if you do not want to save a copy of the unprocessed ebXML message. The ebXML Module will save the ebXML messages after decrypting/decompressing it.
wm.ebxml.task.keepAliveTime	Specifies the maximum time (in seconds) that the threads which are in excess in the thread pool remain idle before they are terminated. This occurs when the number of threads in the thread pool is greater than the wm.ebxml.task.poolSize value. This parameter reduces the consumption of threads when the thread pool is not actively used. The default value is 3600.

Configuration Property	Definition
wm.ebxml.task.poolSize	Specifies the number of threads allowed in the thread pool for processing. If the number of threads in the pool is lesser than the ebxml.task.poolSize value, then a new thread is added to the pool. The default value is 10. The pool size should not be less than zero.
wm.ebxml.task.poolSizeMax	Specifies the maximum number of threads allowed in the thread pool for processing. The wm.ebxml.task.poolSizeMax value should be greater than the wm.ebxml.task.poolSize value. The default value is 20.
wm.ebxml.task.queueSize	Specifies the number of outbound ebXML messages that are allowed in queue for processing. This parameter is dependent on the ebxml.task.poolSize value as follows: ■ If the number of threads in the thread pool is lesser than the ebxml.task.poolSize value, then the Outbound Thread Pool adds a new thread to handle the request even if the other threads are idle rather than request for queuing. ■ If the number of threads in the thread pool is greater than the ebxml.task.poolSize value, then the Outbound Thread Pool adds the messages to the queue rather than adding a new thread for processing. ■ If the queue is full, then a new thread is added to the queue only if the number of threads in the thread pool is lesser than the ebxml.task.poolSizeMax value, otherwise, the request is rejected. The default value is 100. The queue size should not be less than zero.
wm.ebxml.timestamp	Specifies the timestamp format in the ebXML message. ■ Set this property to `default1` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:SSS'Z' (for example, 2008-05-20T13:20:10.663Z). This is the default setting.

Configuration Property	Definition
	■ Set this property to `default2` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:mm (for example, 2008-05-220T13:20:10). ■ Set this property to `default3` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:mm, where yyyy-MM-dd'T'HH:mm:ss is the local time, and -HH:mm is the time zone offset. (For example, "2008-05-20T13:20:10-05:00" represents "20 May 2008 13:20:10 GMT-05:00".) ■ Set this property to any other valid Java timestamp format. **Note:** For more information about the timestamp formats above, see the *XML Schema Part 2: Datatypes* document available at http://www.w3.org/TR/xmlschema-2
wm.ebxml.timezone	Specifies the time zone in the ebXML message. ■ Set this property to `GMT` to use the GMT time zone. This is the default setting. **Note:** When the wm.ebxml.timestamp property is set to either `default1` or `default2`, the wm.ebxml.timezone property, by default, takes the value `GMT`. ■ Set this property to `local` to use the time zone set on your local machine. ■ Set this property to any other valid Java time zone format.
wm.ebxml.TNConversationID	Use to configure the Trading Networks conversation ID that the wm.ip.ebxml.TN:getTNConversationId service generates. This property can have the following values: ■ Set this property to `ebxml` for the value of the ebxml conversation ID element, eb:ConversationID, to be used as the Trading Networks conversation ID. This is the default setting. ■ Set this property to `ebxml-receiverID` for the value of the receiver profile's internal ID appended to the ebXML conversation ID as,

Configuration Property	Definition
	eb:Conversation-receiverId to be used as the Trading Networks conversation ID. ■ Set this property to `ebxml-senderID-receiverID` for the value of the sender and receiver profile's internal ID appended to the ebXML conversation ID as, eb:ConversationId-senderId-receiverId to be used as the Trading Networks conversation ID.
wm.ebxml.resetContentHandlers	Use to reset the content handler to the one that Trading Networks expects for MIME message handling. This property can have the following values: ■ Set this property to `true` to reset the content handler to "com.wm.net.mime.DataContentHandler_WM" for all MIME types handled by Trading Networks. Use this setting if you are using ebXML Module in conjunction with other eStandards modules that use Trading Networks MIME APIs (for example, webMethods Chem eStandards Module or webMethods RosettaNet Module). ■ Set this property to `false` to preserve the content handlers that have been registered by SAAJ for specific MIME types. This is the default.

Configuring the Trading Networks Database

This section lists the high-level steps required to associate a JDBC pool. For more information, see the *webMethods Integration Server Administrator's Guide* for your release.

To configure the Trading Networks database

1. Run the database scripts for Trading Networks using Database Component Configurator.
2. Open Integration Server Administrator if it is not already open.
3. In Integration Server Administrator, select **Settings > JDBC Pools**, click **Create a new Pool Alias Definition** and specify the URL where the Trading Networks database scripts are created in step 1.
4. Click **Save Settings**.
5. Associate the JDBC pool alias with the Trading Networks Associated Function Alias.

Note:

For information about configuring My webMethods to work with Trading Networks, see the *webMethods Trading Networks Administrator's Guide* for your release.

Improving Performance

Caching

To enhance performance, the ebXML Module has a built-in caching mechanism for some webMethods Trading Networks database services.

To turn on caching

1. Open the config.cnf file located at *Integration Server_directory* \packages\WmebXML\config and specify the value of the wm.ebxml.caching property as `true`. For more information, see in *webMethods Trading Networks Administrator's Guide*.

2. Save the config.cnf file.

3. Reload the WmebXML package.

Bypassing the Use of Trading Networks Processing Rules

If you have not customized any of the default processing rules provided, you can achieve an additional performance improvement by bypassing the use of Trading Networks processing rules.

To bypass the use of Trading Networks processing rules

1. Open the config.cnf file located at *`IntegrationServer_directory`*\packages\WmebXML\config and specify the value of the wm.ebxml.bypassRoutingRule property as `true`. For more information, see the description in *webMethods Trading Networks Administrator's Guide*.

2. Save the config.cnf file.

3. Reload the WmebXML package.

4 Working with CPAs, TN Document Types, and Processing Rules

- What Is a Trading Partner Profile? 34
- What Is a Trading Partner Agreement (TPA)? 34
- About Collaboration Protocol Agreements (CPA) 35
- Exchanging Business Documents: User Scenarios 36
- Generating Trading Partner Profiles and a TPA from a CPA 38
- Defining Your Enterprise Profile 38
- Defining Your Trading Partners' Profiles 39
- Manually Creating a TPA 41
- Default TN Document Types Used by ebXML Module 43
- Defining Processing Rules 44

- "What Is a Trading Partner Profile?" on page 34
- "What Is a Trading Partner Agreement (TPA)?" on page 34
- "About Collaboration Protocol Agreements (CPA)" on page 35
- "Exchanging Business Documents: User Scenarios" on page 36
- "Generating Trading Partner Profiles and a TPA from a CPA" on page 38
- "Defining Your Enterprise Profile" on page 38
- "Defining Your Trading Partners' Profiles" on page 39
- "Manually Creating a TPA" on page 41
- "Default TN Document Types Used by ebXML Module " on page 43
- "Defining Processing Rules" on page 44

What Is a Trading Partner Profile?

A trading partner is any person or organization with whom you want to conduct business electronically. In the webMethods ebXML Module, a trading partner is defined by several criteria that you specify in a trading partner profile, including company name and identifying information, contact information, and delivery methods.

In addition to specifying trading partner profiles for all of your trading partners, you must specify a profile for your own organization.

You can define a trading partner profile using My webMethods Server. For procedural information about defining a trading partner profile, as well as descriptions of the fields you must complete when defining a trading partner profile, see the *webMethods Trading Networks Administrator's Guide* for your release.

What Is a Trading Partner Agreement (TPA)?

A Trading Partner Agreement (TPA) is a set of parameters that you can use to govern how business documents are exchanged between two trading partners. You view and customize the TPAs using the Agreement Details screen in Trading Networks Console. For information about working with TPAs in the Trading Networks Console, see the *webMethods Trading Networks Administrator's Guide* for your release.

Every TPA is uniquely identified by a Sender, Receiver, and an Agreement ID. During a transaction between trading partners, the webMethods ebXML Module uses this information to retrieve the TPAs for the initiator/sender and fulfiller/receiver in the transaction and to process the business documents exchanged. Every message that is exchanged in the webMethods ebXML Module is associated with a TPA. The TPAs provided with the webMethods ebXML Module contain a set of parameters that map to some (but not all) elements in the Service Header of a business document. The TPA parameters and the Service Header elements that the parameters map to, vary between *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

Your trading partner profiles, used in conjunction with TPAs, define how you and your trading partners exchange business documents.

About Collaboration Protocol Agreements (CPA)

To use the ebXML Module, you should have an agreement with your trading partners.

The ebXML Module TPA is based on Collaboration-Protocol Agreement (CPA) defined by the ebXML Collaboration-Protocol Profile and Agreement specifications. The CPA describes the message-exchange parameters agreement between two parties. In the ebXML Module, it is assumed that:

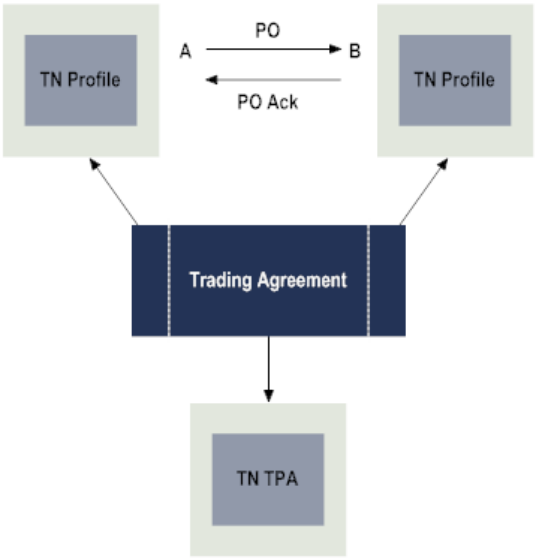
- CPAs have been created through some other means, such as a CPA editor.
- CPAs are used solely to configure webMethods Trading Networks/ebXML Message Service (MS) to facilitate message exchanges.

If you have a CPA, you can automatically generate Trading Networks profiles for the respective trading partners as well as a TPA.

If you do not have a formal agreement, you need to manually set up Trading Networks profiles and a TPA in Trading Networks.

For example, assume that trading partner A sends a Purchase Order (PO) to trading partner B and trading partner B responds by sending a PO acknowledgment (PO Ack) to trading partner A. Trading partner A and trading partner B negotiate a trading agreement, which should be a formal CPA to do business, as shown in the following diagram.

Figure 2. Using a Trading Agreement to Create TN Profiles and a TPA



The trading partner profile generated from the CPA uses CPA data like PartyInfo and Transport to establish the Corporate and Delivery Method details, whereas the trading partner agreement encapsulates the complete CPA.

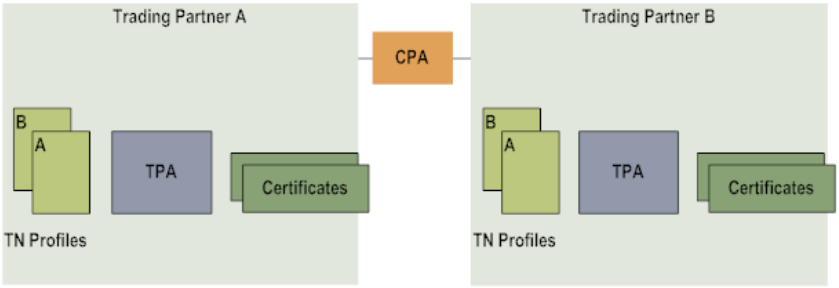
Exchanging Business Documents: User Scenarios

The following sections provide user scenarios of the role of CPAs and TPAs to exchange business documents using the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

Two-Partner Message Exchange

The following diagram illustrates a message exchange between two partners (your enterprise and your trading partner) using CPAs and TPAs. For this example, assume that your enterprise sends a purchase order to your trading partner, who responds with an ebXML acknowledgment.

Figure 3. Two-Partner Message Exchange

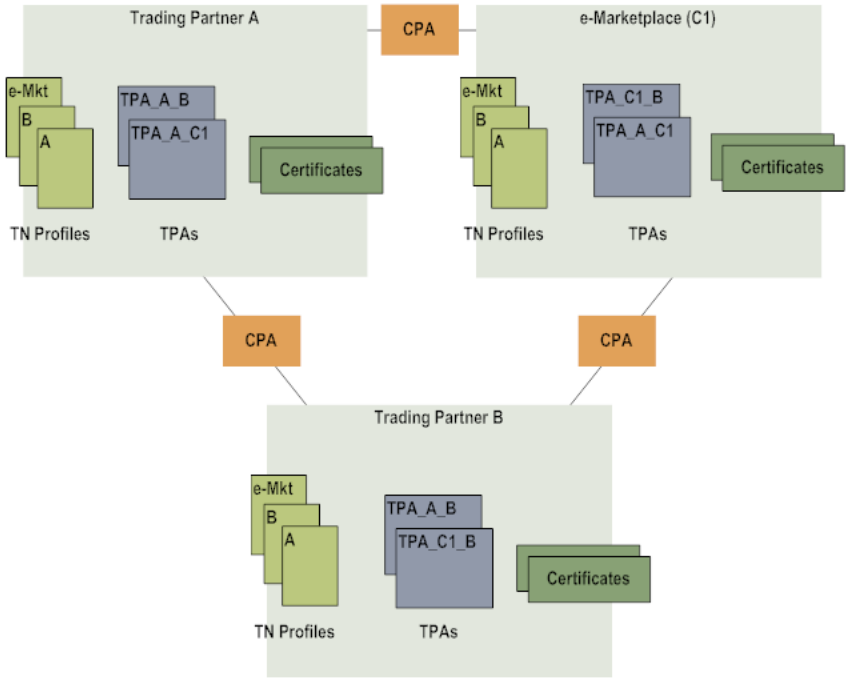


Trading partners A and B negotiate a business arrangement and create a CPA. From the CPA, trading partner A creates two Trading Networks profiles (one for self and one for trading partner B) and a TPA. Trading partner B also creates two profiles (one for self and one for trading partner A) and a TPA from the same CPA. The TPA is identical for both trading partner A and B. If any security measure is required, trading partner A and trading partner B each configure certificates for the Trading Networks profiles. Trading partners A and B are now ready to trade.

Multi-Hub Messaging in the e-Marketplace

The following diagram illustrates a message exchange among three parties: Trading partner A, the e-marketplace, and trading partner B. For this example, assume that trading partner A sends the e-marketplace an EDI purchase order. The e-marketplace maps the EDI purchase order to an XML purchase order and forwards it to trading partner B. Trading partner B responds with an ebXML acknowledgment. The e-marketplace converts the acknowledgment to an EDI 997 and forwards it to trading partner A.

Figure 4. Multi-Hub Messaging in the e-Marketplace



Trading partner A, the e-Marketplace (C1), and trading partner B negotiate and create three CPAs: one for the "trading partner A/e-Marketplace" relationship, one for the "e-Marketplace/trading partner B" relationship, and one for the "trading partner B/ trading partner A" relationship. See the CPA lines that connect the three parties in the diagram. After the parties create their respective CPAs, the following occurs:

- From the "trading partner B/trading partner A" CPA and the "trading partner A/ e-Marketplace" CPA, trading partner A creates three Trading Networks profiles (one for each party) and two TPAs, TPA_A_C1 (for the "trading partner A/e-Marketplace" relationship) and TPA_A_B (for the "trading partner B/trading partner A" relationship).
- From the "trading partner A/e-Marketplace" CPA and the "e-Marketplace/trading partner B" CPA, the e-Marketplace creates three Trading Networks profiles and two TPAs, TPA_A_C1 (for the "trading partner A/e-Marketplace" relationship) and TPA_C1_B (for the "e-Marketplace/trading partner B" relationship).
- From the "e-Marketplace/trading partner B" CPA and the "trading partner B/trading partner A" CPA, trading partner B creates three Trading Networks profiles and two TPAs, TPA_C1_B (for the "e-Marketplace/trading partner B" relationship) and TPA_A_B (for the "trading partner B/trading partner A" relationship).
- If any security measure is required, the parties configure the related certificates for all Trading Networks profiles.

All three parties are now ready to trade.

Note:
To accomplish multi-hop messaging, you must set up your TPAs in a specific way. To find out how to configure your TPAs for multi-hop messaging, see previousMSH, nextMSH, senderMSH, and receiverMSH in "Multi-hop Messaging" on page 66.

Generating Trading Partner Profiles and a TPA from a CPA

At design time, you can generate two Trading Networks partner profiles and a TPA from a CPA by running the wm.ip.ebxml.cpa:importCPAservice provided in the WmebXML package.

If the Trading Networks profiles and TPA already exist, the service can optionally update the existing profiles and TPA so that they contain all relevant information from the CPA. If you want the service to overwrite the existing Trading Networks profiles and TPA, set the inputs updateExistingProfiles and updateExistingTPA to `true`. By default, the service flags an error if the Trading Networks profiles and TPA already exist.

For an example of a service that generates two partner profiles (for example, a Buyer and a Seller) and a TPA, see the wm.ip.ebxml.cpa:importCPA service in the WmebXMLSample package.

To generate partner profiles and a TPA from a CPA

1. In Developer, create a service that invokes the pub.file:getFile service to read the CPA file. Invoke the service pub.string:bytesToString service to convert the bytes/stream output to a string format and pass this string to the wm.ip.ebxml.cpa:importCPA service.

2. Save the service and then run it, using the CPA as the input.

3. In Trading Networks, review the profiles and TPA to make sure they are correct. In some cases, you might be required to perform minor editing of the TPA. For the TPA data schema parameters and the possible values they are allowed to have, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

   Note:
   Certificates configuration is not done automatically. You must configure the certificates manually in the Trading Networks profiles.

4. At run time, the ebXML Messaging Service automatically retrieves the configuration information from the Trading Networks profiles and the TPA. For information about setting the parameters of the exportCPA and importCPA services, see "CPA Folder (wm.ip.ebxml.cpa)" on page 92.

Defining Your Enterprise Profile

Before defining your trading partner profiles in Trading Networks and exchanging business documents with your trading partners, you must first define your Enterprise profile. You define your enterprise profile by using the Partner Profiles page in My webMethods. For procedural information about defining your enterprise profile, as well as descriptions of the fields you must complete when defining your enterprise profile, see the *webMethods Trading Networks Administrator's Guide* for your release.

The following sections specify the required fields you must complete to define your enterprise profile.

Required Profile Fields

- On the **My webMethods** page, go to **Administration > Integration > B2B > Create Enterprise Profile** page and set the following required fields you must complete when defining your enterprise profile.

Required Profile Field for Enterprise	Description
Corporation Name	The name of your enterprise.
External ID Type Value	Your enterprise's D-U-N-S Number.

For descriptions of other fields you complete when you define your enterprise profile, see the *webMethods Trading Networks Administrator's Guide* for your release.

Delivery Setting Information

1. On the **My webMethods**page, go to **Administrator > Integration > B2B > Create Enterprise Profile** page. Click **Delivery Settings**.

2. Specify the following delivery settings:

   - For HTTP/S: If you specify any of the HTTP/S protocols as a delivery method, you must specify `invoke/wm.ip.ebxml.MSH/receive` as the location.
   - For SMTP: If you specify **Primary Email** as your preferred method, you must specify your email address in **Email**.

Security Information

The webMethods ebXML Module uses the same Trading Networks certificate for signing and decrypting.

Activating Your Enterprise Profile

You must activate (or enable) your enterprise profile before you can exchange documents with trading partners. For instructions, see the *webMethods Trading Networks Administrator's Guide* for your release.

Defining Your Trading Partners' Profiles

Each trading partner with whom you want to exchange business documents must have a trading partner profile in Trading Networks. After you have defined your enterprise profile, you are ready to define your trading partners' profiles.

You can define a trading partner profile using either My webMethods or Trading Networks Console. For procedural information about defining a trading partner profile, as well as descriptions of the fields you must complete when defining a trading partner profile, see the *webMethods Trading Networks Administrator's Guide* for your release.

The following sections specify the required fields you must complete to define a trading partner profile.

Required Profile Fields

- On the **My webMethods** page, go to **Administration > Integration > B2B > Create Enterprise Profile** page and set the following required fields you must complete when defining your partner's profile.

Required Profile Field for Trading Partner	Description
Corporation Name	The name of the trading partner.
External ID Type Value	Your trading partner's D-U-N-S Number.

For descriptions of other fields you complete when you define a trading partner profile, see the *webMethods Trading Networks Administrator's Guide* for your release.

Delivery Setting Information

1. On the **My webMethods**page, go to **Administrator > Integration > B2B > Create Enterprise Profile** page. Click **Delivery Settings**.

2. Specify the following delivery settings:

   - If you specify an HTTP/S protocol as the delivery method, specify the URL that your trading partner provides so that the trading partner can receive the ebXML messages that you send to the partner.
   - If you specify Primary EMail as the preferred protocol, enter the email address provided by your trading partner to receive the ebXML messages sent by you.

3. If the preferred protocol is not set for a receiver, the first protocol defined in the Delivery Method will be used for the message exchange. ebXML Module reports an error only if no delivery methods are specified in the partner profile.

Security Information

You can define the certificates you want to use when communicating with your trading partner.

The certificate names you enter in the fields on the security screens are used for signing the ebXML messages you send to your trading partner and for decrypting encrypted ebXML messages you receive from your trading partner.

Activating Your Trading Partners' Profiles

You must activate (or enable) your enterprise profile before you can exchange documents with your trading partners. For instructions, see the *webMethods Trading Networks Administrator's Guide* for your release.

Manually Creating a TPA

If you do not have a formal CPA, you must manually create a TPA by using the Agreement Details screen in the Trading Networks Console. For information on manually creating a TPA, see the *webMethods Trading Networks Administrator's Guide* for your release.

Specifying Agreement Details

When defining the TPA in Trading Networks, specify the following for the TPA fields in the Agreement Details screen in the Trading Networks Console:

In this TPA field...	Specify...
Sender	The name of the trading partner that has the sender role in the TPA.
Receiver	The name of the trading partner that has the receiver role in the TPA.
Agreement ID	A unique combination of alphanumeric characters that identifies the TPA.
IS Document Type	One of the following IS document types based on the version of ebXML CPP and CPA: ■ For ebXML CPP and CPA 1.0, specify com.wm.estd.ebXML.documents.CPA1:cpa_1 ■ For ebXML CPP and CPA 2.0, specify com.wm.estd.ebXML.documents.CPA2:cpa_2 To set the default values in the IS document type, see "Trading Partner Agreement Parameters Version 2. 0" on page 119and "Trading Partner Agreement Parameters Version 2. 0" on page 119.
Export Service	The wm.ip.ebxml.cpa:exportCPA service, which retrieves a stored copy of the original CPA from the TPA data.
Initialization Service	■ wm.ip.ebXML.cpa.initTPA1 service, which populates the com.wm.estd.ebXML.documents.CPA1:cpa_1 (for ebXML CPP and CPA 1.0)

In this TPA field...	Specify...
	■ wm.ip.ebXML.cpa.initTPA2 service, which populates the com.wm.estd.ebXML.documents.CPA2:cpa_2 (for ebXML CPP and CPA 2.0) The initialization service populates the TPA with default values. For information about default values, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

Modifying and Extending the TPA

To modify and extend the TPA as per your requirements, it is important to know the ebXML CPA (v1.0 and v2.0) document structure and its parameters. Although some minor differences exist between the TPAs for exchanging v1.0 messages and v2.0 messages, the overall structure is the same.

A TPA document contains two "PartyInfo" documents. Each PartyInfo document represents the messaging capabilities of a trading partner, identified by the "PartyId" document within PartyInfo. The PartyId value should map to the "External ID" of any one of the profiles defined in Trading Networks. A TPA document also contains one or more "Packaging" documents that define how the ebXML Message Header and payloads (defined within the SimplePart document) are packaged for transmittal.

The messaging characteristics associated with each party are defined using the "DocExchange," "Transport," and "DeliveryChannel" documents available within a PartyInfo document:

- The DocExchange document defines parameters for reliable messaging, digital signature, and encryption.
- The Transport document defines parameters for specifying the transport protocols supported by the partner. One or more "DocExchange" and "Transport" documents can be defined based on the messaging configurations supported by the partner.
- The DeliveryChannel document references a DocExchange and Transport document and contains additional parameters that you can configure to specify whether to:
  - Have a message exchange be synchronous or asynchronous.
  - Return an acknowledgment to acknowledge the receipt of a message.
  - Specify that the acknowledgment, if required, should be signed or unsigned.
  - Eliminate duplicate messages.

    You can define one or more DeliveryChannels for each PartyInfo document.

The "CollaborationRole" document within PartyInfo contains the Service "action" combinations that a party is capable of sending. Each action is linked to a DeliveryChannel and Packaging document.

Note:
If you want to share a TPA with more than one partner, you create a Default TPA by setting the sender and/or receiver to `Unknown`. The Default TPA then can be used as a template by any sender or receiver.

Default TN Document Types Used by ebXML Module

The WmebXML package provides the following default TN document types:

- ebXML Default
- ebXML Error
- ebXML Payload
- ebXML Receipt
- ebXML Routing

Note:
The TN document types supported by ebXML Module 6.0.1 are available with ebXML Module 7.1 SP1 to enable viewing of the messages that are persisted from earlier versions.

These TN document types are all instances of the Java class com.wm.estd.ebXML.doc.EbEnvelopeDocType, which extends the Java class com.wm.app.tn.doc.XMLDocType. The ebXML Module sets the following attributes for the TN document types:

- The ebXML Module sets the DocumentID attribute to the MessageID from the ebXML Envelope/Header/MessageHeader/MessageData element.
- The ebXML Module sets the GroupID attribute, by default, to the ConversationID from the ebXML Envelope/Header/MessageHeader/MessageHeader element. You can override this default so that the ebXML Module uses the CPAId for the GroupID instead. To configure the ebXML Module to do so, set the wm.ebxml.doctype.groupIDproperty in the ebXML configuration file (config.cnf file) to `cpaID`. For more information, see "Configuring webMethods ebXML Module" on page 23.

After determining the TN document type to use for an incoming ebXML message, the ebXML Module uses the TN document type to form a BizDocEnvelope. A BizDocEnvelope contains the original message and includes additional information that Trading Networks requires for routing and processing the message. In the ContentParts field of the BizDocEnvelope, the ebXML Module saves multiple content parts, the original message as "ebxml", the ebXML envelope as "Envelope", the header element as the "Header", the payloads as the content IDs, and so on.

The ebXML Module sets the User Status to reflect the nature of the ebXML message. The following is a list of all the possible values of User Status:

- ErrorMessage
- Ping
- Pong

- StatusRequest
- StatusResponse
- Acknowledgment
- Payload
- Receipt
- MessageInError
- MessageRouting
- Duplicate
- SendMessage
- Persisted

Customizing ebXML TN Document Types

You can customize the ebXML TN document types, if needed.

You might want to update the Identification Information to use XQL queries. As provided, none of the ebXML TN document types make use of XQL queries. Existing TN document types can be customized to create new document types. To limit the ebXML messages that use a customized TN document type, you can narrow the search by specifying XQL queries. You can add XQL queries to query information in the envelope of an ebXML message. You cannot query information in the payload. However, you can query on the Manifest element, which contains the information about the nature of a payload.

For information about how to work with TN document types, see the *webMethods Trading Networks Administrator's Guide* for your release.

Defining Processing Rules

The WmebXML package provides several processing rules for processing transport-level messages. By default, the processing rules have no action defined. The ebXML Module handles the processing of all incoming ebXML messages. If you need take a specific action for an ebXML message, customize the processing rule to define the action for the ebXML message.

Processing ebXML Messages

You can process ebXML messages by having the ebXML Module use Trading Networks processing rules to process a message. Whether the ebXML Module processes the ebXML messages using a processing rule depends on how you configure the wm.ebxml.bypassRoutingRule property. If you set this property to `false` in the ebXML Module configuration file (config.cnf), the ebXML Module will use processing rules. If you do *not* want to use processing rules, set this property to `true`, which is the default.

The ebXML Module provides processing rules for ebXML messages. However, as provided with ebXML Module 7.1 SP1, the processing rules do not specify any actions on the **Action** tab in the Trading Networks processing rules screen. You define how you want ebXML messages processed by configuring actions for the processing rules. For example, you can create a service you want executed to process messages; then you would use the **Execute a Service** action and identify the service you created. For more information, see "Defining Processing Rules" on page 44.

Trading Networks applies the processing rules after a document recognition process only if the associated processing rules are enabled.

The following lists the processing rules that are provided with the ebXML Module. Based on the criteria in the processing rules, Trading Networks determines the processing rule to use for an ebXML message based on the TN document type of the ebXML message.

This TN document type...	Matches these ebXML messages	Proceed using these processing rules	Whether passed to process management
ebXML Default	All transport-level messages: ■ ping ■ pong ■ status request ■ status response ■ duplicate	■ ebXML Ping ■ ebXML Pong ■ ebXML Status Request ■ ebXML Status Response ■ ebXML Duplicated Message	By default, not passed to process management if the ConversationID has no value. To set the ConversationID to the Envelope/Header/MessageHeader/ConversationID, configure the wmebxmldefaultDocTypeconversationID property to `true` in the ebXML Module configuration file, config.cnf.
ebXML Payload	■ A payload containing an ebXML message ■ A message-in-error transport-level message	■ ebXML Payload ■ ebXML Message in Error	Passed to process management if Trading Networks successfully extracts a conversationID from the message.
ebXML Error	ebXML error messages; that is, messages that contains an ErrorList element	ebXML Error Message	Passed to process management if Trading Networks successfully extracts a conversationID from the message.
ebXML Receipt	ebXML messages that contain a deliveryReceipt element **Note:**	ebXML Receipt	Passed to process management if Trading Networks successfully extracts a conversation ID from the message.

This TN document type...	Matches these ebXML messages	Proceed using these processing rules	Whether passed to process management
	The deliveryReceipt element is not present in the *ebXML Message Service Version 2.0 Specification*.		
ebXML Routing	Multi-hop messages	ebXML Message Routing	Passed to process management if Trading Networks successfully extracts a conversation ID from the message.

5 Working with ebXML Messages

- Supported Communication Protocols 48
- Initiating the ebXML Module Handshake 49
- Sending ebXML Messages 53
- Receiving ebXML Messages 55
- Checking the Status of ebXML Messages 57
- Viewing Transactions 59

- "Supported Communication Protocols" on page 48
- "Initiating the ebXML Module Handshake" on page 49
- "Sending ebXML Messages" on page 53
- "Receiving ebXML Messages" on page 55
- "Checking the Status of ebXML Messages" on page 57
- "Viewing Transactions" on page 59

Supported Communication Protocols

webMethods ebXML Module supports the following communication protocols for sending and receiving the ebXML messages.

- HTTP and HTTPS in both synchronous and asynchronous forms of transfer.
- SMTP in asynchronous form of transfer.

HTTP/S Support

To send an ebXML message via HTTP/S, you must configure webMethods Integration Server and trading partner profiles to use an HTTP/S server. For more information about configuring an HTTP/S port, see the *webMethods Integration Server Administrator's Guide* and *webMethods Trading Networks User's Guide* for your release.

To send and receive an ebXML message via HTTP/S

1. Configure an HTTP/S port on Integration Server.

2. Create a trading partner profile and within the profile, define an HTTP or HTTPS delivery method that uses the above port. You can create the profiles by using My webMethods. For more information about creating profiles, see the *webMethods Trading Networks Administrator's Guide* for your release.

Sending a Message via SMTP

To send an ebXML message via SMTP, you must configure webMethods Integration Server to use your local SMTP server. Ensure that an email account and listening services are configured to route the incoming emails to the ebXML services. You can set this information up in Integration Server Administrator under **Security > Ports > Add Port > webMethods/Email**. For detailed instructions, see the *webMethods Integration Server Administrator's Guide* for your release.

To send an ebXML message via SMTP

1. From Integration Server Administrator, select **Settings > Extended**.

2. Set the values for the following:

    a. Set the value of the watt.server.smtpServerproperty to your local SMTP server.

    b. Set the value of the watt.server.smtp.serverPort property to the port of the SMTP server. By default, the value is set to 25.

    For more information about setting the properties, see the *webMethods Integration Server Administrator's Guide* for your release.

3. From My webMethods, select **Administrator > Integration > B2B > Partner Profiles.**Create a trading partner profile and within the profile, define an SMTP delivery method that specifies the email address of the receiver.

Receiving a Message via SMTP

To receive an ebXML message via POP or IMAP

1. Add an email account with any email service provider that supports either POP3 or IMAP protocol. For instructions on adding an email port, see the *webMethods Integration Server Administrator's Guide* for your release.

2. In the Integration Server Administrator:

    a. Set up an email listener that points to the wm.ip.ebxml.MSH:receive service. Specify Global Service as the name of the listener.

    b. Select **Ports > Add Port > webMethods/Email**. For more information about setting the POP or IMAP ports, see the *webMethods Integration Server Administrator's Guide* for your release.

    c. Set the **Invoke service for each part of multipart message** to No.

    d. Set **Include email headers when passing message to content handler** to No and click **Save Changes**.

    e. From the **Ports** page, click **Access Mode** and set it to Allow By Default.

Initiating the ebXML Module Handshake

The following sections provide a brief introduction to the services in the wm.ip.ebxml.MSH folder that provide an entry point for message exchange between two partners using theebXML Module. For details about the services in the wm.ip.ebXML.MSH folder, see "Built-In Services" on page 91.

The pingUsingTPA Service

The wm.ip.ebXML.MSH:pingUsingTPAservice provides a ping service to another ebXML Message Service provider, that is, another trading partner's ebXML Message Service Handler (MSH). The pingUsingTPA service sends out an ebXML Ping message to the target ebXML MSH. If the target ebXML MSH is active, it responds with an ebXML Pong message. The communication channel is established when a sender MSH receives a Pong message.

The pingUsingTPA service is a part of both the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

To configure the pingUsingTPA service

1. From the My webMethods screen, set up two trading partner profiles in two instances of the webMethods Trading Networks.

   For example, define the first profile with *Name* = `Buyer Inc.`, *ExternalID* = `123456789`, and *ID type* = `DUNS`. Define the second profile with *Name* = `Seller Inc.`, *ExternalID* = `987654321`, and *ID type* = `DUNS`. If you do not want to run two instances of webMethods Trading Networks, you can define these two profiles on one instance of webMethods Trading Networks with the Buyer Inc. profile as the host profile.

2. Create a TPA using either *ebXML Message Service Version 1.0 Specification* or *ebXML Message Service Version 2.0 Specification*.

   - To manually create a TPA with the necessary parameters to initiate the messaging channel, set the following parameters for the buyer with the values as shown (example values shown for the *ebXML Message Service Version 2.0 Specification*):

Parameter	Value
CollaborationProtocolAgreement:cpaid	sample
CollaborationProtocolAgreement:version;	2.0
CollaborationProtocolAgreement.PartyInfo:partyName	Buyer
CollaborationProtocolAgreement.PartyInfo:defaultMshChannelId	buyerChannelId
CollaborationProtocolAgreement.PartyInfo:defaultMshPackageId	buyerPackageId
CollaborationProtocolAgreement.PartyInfo.PartyId:body	123456789
CollaborationProtocolAgreement.PartyInfo.PartyId:type	DUNS
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel:channelId	buyerChannelId
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel:docExchangeId	buyerDocExchangeId
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics:syncReplyMode	signalsAndResponse

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.DocExchange:docExchangeId	buyerdocExchangeId
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding:version	2.0

- To manually create a TPA with the necessary parameters to initiate the messaging channel, set the following parameters for the sender (example values shown):

Parameter	Value
CollaborationProtocolAgreement:PartyInfo:partyName	Seller
CollaborationProtocolAgreement.PartyInfo:defaultMshChannelId	sellerChannelId
CollaborationProtocolAgreement.PartyInfo:defaultMshPackageId	sellerPackageId
CollaborationProtocolAgreement.PartyInfo.PartyId:body	987654321
CollaborationProtocolAgreement.PartyInfo.PartyId:type	DUNS
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel:channelId	sellerChannelId
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel:docExchangeId	sellerDocExchangeId
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics:syncReplyMode	signalsAndReponse
CollaborationProtocolAgreement.PartyInfo.DocExchange:docExchangeId	sellerDocExchangeId
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding:version	2.0
CollaborationProtocolAgreement.SimplePart:id	Envelope
CollaborationProtocolAgreement.SimplePart:mimetype	text/xml
CollaborationProtocolAgreement.Packaging.CompositeList.Composite:id	Envelope
CollaborationProtocolAgreement.Packaging.CompositeList.Composite:mimetype	text/xml

3. From Developer, invoke the wm.ip.ebxml.MSH:pingUsingTPA service. When specifying the input parameters, specify:
   - ID and ID type of the Buyer Inc. information for the From Party.
   - ID and ID type of the Seller Inc. information for the To Party.
   - `sample`, for *CPAID*, which is the value you specified in the Buyer Inc. TPA.
4. From My webMethods, go to **Monitoring > Integration > B2B > Transactions** to view the transactions for the exchange between the sender and the receiver. Here is a sample of the transactions.

On the Buyer Inc. side:			
TN Document Type	Sender	Receiver	UserStatus
ebXML Default	Buyer Inc.	Seller Inc.	SendMessage:received
ebXML Default	Seller Inc.	Buyer Inc.	Pong

On the Seller Inc. side:			
TN Document Type	Sender	Receiver	UserStatus
ebXML Default	Buyer Inc.	Seller Inc.	Ping
ebXML Default	Seller Inc.	Buyer Inc.	SendMessage:sync

On the Buyer Inc. partner side, the first message is the ebXML Ping message, which the Buyer Inc. partner created and sent to the Seller Inc. partner. When the ping message arrives at the Seller Inc. partner, the ebXML Module on the Seller Inc. recognizes it as an ebXML Ping message (see the first message on the Seller Inc. side). The Seller Inc. partner then creates an ebXML Pong message and sends it back to the Buyer Inc. partner (see the second message on the Seller Inc. side).

Note that the User Status of the second message on the Seller Inc. partner is SendMessage:sync, which indicates that the response message is sent back synchronously. The ebXML Module sends the message back synchronously because the TPA specified when invoking the pingUsingTPA service has the syncReplyMode parameter set to `signalsAndResponse`.

Note:
The syncReplyMode parameter takes four values, `none`, `responseOnly`, `signalsAndResponse`, `signalsOnly`. For *ebXML Message Service Version 2.0 Specification*, there is an additional value, `mshSignalsOnly`. When the value is set to `none` the reply is asynchronous. At present, ebXML Module interprets the values of `responseOnly`, `signalsAndResponse`, `signalsOnly`, and `mshSignalsOnly`as synchronous, and does not specifically handle each of these values.

When the pong message arrives at the Buyer Inc. partner, the ebXML Module on the Buyer Inc. side recognizes it as an ebXML Pong message (see the second message on the Buyer Inc. side).

5. From My webMethods, select **Monitoring > Integration > B2B > Transactions > View Related Documents** for the first message on the Buyer Inc. side. You should see that the pong message is related to the ping message that the Buyer Inc. partner sent. The ebXML Module extensively uses the Trading Networks related documents feature to provide you a better view of all ebXML messages in a business process.

6. Note also that all four messages have a TN document type of ebXML Default and the ebXML Module uses the User Status in each message to indicate the character of the message. The ebXML Module maps the MessageId in the ebXML envelope to the DocumentID. You can see the DocumentID in the My webMethods Transaction page. The ebXML Module maps the

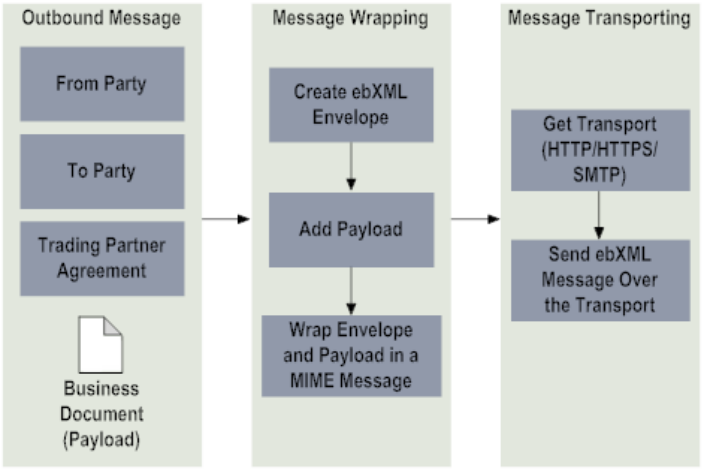
conversationId in the ebXML envelope is mapped to the GroupID. You can see the GroupID in the My webMethods Transaction page.

Note:
For ebXML messages that do not contain multiple parts, that is, the ping and pong messages, ebXML Module uses the non-multipart MIME structure. For information about the non-multipart MIME structure, see the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

Sending ebXML Messages

The send services that the ebXML Module provides allow you to wrap a business document with an ebXML envelope to form an ebXML message that you can send to a trading partner via the ebXML transport. The following figure illustrates the data flow in the send services to its destination over HTTP/S or SMTP.

Figure 5. Data Flow for an Outbound ebXML Message



The ebXML Module invokes the WmPublic services pub.client.http or pub.client.smtp to deliver the ebXML message.

Before sending an ebXML message to the receiver, the ebXML Module performs the following steps:

1. Obtains the input information that ebXML Module needs to create the outbound messages:
   - The From Party and To Party information of the trading partners.
   - The CPAId between the two trading partners.
   - The business document or the payload to send to the trading partner via the ebXML transport.
2. Creates the outbound message by wrapping the business document within an ebXML envelope. The ebXML Module:

- Creates an ebXML envelope with the From Party, To Party, and TPA information provided in Step 1.
- Adds all the payloads to be transferred to the ebXML message at the end of the envelope as attachments.
- Packages the envelope and the payloads together into an output message. This output message is a SOAP message for messages without a payload and a MIME message for messages with payloads.
- Computes and sets the value of the TimeToLive element, as follows:
  - For Non Reliable Messaging, TimeToLive = Timestamp in the message header + PersistDuration
  - For Reliable Messaging, TimeToLive = Timestamp in the message header + ((Retries + 1) * RetryInterval)

3. Transports the outbound message to the receiver. The ebXML Module:
   - Uses the transport method that you configure in the TPA to transport the wrapped ebXML message. The supported transport methods are HTTP/S and SMTP.
   - Transports the SOAP message or the MIME message over the configured transport.

ebXML Module Send Services

The ebXML Module provides five send services and one route service in the wm.ip.ebxml.MSH folder.

wm.ip.ebxml.MSH:sendAck
This service sends a signed/unsigned acknowledgment for a received ebXML message. For more information about the parameters of this service, see "wm.ip.ebxml.MSH:sendAck" on page 96.

wm.ip.ebxml.MSH:sendBizDoc
A back-end system sends a document and Trading Networks recognizes it using a TN document type that defines the structure of the back-end system document. To process the back-end system document, the user can set up the processing rule that uses the **Execute a Service** action. In the **Execute a Service** action, the service can invoke this sendBizDoc service specifying a TPA to use for the ebXML parameters. The sendBizDoc service gets the BizDocEnvelope for the back-end system document, wraps it with an ebXML envelope, and then transports it to the trading partner as the TPA specifies. For more information about the parameters of this service, see wm.ip.ebxml.MSH:sendBizDoc.

wm.ip.ebxml.MSH:sendError Msg
This service sends an ebXML Error Message for a received ebXML message. For information about the parameters of this service, see wm.ip.ebxml.MSH:sendErrorMsg.

wm.ip.ebxml.MSH:sendReceipt
This service sends an ebXML Receipt for a received ebXML message. This service is only applicable to the *ebXML Message Service Version 1.0 Specification*. For more information about the parameters of this service, see wm.ip.ebxml.MSH:sendReceipt.

wm.ip.ebxml.MSH:sendUsingTPA

This service provides the send service you should use for transmitting a business document to a trading partner using the *ebXML Message Service Version 1.0 Specification* or the *ebXML Message Service Version 2.0 Specification*.

For example, to use the sendUsingTPA service, to send an XML purchase order to the Seller Inc. partner, perform the following steps (example values are shown):

1. Create a TPA manually or import an existing CPA. For more information about creating a TPA manually, see "Manually Creating a TPA" on page 41
2. Create a java.io.InputStream element from the XML purchase order and pass it the parameter Payloads:stream. Set the parameter Payloads:partID to a value that maps to one of the SimplePart IDs in the TPA.
3. Specify values for required To, From, and CPAId input parameters to the service. These values set the values for the MessageHeader elements To, From, and CPAId for the outbound message. ebXML Module also uses these values to identify the TPA required to populate the other values in the message.
4. Optionally, specify a value for the ConversationId, MessageId, service, and action input parameters..

For more information about the parameters of the sendUsingTPA service, see wm.ip.ebxml.MSH:sendUsingTPA

wm.ip.ebxml.MSH:routeUsingTPA

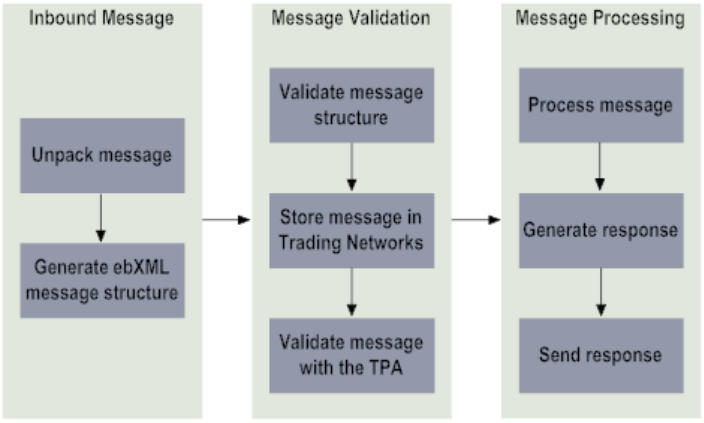
The wm.ip.ebxml.MSH:routeUsingTPA service sends/routes an ebXML message through an intermediary node. You have to modify the TPA to include the routing information that comprises the list of intermediary nodes. This service does not contain packaging information for the payloads, and is therefore easier to manage. The TPA contains the packaging information. It is recommended that you use this service for routing. For more information about the parameters of this service, see wm.ip.ebxml.MSH:routeUsingTPA.

Note:
For ebXML messages that do not contain multiple parts, that is, the Ping and Pong messages, ebXML Module uses the non-multipart MIME structure. For information about the non-multipart MIME structure, see the ebXML Message Service Version1.0 Specification and ebXML Message Service Version2.0 Specification.

Receiving ebXML Messages

The wm.ip.ebxml.MSH:receive service is the entry point for all inbound ebXML messages. The following figure shows the data flow in the receive service.

Figure 6. Data Flow for an Inbound ebXML Message



The receive service first unpacks the incoming MIME message into two or more parts: the ebXML envelope and the payload(s). If the incoming message is a SOAP message, the receive service translates it into an ebXML envelope. The service accepts the message if it can extract the sender, receiver, and message ID from the envelope. Otherwise, it generates a SOAP Fault that it returns it to the sender, according to the SOAP specifications.

The service then validates the ebXML envelope against the envelope schema specified in the *ebXML Message Service Version 1.0 Specification* and the *ebXML Message Service Version 2.0 Specification*.

The receive service then checks the ebXML envelope for a message type and retains the message type as User Status in the Trading Networks. The ebXML Module provides TN document types that will trigger a different processing rule. The ebXML Module checks for all duplicate or routing messages and processes them accordingly. For more information about TN document types and processing rules, see "Working with CPAs, TN Document Types, and Processing Rules" on page 33.

If there is a need to process the payload(s), the service unpacks the payload(s) and invokes the services specified in the ebXML message.

The receive service then forwards the message to webMethods Trading Networks, which recognizes the incoming document using one of the TN document types that the ebXML Module provides. For example, if the message type is a Payload, that is, the message carries payload(s), Trading Networks recognizes it as an ebXML Payload TN document type.

Depending on the message type, Trading Networks invokes the corresponding processing rule. For more information about processing rules, see "Defining Processing Rules" on page 44.

For messages of the type Payload, Receipt, and Error, the TN document types indicates that Trading Networks is to extract a conversationId. As a result, after Trading Networks performs the actions in a processing rule, it passes the document to the Process Engine so the Process Engine can attempt to process the message as part of a business process. As stated in the earlier section about processing rules, you can customize these three processing rules to meet your requirements.

The ebXML Module then generates a synchronous/asynchronous response for the message. For example, if the inbound message is a ping message, it generates a pong message and delivers the pong message to the sender.

ebXML Module Receive Service

The wm.ip.ebxml.MSH:receive service provides the receive service you should call for all incoming ebXML Message Service Protocol messages. The following is an example of an ebXML Ping message posted by HTTP.

```
Content-Length:1307
SOAPAction:"ebXML"
Content-Type:text/xml; charset="UTF-8"
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://www.oasis-
open.org/committees/ebxml-msg/schema/envelope.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
<SOAP-ENV:Header>
<eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="2.0">
<eb:From><eb:PartyId eb:type="DUNS">123456789</eb:PartyId></eb:From>
<eb:To><eb:PartyId eb:type="DUNS">987654321</eb:PartyId></eb:To>
<eb:CPAId>basic71/ms20/defaultTPA</eb:CPAId>
<eb:ConversationId>ebXML</eb:ConversationId>
<eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
<eb:Action>Ping</eb:Action>
<eb:MessageData>
<eb:MessageId>53o6dg0035vuulvq00000044</eb:MessageId>
<eb:Timestamp>2008-04-24T10:16:42.312Z</eb:Timestamp>
<eb:TimeToLive>2008-04-24T12:16:42.312Z</eb:TimeToLive>
</eb:MessageData>
</eb:MessageHeader>
<eb:SyncReply SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
SOAP-ENV:mustUnderstand="1" eb:version="2.0"/>
</SOAP-ENV:Header><SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
```

Checking the Status of ebXML Messages

The wm.ip.ebxml.MSH:statusUsingTPA service provides the "Message Status Request Service", as specified by the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. This service sends an ebXML *Message Status Request* to check the status of an ebXML message on another MSH. That MSH, in turn, responds with an ebXML *Message Status Response* message.

ebXML Module Message Status Request Service

To run the message status request service

1. If you have not already done so, set up the profiles for using the wm.ip.ebxml.MSH:pingUsingTPA service as described in "Initiating the ebXML Module Handshake" on page 49.

2. From Developer, invoke the wm.ip.ebxml.MSH:pingUsingTPA service.

3. Go to My webMethods: **Monitoring > Integration > B2B > Transactions** and select a DocumentID from the Seller Inc. partner's transaction view.

4. From Developer, invoke the wm.ip.ebxml.MSH:statusUsingTPA service. Set the MessageID input variable to the DocumentID value that you selected in the previous step. Also set the following example parameters:

   - To:ID = 987654321
   - To:IDType = DUNS

5. Go to My webMethods: **Monitoring > Integration > B2B > Transactions**and view the transactions for both trading partners. Here is an example that illustrates the transactions:

On the Buyer Inc. side:			
TN Document Type	Sender	Receiver	UserStatus
ebXML Default	Buyer Inc.	Seller Inc.	SendMessage:sent
ebXML Default	Seller Inc.	Buyer Inc.	StatusResponse

On the Seller Inc. side:			
TN Document Type	Sender	Receiver	UserStatus
ebXML Default	Buyer Inc.	Seller Inc.	StatusRequest
ebXML Default	Seller Inc.	Buyer Inc.	SendMessage:sent

The first message on the Buyer Inc. side is the ebXML *Message Status Request* message that the Buyer Inc. created and sent to the Seller Inc. partner. When the Seller Inc. partner receives it (see the first message on the Seller Inc. side), it creates an ebXML *Message Status Response* message and sends it back to the Buyer Inc. partner (see the second message on the Seller Inc. side).

Note that User Status of the second message for the Seller Inc. partner is SendMessage:sent. This indicates that the response message is asynchronous, based on the TPA parameter, syncReplyMode = `none`, that you set in the configuring the pingUsingTPA service procedure in "The pingUsingTPA Service" on page 50.

Note:
The syncReplyMode parameter takes four values, `none`, `responseOnly`, `signalsAndResponse`, `signalsOnly`. For *ebXML Message Service Version 2.0 Specification*, there is also an additional value, `mshSignalsOnly`. When the value is set to `none` the reply is asynchronous. At present, ebXML Module interprets the values of `responseOnly`, `signalsAndResponse`, `signalsOnly`, and `mshSignalsOnly` as synchronous, and does not specifically handle each of these values.

When the status response message arrives at the Buyer Inc. partner, the ebXML Module recognizes it as an ebXML *Message Status Response* message.

The ebXML *Message Status Response* will indicate the status of the message. For information on the status of the message, see the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

Viewing Transactions

All the ebXML messages that the ebXML Module send services generate and their responses are available as transactions. You can view the transactions to determine the status of each transaction and the activity of the various involved ebXML messages.

Viewing Transactions Using My webMethods

On My webMethods, go to **Monitoring > Integration > B2B > Transactions** page to query and analyze results of messages that are sent or received by Trading Networks. For information about transaction analysis, see the *webMethods Trading Networks User's Guide* for your release.

Viewing Transactions Using Trading Networks Console

On the Trading Networks Console, go to **View > Transaction Analysis**. The Transaction Analysis screen allows you to query and analyze results of messages that are sent or received by Trading Networks. For information about transaction analysis, see the *webMethods Trading Networks User's Guide* for your release.

Note:
It is recommended to use the Transaction page of My webMethods as the Transaction Analysis feature of Trading Networks Console is deprecated.

6 ebXML Module Messaging Features

- Reliable Messaging 62
- Message Ordering 64
- Multi-hop Messaging 66
- Large Business Document Handling 70
- Payload Compression 71
- MIME Encoding Payloads 72
- Message Exchange Using a Proxy Server 74
- Using the ebXML Module in a Clustered Environment 74

- "Reliable Messaging" on page 62
- "Message Ordering" on page 64
- "Multi-hop Messaging" on page 66
- "Large Business Document Handling" on page 70
- "Payload Compression" on page 71
- "MIME Encoding Payloads" on page 72
- "Message Exchange Using a Proxy Server" on page 74
- "Using the ebXML Module in a Clustered Environment" on page 74

Reliable Messaging

The Reliable Messaging feature defines an interoperable protocol for the MSHs to reliably exchange messages. That is, when an MSH sends a message, the recipient MSH receives the message *once and only once*. Reliable Messaging also handles the delivery and acknowledgment of ebXML Messages. The feature includes message handling for persistence, retry, error notification, duplicate detection, and acknowledgment of messages that require reliable messaging. Reliable Messaging is achieved when a receiver MSH responds to a message with an acknowledgment.

Enabling ebXML Module to Use Reliable Messaging

To configure ebXML Module for reliable messaging

1. Start the Trading Networks Console.
2. Open the agreement for edit:
   a. From Trading Networks Console, select **View > Agreements**.
   b. Select the agreement you would like to edit. Click**Edit**.

      The **Agreements Details** screen appears.
   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.

      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.

3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

4. Update the settings in the TPA structure to enable reliable messaging as shown:

   - If you are using *ebXML Message Service Version 1.0 Specification*, set the following example values for the parameters:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.deliverySemantics	OnceAndOnlyOnce
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.deliveryReceiptRequested	None
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.Retries	4
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.RetryInterval	30
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.Characteristics.ackRequested	true

     For information on each of the parameters and their values, see "Trading Partner Agreement Parameters Version 2. 0" on page 119.

   - If you are using *ebXML Message Service Version 2.0 Specification*, set the following example values for the parameters:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics.duplicateElimination	always
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding.ReliableMessaging.Retries	4
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding.ReliableMessaging.RetryInterval	30
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics.ackRequested	true

     For information on each of the parameters and their values, see "Trading Partner Agreement Parameters Version 2. 0" on page 119.

5. After enabling reliable messaging by updating the TPA, the ebXML Module ensures that the message is sent reliably to the receiver MSH. Reliable Messaging depends on the sender MSH receiving the acknowledgment message from the receiver MSH. If the acknowledgment message does not reach the sender MSH, the ebXML Module re-sends the message for the defined number of maximum retries, which are triggered for a defined retry interval, until the ebXML Module delivers the message.

6. For a Reliable message delivered, TimeToLive is computed as:

   TimeToLive = Timestamp in the message header + ((Retries + 1) * RetryInterval)

Note:
If you want the receiver of a message to send back a transport-level acknowledgment, set the ackRequested parameter to true.

Testing the Reliable Messaging Configuration

After completing the TPA configuration for Reliable Messaging, you need to test the configuration:

To test the reliable messaging configuration

1. From Developer, invoke the wm.ip.ebXML.MSH: sendUsingTPA service to send the ebXML messages to the trading partner using the configured TPA.

2. From My webMethods, select **Monitoring > Integration > B2B > Transactions**. Specify the criteria for **Saved Search** to search for the ebXML message you want to view.

3. Note the User Status for the ebXML message.

   On the Transactions page of My webMethods, view the User Status to determine whether the ebXML Module has successfully delivered the message. The ebXML Module sets the User Status to:

   - SendMessage:*n*, if it has attempted to deliver the message, but the message has not yet been acknowledged. In the User Status, *n* indicates the number of retries the ebXML Module has attempted to deliver the message.
   - SendMessage:ack or SendMessage:received if the ebXML Module has successfully delivered the message.

4. On the receiver side, check the processing of duplicate messages. The CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics:duplicateElimination parameter determines whether the duplicateElimination element within the MessageHeader element in the SOAP Header is to be present. The duplicateElimination parameter, if present, identifies a request by the sender for the receiving MSH to check for duplicate messages. If the duplicateElimination parameter is not present, the ebXML Module does not eliminate the duplicate messages.

Message Ordering

The Message Ordering feature allows presenting messages to the receiver in a specific order. The receiver MSH forwards the messages in the same specific order to the receiver application for processing the messages. The Message Ordering feature requires the use of the Reliable Messaging feature.

Enabling the Message Ordering Feature

To enable Message Ordering, create a TPA on the sender side and the receiver side.

To configure ebXML Module for Message Ordering

1. Start the Trading Networks Console.
2. Open the agreement for edit:
   a. From Trading Networks Console, select **View > Agreements**.
   b. Select the agreement you would like to edit. Click**Edit**.
      The **Agreements Details** screen appears.
   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.
      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.
3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.
4. Update the settings in the TPA structure to enable reliable messaging as shown:
   - If you are using *ebXML Message Service Version 1.0 Specification*, set the following parameters:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.deliverySemantics	OnceandOnlyOnce
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.messageOrderSemantics	Guaranteed
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging.PersistDuration	10 (time in seconds)
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.Characteristics.syncReplyMode	none

   - If you are using *ebXML Message Service Version 2.0 Specification*, set the following parameters:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics.duplicateElimination	always
PartyInfo.DeliveryChannel.Characteristics:ackRequested	always
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding.ReliableMessaging.RetryInterval	5 (time in seconds)

Parameter	Value
CollaborationProtocolAgreement:PartyInfo:DocExchange:ebXMLSenderBinding:ReliableMessaging:Retries	30
CollaborationProtocolAgreement:PartyInfo:DocExchange:ebXMLReceiverBinding:PersistDuration	10 (time in seconds)
CollaborationProtocolAgreement:PartyInfo:DeliveryChannel:MessagingCharacteristics:syncReplyMode	none

Note:
For Message Ordering, do not set the syncReplyMode parameter to work in synchronous mode because as per the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*, Message Ordering is not supported with synchronous messaging. Also the duplicateElimination parameter must not be set to `never`.

5. Invoke the wm.ip.ebxml.MSH:sendUsingTPA service for the ebXML Module to deliver a sequence of messages to the receiver. Trading Networks creates a sender specific TPA to track the sequence of messages that the sender sends.

   When a message is sent using the Message Ordering feature, this TPA keeps track of the sequence of the messages as sent by the sender. The messages that the ebXML Module cannot deliver to the receiver are persisted by the sender, and the sender retries to send the messages. The receiver does not pass the messages to the receiver application if the messages are out of sequence. The ebXML Module starts the processing of the messages on the receiver side only after receiving all the messages of the defined sequence. You can view the transactions in My webMethods: **Monitoring > Integration > B2B > Transactions**. For more information about viewing transactions, see "Viewing Transactions" on page 59.

Multi-hop Messaging

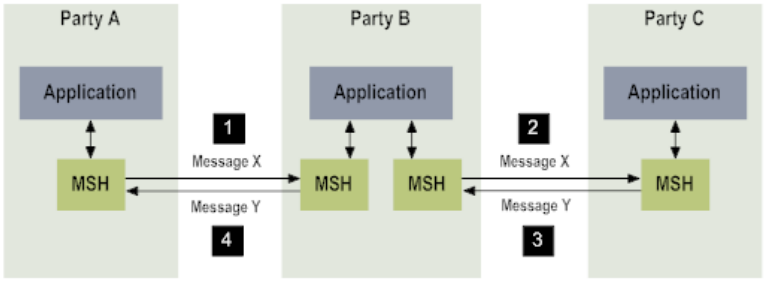
Multi-hop is the process of passing the message through one or more intermediary nodes or MSHs. An intermediary node is any node or MSH that receives the message, but is not the sending or the receiving MSH. The ebXML Module uses the intermediary nodes as store-and-forward entities.

In the following figure, Party A sends Message X to Party B. Party B then performs some processing and forwards the message to Party C. The payload(s) that Party C receives in Message X will be the same payloads that Party A sent out.

- The TraceHeaderList element in the ebXML envelope records the routing path a message takes for the *ebXML Message Service Version 1.0 Specification*.
- The MessageHeader:From and MessageHeader:To elements record the routing path a message takes for the *ebXML Message Service Version 2.0 Specification*.

Note: In the case of *ebXML Message Service Version 1.0 Specification*, if the ebXML Module sends your message to a hub first, and you want the hub to process your messages before it forwards the messages to the next hub, assign meaningful values to the advanced/Via/Service and the advanced/Via/Action parameters. The hub completes the processing of the payloads before it forwards the messages to the next hub.

Figure 7. ebXML Message Routing in a Multi-hop Scenario



Configuring Multi-hop Messaging

Use the routing parameter in the TPA to configure multi-hop messaging.

To configure routing information for a multi-hop scenario

1. Start the Trading Networks Console.
2. Open the agreement for edit:
   a. From Trading Networks Console, select **View > Agreements**.
   b. Select the agreement you would like to edit. Click**Edit**.
      The **Agreements Details** screen appears.
   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.
      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.
3. Modify the default values of the TPA parameters, as required. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.
4. Update the values of the following Routing parameters in the TPA to provide the routing information for multi-hop messaging:
   - previousMSH
   - nextMSH
   - senderMSH
   - receiverMSH

The intermediary nodes use the values of the parameters previousMSH and nextMSH to determine the next MSH to forward the message. The sender and receiver intermediaries (the origin node and the destination node) use the values of the parameters senderMSH and receiverMSH to load the original sender-receiver TPA.

For example, assume that Party A is the sender and Party B is the receiver of a message and Party C1, C2 … C*n* are the intermediary nodes. To accomplish the multi-hop messaging, you must set up the TPAs as follows:

Party Name	For the agreement between...	Set the following TPA parameters to...
Party A	Party A and Party B	nextMSH.ids=C1 nextMSH.idTypes=DUNS previousMSH.ids=Cn previousMSH.idTypes=DUNS Leave all the other Routing parameters blank.
	Party A and Party C1	nextMSH.ids=C2 nextMSH.idTypes=DUNS senderMSH.ids=A senderMSH.idTypes=DUNS receiverMSH.ids=B receiverMSH.idTypes=DUNS Leave the previousMSH parameter blank.
Party C1	Party A and Party C1	nextMSH.ids=C2 nextMSH.idTypes=DUNS senderMSH.ids=A senderMSH.idTypes=DUNS receiverMSH.ids=B receiverMSH.idTypes=DUNS Leave the previousMSH parameter blank.
	Party C1 and Party C2	previousMSH.ids=A previousMSH.idTypes=DUNS nextMSH.ids=C3 nextMSH.idTypes=DUNS

Party Name	For the agreement between...	Set the following TPA parameters to...
		Leave all the other Routing parameters blank.
Party C2	Party C1 and Party C2	previousMSH.ids=A previousMSH.idTypes=DUNS nextMSH.ids=C3 nextMSH.idTypes=DUNS Leave all the other Routing parameters blank.
	Party C2 and C3	previousMSH.ids=C1 previousMSH.idTypes=DUNS nextMSH.ids=C4 nextMSH.idTypes=DUNS Leave all the other Routing parameters blank.
Party C*n*	Party C*n*-1 and Party C*n*	previousMSH.ids=C*n*-2 previousMSH.idTypes=DUNS nextMSH.ids=B nextMSH.idTypes=DUNS Leave all the other Routing parameters blank.
	Party C*n* and Party B	previousMSH.ids=C*n*-1 previousMSH.idTypes=DUNS senderMSH.ids=A senderMSH.idTypes=DUNS receiverMSH.ids=B receiverMSH.idTypes=DUNS Leave the nextMSH parameter blank.
Party B	Party C*n* and Party B	previousMSH.ids=C*n*-1 previousMSH.idTypes=DUNS senderMSH.ids=A senderMSH.idTypes=DUNS receiverMSH.ids=B

Party Name	For the agreement between...	Set the following TPA parameters to...
		receiverMSH.idTypes=DUNS Leave the nextMSH parameter blank.
	Party A and Party B	nextMSH.ids=C1 nextMSH.idTypes=DUNS previousMSH.ids=Cn previousMSH.idTypes=DUNS Leave all the other Routing parameters blank.

Each party only needs to be aware of two other parties and two TPAs. For example, Party A only needs to know Party B and Party C1, and TPA_A_B and TPA_A_C1. Party C2 only needs to know Party C1 and C3, and TPA_C1_C2 and TPA_C2_C3.

5. To use reliable message with multi-hop messaging, you must enable reliable messaging for all intermediary nodes by configure the TPAs of all the intermediaries. For more information, see "Reliable Messaging" on page 62.

Large Business Document Handling

To send or receive large ebXML messages, configure your system to use the facilities of the ebXML Module for processing large messages. To use large document handling, you need to set configuration properties in the Trading Networks properties file and perform configuration tasks on the Integration Server.

To enable the ebXML Module to handle large ebXML messages

1. Edit the *IntegrationServer_directory*\packages\WmTN\config\properties.cnf file to specify the following property:

   tn.BigDocThreshold = *size*

   where *size* specifies how many bytes a document must contain for Trading Networks to consider the document to be large. For example, if you specify the following, Trading Networks considers all documents greater than 1,000,000 bytes as large, when tn.BigDocThreshold=1000000

2. Save the properties.cnf file.

3. Reload the WmTN package.

   When an ebXML message size is larger than the one you specify in the line above, webMethods Trading Networks processes the large message using the large message handling facility. For detailed information about the large message handling facilities, see the *webMethods Trading Networks Administrator's Guide* for your release.

4. Configure settings for the webMethods Integration Server:

   a. From the Integration Server Administrator, select **Settings > Extended**.

   b. Click **EditExtendedSettings**. In the Extended Settings editor, specify values for the following properties:

      - watt.server.tspace.timeToLive - The value set for this property specifies the minimum amount of time tspace will be alive. In case of a "File Not Found" exception message, increase the timeToLive value to a higher duration. The default value for this property is 30 seconds.
      - watt.server.tspace.location - The absolute directory path of the hard disk drive space where Trading Networks is to temporarily store large documents rather than keep them in memory. The directory you specify is on the same machine as the Integration Server. For example:

        For Windows: watt.server.tspace.location=D:\LargeDocTemp

        For UNIX: watt.server.tspace.location=/opt/webmethods/tspace

      If you do not specify a value for watt.server.tspace.location, Trading Networks uses the value defined by the Java system property java.io.tmpDir, which defaults to the value of the environment variable Temp on most platforms.

      For information about these properties, see the *webMethods Trading Networks Administrator's Guide* for your release.

5. Click **Save Changes**.

6. Restart the webMethods Integration Server for the changes to take effect.

Payload Compression

ebXML Module provides support for payload compression using zip and gzip compression algorithms. For outbound ebXML messages, ebXML Module performs the payload message compression only if you set the TPA parameters to perform compression.

To enable payload compression

1. Start the Trading Networks Console.

2. Open the agreement for edit:

   a. From Trading Networks Console, select **View > Agreements**.

   b. Select the agreement you would like to edit. Click**Edit**.

      The **Agreements Details** screen appears.

c. On the right side of the **Agreement Details** screen, click **Set Inputs**.

The input for com.wm.estd.ebxml.documents appears.

Note:
You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.

3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

- If you are using *ebXML Message Service Version 1.0 Specification*, set the following example parameters:

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:mimetype	application/zip or application/gzip
CollaborationProtocolAgreement.Packaging.SimplePart:Id	payload-id

- If you are using *ebXML Message Service Version 2.0 Specification*, set the following example parameters:

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:mimetype	application/zip or application/gzip
CollaborationProtocolAgreement.SimplePart:Id	payload-id

4. For inbound ebXML messages, the ebXML Module processes a payload for decompression only if the Content-Type MIME header for the payload is specified as "application/xml" or "application/gzip". Note that the Content-Type of the decompressed payload defaults to "text/xml" if there is no matching "Content-ID" in the processing TPA packaging section.

MIME Encoding Payloads

The ebXML Module supports MIME encoding for payloads. The ebXML Module supports all the character encodings defined in RFC 2045 such as "base64", "quoted-printable", "7bit", "8bit", and "binary". Also supports "uuencode" character encoding.

To configure MIME encoded payload support, set the appropriate TPA parameters.

To configure the ebXML Module for MIME encoded payloads

1. Start the Trading Networks Console.

2. Open the agreement for edit:

   a. From Trading Networks Console, select **View > Agreements**.

   b. Select the agreement you would like to edit. Click**Edit**.

      The **Agreements Details** screen appears.

   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.

      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.

3. Modify the default TPA parameters, as required. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

4. Set the required character encoding, as follows:

   - For *ebXML Message Service Version 1.0 Specification* messages, set the following TPA parameter as shown below:

Set this parameter...	To...
[illegible]	The value of the required character encoding. For example, for base64 character encoding, set the value to `content-transfer-encoding="base64".`

   - For *ebXML Message Service Version 2.0 Specification*, set the following TPA parameter as shown below:

Set this parameter...	To...
[illegible]	The value of the required character encoding. For example, for base64 character encoding, set the value to `content-transfer-encoding="base64".`

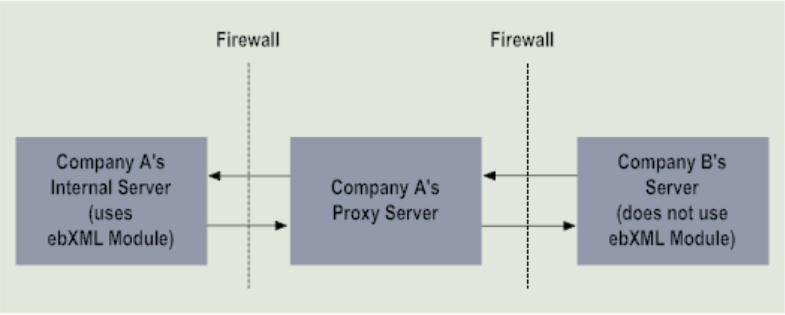
Message Exchange Using a Proxy Server

webMethods ebXML Module uses a proxy server in scenarios where the trading partner does not want to expose the ebXML Module services to another trading partner. The ebXML Module uses webMethods Enterprise Gateway to provide the proxy feature. To configure the Enterprise Gateway Server for a sender that uses ebXML Module, see the *webMethods Integration Server Administrator's Guide* for your release.

For scenarios where the sender requires a proxy server (for example, outside a firewall), ebXML Module sends the outbound message first to the proxy server, which forwards it to the receiver. The sender's server receives an inbound message from the receiver first to the proxy server, which forwards the message to the sender. For the sender and the receiver to communicate with each other using the proxy server, it is necessary to provide the URL of the proxy server to both the sender and the receiver.

In the following diagram, Company A sends an outbound message to Company B through Company A's proxy server. The proxy server picks up the message and forwards the message to Company B's server. Company B responds by sending an outbound message to Company A through the proxy server. To do so, Company A's and Company B's servers should have the Delivery Method set to the URL of the proxy server in the trading partner profile.

Figure 8. Message Exchange Using a Proxy Server



Using the ebXML Module in a Clustered Environment

Clustering is an advanced feature of the webMethods platform that substantially extends the reliability, availability, and scalability of the webMethods Integration Server.

The clustering feature uses a shared cluster store to hold webMethods Integration Server state information and utilization metrics for use in load balancing and automatic failover support. Because this activity is transparent to the client, clustering makes multiple servers look and behave as one.

Note:webMethods Integration Server clustering redirects HTTP and HTTPS requests, but does not redirect SMTP requests.

For details on webMethods Integration Server clustering, see the *webMethods Integration Server Clustering Guide* for your release.

Clustering Considerations and Requirements

The following considerations and requirements apply to the ebXML Module in a clustered environment.

Note:
The following sections assume that you have already configured the webMethods Integration Server cluster. For details about webMethods clustering, see the *webMethods Integration Server Clustering Guide* for your release.

Requirements for Each Integration Server in a Cluster

The following table describes the requirements of each Integration Server in a given cluster:

All Integration Servers in a given cluster musthave identical...	For Example...
Integration Server versions	One Integration Server in the cluster cannot be version 7.1 and another Integration Server in the cluster be version 7.1.1-all servers must be the same version, with the same service packs and fixes (updates) applied.
ebXML Module packages	All ebXML Module packages on one Integration Server should be replicated to all other Integration Servers in the cluster.
ebXML Module versions	One Integration Server in the cluster cannot have ebXML Module Version 7.1 SP1 and another Integration Server in the cluster have ebXML Module Version 6.0.1-all ebXML Module must be the same version, with the same fixes (updates) applied.
ebXML Module services	If you configure a specific ebXML Module service, this same adapter service must appear on all servers in the cluster so that any Integration Server in the cluster can handle the request identically. If you allow different Integration Servers to contain different services, you might not derive the full benefits of clustering. For example, if a client requests a service that resides on only one server, and that server is unavailable, the request cannot be successfully redirected to another server.

Considerations When Installing ebXML Module Packages

For each Integration Server in the cluster, use the standard ebXML Module installation procedures for each machine, as described in the "Installing ebXML Module 7.1 SP1" on page 18.

7 ebXML Module Security Features

- Overview 78
- Configuring Certificates for Secure Messaging 79
- Configuring XML Signature Support 79
- Configuring S/MIME Support 82
- Combining Compression and S/MIME Support 88

- "Overview" on page 78
- "Configuring Certificates for Secure Messaging" on page 79
- "Configuring XML Signature Support" on page 79
- "Configuring S/MIME Support" on page 82
- "Combining Compression and S/MIME Support" on page 88

Overview

webMethods ebXML Module provides the following security features:

- XML signature - to sign the outbound messages, and to verify the authenticity of the inbound messages.
  - DSA and RSA algorithm to sign the outbound messages, and verify the inbound signed messages.
- S/MIME signature and encryption - to sign and encrypt the outbound payload and to verify and decrypt the inbound payload.
  - TripleDES (default), DES, or RC2 encryption algorithm. The values for the length of the encryption key for RC2 encryption are 40, 64, or 128 (default).
  - Standard outbound encryption permutations (signed, encrypted, or signed and encrypted) at the send-service level.
- Sending and receiving synchronous or asynchronous, signed or unsigned messages.
- Signed acknowledgment.

When the XML signature feature is used along with S/MIME signature and encryption, the following processing takes place when you send and receive the ebXML messages:

- For an outbound message, first the XML signature feature generates the XML signature. Then the S/MIME feature signs, encrypts, or signs and encrypts the payload(s).
- For an inbound message, first the S/MIME feature verifies, decrypts, or decrypts and verifies the payload(s). Then the XML signature is verified.

Enabling Secure Messaging

If you want the ebXML Module to be able to send and receive secure messages, you need to:

- **Configure the certificate sets.** For more information, see "Configuring Certificates for Secure Messaging" on page 79.
- **Set the values of the appropriate TPA parameters.** For information about configuring for XML signature support, see "Configuring XML Signature Support" on page 79. For information about configuring for S/MIME support, see "Configuring S/MIME Support" on page 82.

Configuring Certificates for Secure Messaging

Certificate management in webMethods Trading Networks is designed for the "Hub-Spoke" model. That is, you typically have only one host (hub) speaking to several trading partners (spokes). Additionally, some of the samples in the WmebXMLSample package use the multiple hosts model. You need to specify which model to use for certificate management.

To configure the certificates for secure messaging

1. Set the wm.ebxml.certificates.multihosts property in the WmebXML/config/config.cnf file as appropriate to your requirements.

   - For the hub-spoke model, set the value of the `wm.ebxml.certificates.multihosts` property to `false`.
   - For the multi-hosts model, set the value of the `wm.ebxml.certificates.multihosts` property to `true`.

2. Set up the certificates in your profile and in your partner's profile following the instructions in the *webMethods Trading Networks Administrator's Guide* appropriate for your release in the "Document Titles" section in "About this Guide" on page 5.

   You can add certificate sets that Trading Networks uses during processing.

Add this type of certificate set...	To…
Decrypt/Encrypt	Decrypt documents that your Enterprise receives from partners and encrypt documents that Enterprise sends to partners. On the receiver's end, when primary encrypt/decrypt certificate fails to decrypt the inbound payload, the secondary encrypt/decrypt certificate is used if there is one defined in the receiver's profile. The receiver's profile must be active. In such a case, the encrypt/decrypt certificate is used for decryption.
Sign/Verify	Digitally sign documents that your Enterprise sends to partners and verify documents that your Enterprise receives from partners. On the receiver's end, when primary sign/verify certificate fails to verify the inbound payload, the secondary sign/verify certificate is used if there is one defined in the receiver's profile. The receiver's profile must be active. In such a case, the sign/verify certificate is used for verification.

Configuring XML Signature Support

To configure XML signature support, set the appropriate TPA parameter to enable or disable the XML signature/verification for outbound and inbound messages.

To configure the ebXML Module for XML signature support

1. Start the Trading Networks Console.
2. Open the agreement for edit:
   a. From the Trading Networks Console, select **View > Agreements**.
   b. Select the agreement you would like to edit. Click**Edit**.
      The **Agreements Details** screen appears.
   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.
      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.
3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.
4. Enable XML signature for outbound and inbound messages, as follows:
   - For *ebXML Message Service Version 1.0 Specification* messages, set the following TPA parameter as shown below:

Parameter	Description
CollaborationProtocolAgreement.PartyInfo.DeliveryChannelCharacteristics.nonrepudiationOfOrigin	Set the value to `true` to enable the XML signature/verification for outbound and inbound *ebXML Message Service Version 1.0 Specification* messages, else set the value to `false`.

   - For *ebXML Message Service Version 2.0 Specification*, set the following TPA parameter as shown below:

Parameter	Description
CollaborationProtocolAgreementPartyInfoDeliveryChannelCharacteristicsNonRepudiationRequired	Set the value to `true` to enable the XML signature/verification for outbound and inbound *ebXML Message Service Version 2.0 Specification* messages, else set the value to `false`.

5. Set the appropriate TPA parameters for XML signature support, as follows:

   - For *ebXML Message Service Version 1.0 Specification* messages, set the following TPA parameters as shown below:

Parameter	Description
CollaborationProtocolAgreementPartyInfoDeliveryChannelCharacteristicsNonRepudiationOfOrigin	Set the value to `true` if XML signature support is required, else set the value to `false`.
CollaborationProtocolAgreementPartyInfoDocExchangebXMLBindingNonRepudiationProtocol	Set the value to `http://www.w3.org/2000/09/xmldsig#` only if the CollaborationProtocolAgreementPartyInfoDeliveryChannelCharacteristicsNonRepudiationOfOrigin parameter is set to `true`.
CollaborationProtocolAgreementPartyInfoDocExchangebXMLBindingNonRepudiationHashFunction	Set the value to `http://www.w3.org/2000/09/xmldsig#sha1` only if the CollaborationProtocolAgreementPartyInfoDeliveryChannelCharacteristicsNonRepudiationOfOrigin parameter is set to `true`.
CollaborationProtocolAgreementPartyInfoDocExchangebXMLBindingNonRepudiationSignatureAlgorithm	Set the value to `http://www.w3.org/2000/09/xmldsig#rsa-sha1` or `http://www.w3.org/2000/09/xmldsig#dsa-sha1` only if the CollaborationProtocolAgreementPartyInfoDeliveryChannelCharacteristicsNonRepudiationOfOrigin parameter is set to `true`. **Note:** rsa-sha1 and dsa-sha1 are the allowed signature algorithms.

   - For *ebXML Message Service Version 2.0 Specification*, set the following TPA parameters as shown below:

Parameter	Description
[illegible]	Set the value to `true` if XML signature support is required, else set the value to `false`.
[illegible]	Set the value to `http://www.w3.org/2000/09/xmldsig#` only if the [illegible] parameter is set to `true`.
[illegible]	Set the value to `http://www.w3.org/2000/09/xmldsig#sha1` only if the [illegible] parameter is set to `true`.
[illegible]	Set the value to `http://www.w3.org/2000/09/xmldsig#rsa-sha1` or `http://www.w3.org/2000/09/xmldsig#dsa-sha1` only if the [illegible] parameter is set to `true`. **Note:** rsa-sha1 and dsa-sha1 are the allowed signature algorithms.

Configuring S/MIME Support

ebXML Module enables you to sign, encrypt, or sign and encrypt any outbound business document(s) carried in an ebXML message.

When both signature and encryption are required for outbound business documents, the message will be signed first before being encrypted. While defining S/MIME encryption for your message payload, set the encryption types and key lengths for each of your trading partners using the TPA parameters.

To use these facilities, set the appropriate input parameters in the wm.ip.ebxml.MSH:sendusingTPA service or in a TPA as per your requirements.

To configure ebXML Module for S/MIME signed, encrypted, or signed and encrypted messages

1. Start the Trading Networks Console.

2. Open the agreement for edit:

a. From the Trading Networks Console, select **View > Agreements**.

b. Select the agreement you would like to edit. Click**Edit**.

The **Agreements Details** screen appears.

c. On the right side of the **Agreement Details** screen, click **Set Inputs**.

The input for com.wm.estd.ebxml.documents appears.

> **Note:**
> You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.

3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

4. Enable the S/MIME feature and define the encryption algorithm, as follows:

- For *ebXML Message Service Version 1.0 Specification* messages, set the following TPA parameters as shown below:

Parameter	Description
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.Characteristics.isConfidentiality	To enable the S/MIME signing and encryption feature, set the value to `true`.
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.DigitalEnvelope.Protocol.body	To define the encryption mechanism, set the value to `S/MIME`.
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.DigitalEnvelope.Protocol.version	Defines the S/MIME version. Set the value to `2.0`
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLBinding.DigitalEnvelope.EncryptionAlgorithm	Defines the encryption algorithm. Set the value to `tripleDES` (default), `DES`, or `RC2`

- For *ebXML Message Service Version 2.0 Specification*, set the following TPA parameters as shown below:

Parameter	Description
CollaborationProtocolAgreement.PartyInfo.CollaborationRole.ServiceBinding.ActionBinding.BusinessTransactionCharacteristics.isConfidential	To enable the S/MIME signing and encryption feature, set the

Parameter	Description
	value to `persistent` or `transient-and-persistent`.
Collaboration Protocol Agreement Party ID Exchange ebXML Receiving Reliable Digital Envelope Protocol	To define the encryption mechanism, set the value to `S/MIME`.
Collaboration Protocol Agreement Party ID Exchange ebXML Receiving Reliable Digital Envelope Protocol version	Defines the S/MIME version. Set the value to `2.0`
Collaboration Protocol Agreement Party ID Exchange ebXML Receiving Reliable Digital Envelope Encryption Algorithm	Defines the encryption algorithm. Set the value to `tripleDES` (default), `DES`, or `RC2`
Collaboration Protocol Agreement Party ID Exchange ebXML Sending Reliable Digital Envelope Encryption Algorithm minimum length	Defines the encryption key length. For RC2 encryption, set the value to `40`, `64`, or `128`(default).

5. For each ebXML payload attachment, set the S/MIME signature and encryption TPA parameter as shown below:

Set This Parameter...	To This Value...	To Achieve These Results...
Collaboration Protocol Agreement Packaging Composite Encapsulation mime parameters	`smime-type="signed-data"`	Signs but does not encrypt all business documents with S/MIME security feature.
	`smime-type="enveloped-data"`	Encrypts but does not sign all business documents with S/MIME security feature.
	`smime-type="signed-*encrypted-*"`	Signs and encrypts all business documents with S/MIME security feature.
Collaboration Protocol Agreement Packaging Composite Encapsulation mime type	`application/pkcs7-mime`	Enables the S/MIME security feature. **Note:**

Set This Parameter...	To This Value...	To Achieve These Results...
		This parameter must be set in addition to the [illegible] parameter for S/MIME security feature.
CollaborationProtocolAgreementPackagingCompositeListEncapsulationId	Any arbitrary value. For example: `sig+enc_001`	Defines the encapsulation id. **Note:** The value of the [illegible] parameter must refer to the same value defined in this parameter.
CollaborationProtocolAgreementPackagingCompositeListEncapsulationConstituentId	Value of the SimplePart id of the payload. For example: `c_001`	Defines the payload that needs to be encrypted, signed, or signed and encrypted.

6. When a document is configured for S/MIME support, the ebXML Module adds the following entry to the *Manifest* element of the envelope:

```
Manifest/Reference/Schema/location=
http://www.ietf.org/rfc/rfc2311.trxt
Version=2.0
```

7. For an inbound ebXML message, the ebXML Module automatically verifies and/or decrypts any business documents carried in the message.

S/MIME Support - Sample

For example, you perform the following procedures to set the TPA parameters to send XML signed and S/MIME signed and encrypted message and receive signed acknowledgment for *ebXML Message Service Version 2.0 Specification*.

To set the TPA parameters for sending XML signed and S/MIME encrypted messages, and receiving signed acknowledgment

1. Start the Trading Networks Console.

2. Open the agreement for edit:

   a. From Trading Networks Console, select **View > Agreements**.

   b. Select the agreement you would like to edit. Click**Edit**.

      The **Agreements Details** screen appears.

   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.

      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.

3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

4. For signed asynchronous acknowledgment, set the following TPA parameters as shown below:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics: syncReplyMode	None
CollaborationProtocolAgreement.PartyInfo.DeliveryChannel.MessagingCharacteristics.ackRequested	Signed

5. To create an XML signed ebXML Envelope using SHA1 hash algorithm and RSA signature algorithm, set the following TPA parameters as shown below:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.CollaborationRole.ServiceBinding.CanSend. ThisPartyActionBinding.BusinessTransactionCharacteristics: isNonRepudiationRequired	true
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding.SenderNonRepudiation.NonRepudiationProtocol	http://www.w3.org/2000/09/xmldsig#
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding.SenderNonRepudiation.HashFunction	http://www.w3.org/2000/09/xmldsig#sha1
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLSenderBinding.SenderNonRepudiation.SignatureAlgorithm	http://www.w3.org/2000/09/xmldsig#rsa-sha1

6. Set the TPA parameters for S/MIME signature and tripleDES encryption algorithm, as shown below:

Parameter	Value
CollaborationProtocolAgreement.PartyInfo.ActionBinding.BusinessTransactionCharacteristics.isConfidential	`persistent` or `transient-and-persistent` to enable the S/MIME signing and encryption feature
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLReceiveBinding.ReceiveDigitalEnvelope.DigitalEnvelopeProtocol.body	`S/MIME`
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLReceiveBinding.ReceiveDigitalEnvelope.DigitalEnvelopeProtocol.version	`2.0`
CollaborationProtocolAgreement.PartyInfo.DocExchange.ebXMLReceiveBinding.ReceiveDigitalEnvelope.EncryptionAlgorithm.body	`tripleDES`

7. For each payload, set the following TPA parameters as shown below:

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation.id	`sig+enc_001`
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation.mimetype	`application/pkcs7-mime`
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation.mimeparameters	`smime-type="signed-*encrypted-*"`
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation.Constituent.idref	`c_001` **Note:** This parameter value is set assuming that there is an entry in the SimplePart section of the TPA with an id `c_001`.

8. Package the message by setting the following TPA parameter as shown below:

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Composite:id	`xmlSignAndEncComposite`
CollaborationProtocolAgreement.Packaging.CompositeList.Composite.Constituent:idref	`Sample-Envelope`
CollaborationProtocolAgreement.Packaging.CompositeList.Composite.Constituent:idref: excludedFromSignature	`false`
CollaborationProtocolAgreement.Packaging.CompositeList.Composite.Constituent:idref	`sig+enc_001`
CollaborationProtocolAgreement.Packaging.CompositeList.Composite.Constituent.idref.excludedFromSignature	`false`

Combining Compression and S/MIME Support

Apart from indicating the S/MIME security mechanism used, the Encapsulation element can also be used to represent the application of a compression algorithm.

The CompositeList element is required to be defined in the Packaging element when the security encapsulations or composite multiparts are used. The CompositeList child element specifies the way in which the simple parts are combined into groups or encapsulated within the security-related MIME content-types. The content model for the CompositeList element is a repeatable sequence of choices of Composite or Encapsulation elements. The sequence in which the Composite and Encapsulation elements are presented is important because the MIME packaging is done in a recursive way. In addition to the message parts characterized within the SimplePart subtree, the composites or encapsulations can also include any of the previously mentioned composites or encapsulations.

When you want to use a compression algorithm such as zip to compress some part of the payload prior to it being signed, encrypted, or signed and encrypted, set the TPA parameters for each of the ebXML payload attachment as follows.

To configure the ebXML Module for ZIP and S/MIME support

1. Start the Trading Networks Console.

2. Open the agreement for edit:

   a. From the Trading Networks Console, select **View > Agreements**.

   b. Select the agreement you would like to edit. Click**Edit**.

      The **Agreements Details** screen appears.

   c. On the right side of the **Agreement Details** screen, click **Set Inputs**.

      The input for com.wm.estd.ebxml.documents appears.

      Note:
      You can edit the TPA parameters in an agreement only if the **Agreement Status** is **Proposed**. For information about changing the TPA agreement status, see the *webMethods Trading Networks Administrator's Guide* for your release.

3. Modify the default TPA parameters, as necessary. For a list of the TPA parameters and the valid values for individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

4. Define Zip as the compression algorithm.

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:id	`comp_001`
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:mimetype	`application/zip`

5. Define the Constituent idref to the Simple Part with id c_001.

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation.Constituent:idref	`c_001`

6. Define S/MIME signature and encryption security feature to the payload.

Parameter	Value
[illegible]	`persistent` or `transient-and-persistent` to enable the S/MIME signing and encryption feature
[illegible]	`S/MIME`
[illegible]	`2.0`
[illegible]	`tripleDES`
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:id	`sig+enc_001` **Note:** The value of the [illegible] parameter must refer to the value defined in this parameter.
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:mimetype	`application/pkcs7-mime`
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation:mimeparameters	`smime-type="signed-*encrypted-*"`

7. Point the Constituent idref to the Encapsulation id of the element where you defined the compression algorithm.

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Encapsulation.Constituent:idref	`comp_001`

8. Point the idref of the Composite element to the id of the Encapsulation element that defines the S/MIME security feature.

Parameter	Value
CollaborationProtocolAgreement.Packaging.CompositeList.Composite:id	`zipAndSMIMEComposite`
CollaborationProtocolAgreement.Packaging.CompositeList.Composite.Constituent:idref	`Sample-Envelope` **Note:** This parameter value is set assuming that there is an entry in the SimplePart section of the TPA with an id `Sample-Envelope`, which represents the ebXML envelope.
CollaborationProtocolAgreement.Packaging.CompositeList.Composite.Constituent:idref	`sig+enc_001`

A Built-In Services

- "Summary of Elements" on page 91
- "CPA Folder (wm.ip.ebxml.cpa)" on page 92
- "MSH Folder (wm.ip.ebxml.MSH)" on page 94
- "TN Folder (wm.ip.ebxml.TN)" on page 103
- "util Folder (wm.ip.ebxml.util)" on page 105

Summary of Elements

The following elements are available in this folder:

Element	Package and Description
CPA Folder (wm.ip.ebxml.cpa)	
wm.ip.ebxml.cpa:exportCPA	Exports the contents of a Trading Partner Agreement in the format of a CPA document.
wm.ip.ebxml.cpa:importCPA	Creates and configures two Trading Networks profiles and a TPA from a CPA.
wm.ip.ebxml.cpa:initTPA1	Populates the TPA with default values for *ebXML Message Service Version 1.0* Specification.
wm.ip.ebxml.cpa:initTPA2	Populates the TPA with default values for *ebXML Message Service Version 2.0* Specification.
MSH Folder (wm.ip.ebxml.MSH)	
wm.ip.ebxml.MSH:pingUsingTPA	Provides the MSH Ping service as specified in the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. Enables one MSH to determine whether another MSH is operating.
wm.ip.ebxml.MSH:receive	Provides the entry point for all incoming ebXML messages. Submit all received ebXML messages to this service. This service has no output parameters.
wm.ip.ebxml.MSH:routeUsingTPA	Routes a TN document type such as Default, Payload, Receipt to the next MSH, using intermediary parties, optionally with additional payloads.
wm.ip.ebxml.MSH:sendAck	Sends an acknowledgment for a given BizDocEnvelope.

Element	Package and Description
wm.ip.ebxml.MSH:sendBizDoc	Sends the content parts of an existing BizDocEnvelope object as an ebXML message. This service must be used with a TPA.
wm.ip.ebxml.MSH:sendErrorMsg	Generates and sends an error message for a specific BizDocEnvelope object.
wm.ip.ebxml.MSH:sendReceipt	Sends a receipt for any ebXML TN document type. This service is supported only with ebXML Messaging Services protocol version 1.0.
wm.ip.ebxml.MSH:sendUsingTPA	The send service that uses a TPA.
wm.ip.ebxml.MSH:statusUsingTPA	Provides the Message Status Request service as specified in the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.
TN Folder (wm.ip.ebxml.TN)	
wm.ip.ebxml.TN:getTNConversationId	Creates a Trading Networks conversation ID for an ebXML message.
wm.ip.ebxml.TN:getPayloads	Retrieves the business documents carried in the ebXML message.
util Folder (wm.ip.ebxml.util)	
wm.ip.ebxml.util:migrateTPA	Migrates the ebXML Module 6.0.1 TPA to the ebXML Module 7.1 TPA. Also creates a backup of the existing 6.0.1 TPA with the agreementId suffixed by `_6-0-1`.

Note:
Some of the services that existed in webMethods ebXML Module 6.0.1 have been removed to comply with the *ebXML Message Service Version 2.0 Specification*, which mandates the use of TPAs.

The ebXML Module also uses services in the WmEstdCommonLib package. For information about these services, see the *webMethods eStandards Modules Common Built-In Services Reference* .

CPA Folder (wm.ip.ebxml.cpa)

wm.ip.ebxml.cpa:exportCPA

Exports the contents of a Trading Partner Agreement in the format of a CPA document.

Input Parameters

senderID **String** The internal ID for the message sender.

senderIDType	**String** The ID type of the sender.
receiverID	**String** The internal ID for the message receiver.
receiverIDType	**String** The ID type of the receiver.
agreementID	**String** The agreementID of the TPA.

Output Parameters

cpaString	**String** The contents of the exported CPA.

wm.ip.ebxml.cpa:importCPA

Creates and configures two Trading Networks profiles and a TPA from a CPA.

Input Parameters

cpaString	**String** The contents of the CPA to use to generate the Trading Networks profiles and TPA.
updateExistingProfiles	**String** (optional) Indicates whether to overwrite any existing profiles that have the same external IDs as those in the CPA. Specify either `true` or `false`. The default value is `false`.
updateExistingTPA	**String** (optional) Indicates whether to overwrite any existing TPA that has the same senderID, receiverID, and agreement ID as those in the CPA. Specify either `true` or `false`. The default value is `false`.

Output Parameters

TPA	**Document** The TPA created from the CPA.
error	**String** The error information.

wm.ip.ebxml.cpa:initTPA1

Populates the TPA with default values for *ebXML Message Service Version 1.0 Specification*.

Input Parameters

None.

Output Parameters

tpaData (cpa_1)	Refers to cpa_1

wm.ip.ebxml.cpa:initTPA2

Populates the TPA with default values for *ebXML Message Service Version 2.0 Specification*.

Input Parameters

None.

Output Parameters

tpaData (cpa_2)	Refers to cpa_1

MSH Folder (wm.ip.ebxml.MSH)

wm.ip.ebxml.MSH:pingUsingTPA

Provides the MSH Ping service as specified in the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*. Enables one MSH to determine whether another MSH is operating.

This service constructs an ebXML Message and submits it to the internal queue as a task to be processed.

Input Parameters

To	**Document** The destination to which to send the Ping message.	
	ID	**String** A unique alphanumeric identifier of the receiver.
	IDType	**String** The ID type of the receiver.
From	**Document** (optional) The sender of the Ping message.	
	ID	**String** The identification number of the sender.
	IDType	**String** The ID type of the sender.
CPAId	**String** A valid agreementID for a TPA.	

Output Parameters

messageId	**String** The message ID for the ebXML message.	
errorMsg	**Document** Error information.	
	errorCode	**String** The ebXML error code.
	severity	**String** (Optional) The error severity. It can have two values, either `Error` or `Warning`.
	location	**String** (Optional) The XPATH of the error location.
	errorMessage	**String** (Optional) The error message.
	codeContext	**String** The context of the error message.

wm.ip.ebxml.MSH:receive

Provides the entry point for all incoming ebXML messages. Submit all received ebXML messages to this service. This service has no output parameters.

Input Parameters

contentStream	**Document** Input stream that contains the ebXML Message.
node	**Document** The node that contains the ebXML SOAP Envelope.

wm.ip.ebxml.MSH:routeUsingTPA

Routes a TN document type such as Default, Payload, Receipt to the next MSH, using intermediary parties, optionally with additional payloads.

This service constructs an ebXML Message and submits it to the internal queue as a task to be processed.

Input Parameters

bizdoc	**Object** A BizDocEnvelope object that uses the ebXML Routing TN document type. For information on the sub-parameters of bizdoc, see the *webMethods Trading Networks Administrator's Guide* for your release.	
Payloads	**Document list** (optional) New business document to forward to the next MSH.	
	partID	**String** The packaging ID for wrapping this document. It must be identical to one of

	CollaborationProtocolAgreement.Packaging.id in com.wm.estd.ebxml.documents.CPA1.cpa_1 (for CPA 1.0)and CollaborationProtocolAgreement.Packaging.id in com.wm.estd.ebxml.documents.CPA1.cpa_2 (for CPA 2.0)
stream	**InputStream** A business document in Java.io.InputStream form.

Output Parameters

messageId	**String** The message ID for the ebXML message.
errorMsg	**Document** Error information.
errorCode	**String** The ebXML error code.
severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
location	**String** The XPATH of the error location.
errorMessage	**String** The error message.
codeContext	**String** The context of the error message.

wm.ip.ebxml.MSH:sendAck

Sends an acknowledgment for a given BizDocEnvelope.

Input Parameters

bizdoc	**Object** The BizDocEnvelope object that contains an ebXML message.
ackType	**String** (Optional) Specify `signed` for a signed acknowledgment and `Unsigned` for an unsigned acknowledgment.
actor	**String** (Optional) In a multi-hop scenario, indicates forwarding of the message acknowledgment to the specified MSH. Specify one of the following: ■ Select `nextMSH` to send the acknowledgment to the MSH of the next party in the routing. ■ Select `toPartyMSH` to send the acknowledgment to the MSH of the sending party of the original message. For more information about using this parameter, see the following section.

Output Parameters

messageId	**String** The message ID for the ebXML message.	
errorMsg	**Document** Error information.	
	errorCode	**String** The ebXML error code.
	severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
	location	**String** The XPATH of the error location.
	errorMessage	**String** The error message.
	codeContext	**String** The context of the error message.

Usage Notes

- **Sending message acknowledgments to the sending MSH.** Consider using the *actor* parameter in a multi-hop scenario where a receiver MSH can acknowledge the receipt of a message to the previous MSH or to the original sender MSH. To send an acknowledgment to the previous MSH, set the *actor* parameter to `nextMSH`. To send an acknowledgment to the original sender MSH, set the *actor* parameter to `toPartyMSH`.

wm.ip.ebxml.MSH:sendBizDoc

Sends the content parts of an existing BizDocEnvelope object as an ebXML message. This service must be used with a TPA.

Input Parameters

bizdoc	**Object** The BizDocEnvelope object that contains the business document that you want to send. The DocumentID of the *bizdoc* is used as the MessageId for the ebXML message.	
Packaging	**Document list** (optional) The packaging information.	
	partName	**String** The name of the content part in the *bizdoc* object that you want to send.
	partID	**String** The packaging ID for wrapping this content part. It must be one of CollaborationProtocolAgreement.Packaging.SimplePart.id in com.wm.estd.ebxml.documents.CPA1.cpa_1 (for CPA 1.0)and CollaborationProtocolAgreement.SimplePart.id in com.wm.estd.ebxml.documents.CPA1.cpa_2 (for CPA 2.0)
CPAId	**String** The CPA ID. It must be a valid agreementID for a TPA.	

ConversationId	**String** (optional) The conversation ID. If it is not present, the conversation ID in the *bizdoc* will be used.
action	**String** Specifies the action to be executed for any instance of the *bizdoc*. This value must correspond to one of the action values specified in the TPA. By default, the first action specified by the TPA on the sender side is used.

Output Parameters

messageId	**String** The message ID for the ebXML message.	
errorMsg	**Document** Error information.	
	errorCode	**String** The ebXML error code.
	severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
	location	**String** The XPATH of the error location.
	errorMessage	**String** The error message.
	codeContext	**String** The context of the error message.

wm.ip.ebxml.MSH:sendErrorMsg

Generates and sends an error message for a specific BizDocEnvelope object.

Input Parameters

errors	**Document** The error information.	
	errorCode	**String** (Optional) The ebXML error code. This must conform to the ebXML Message Service specification.
	severity	**String** (Optional) The error severity. This parameter can have two values, either `Error` or `Warning`. This must conform to the ebXML Message Service specification.
	location	**String** (Optional) The XPATH of the error location. This must conform to the ebXML Message Service specification.

	errorMessage	**String** (Optional) The error message. This must conform to the ebXML Message Service specification.
	codeContext	**String** The context of the error message. This must conform to the ebXML Message Service specification.
bizdoc	**Object** A BizDocEnvelope object that uses any ebXML TN document type.	

Output Parameters

messageId	**String** The message ID for the ebXML message.	
errorMsg	**Document** Error information.	
	errorCode	**String** The ebXML error code.
	severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
	location	**String** The XPATH of the error location.
	errorMessage	**String** The error message.
	codeContext	**String** The context of the error message.

wm.ip.ebxml.MSH:sendReceipt

Sends a receipt for any ebXML TN document type. This service is supported only with ebXML Messaging Services protocol version 1.0.

Input Parameters

bizdoc	**Object** A BizDocEnvelope object that uses any ebXML TN document type.

Output Parameters

messageId	**String** The message ID for the ebXML message.	
errorMsg	**Document** Error information.	
	errorCode	**String** The ebXML error code.

severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
location	**String** The XPATH of the error location.
errorMessage	**String** The error message.
codeContext	**String** The context of the error message.

wm.ip.ebxml.MSH:sendUsingTPA

The send service that uses a TPA.

This service constructs an ebXML Message and submits it to the internal queue as a task to be processed. If the required TPA is not present, an exception is thrown.

At the receiver end, a payload processing service can be invoked in one of the following two ways:

- Setting thewm.ebxml.payloadProcessSvc property

  This property is defined in the `IntegrationServer_directory`\packages\WmebXML\config\config.cnf file. The service assigned as the value of the wm.ebxml.payloadProcessSvc property will be invoked to process the payloads.

- Setting the corresponding parameters in the TPA

  In the sender party section of the TPA, set the value of CollaborationProtocolAgreement/PartyInfo/CollaborationRole/ServiceBinding/Service/*body and CollaborationProtocolAgreement/PartyInfo/CollaborationRole/ServiceBinding/Service/type parameters as "webMethods".

The payload processing service should follow the specifications wm.ip.ebxml.rec:payloadProcess for ebMS 1.0 and wm.ip.ebxml.rec:payloadProcess_v2for ebMS 2.0. These specifications indicate available inputs for the implemented service.

Input Parameters

To	**Document** Information about the receiver.	
	ID	**String** The receiver's ID.
	IDType	**String** The receiver's ID type.
From	**Document** Information about the sender. If it is absent, this service uses information about the host.	
	ID	**String** The sender's ID. If it is absent, this service uses the host's ID.

	IDType	**String** The sender's ID type. If it is absent, this service uses the host's ID type.
CPAId		**String** The CPA ID. It must be a valid agreementID for a TPA.
ConversationId		**String** (optional) The conversation ID.
MessageId		**String** (optional) Used as the message ID for the message, if specified. It should be unique within the context of the message exchange between two trading partners.
RefMessageId		**String** (optional) The message ID of a message to which this message refers.
Payloads		**Document list** (optional) Any business document to be sent.
	partID	**String** The package ID for wrapping this document. It must be one of CollaborationProtocolAgreement.Packaging.SimplePart.id in com.wm.estd.ebxml.documents.CPA1.cpa_1 (for CPA 1.0)and CollaborationProtocolAgreement.SimplePart.id in com.wm.estd.ebxml.documents.CPA1.cpa_2 (for CPA 2.0)
	stream	**InputStream** A business document in Java.io.InputStream form.
action		**String** (optional) Specifies the action to be executed for any instance of the bizdoc. This value must correspond to one of the action values specified in TPA. By default, the first action specified by the TPA on the sender side is used. If you do not specify an action, ebXML Module uses the first action within the first service specified in the TPA. If you specify an action but you do not supply a value for *service*, the action is assumed to be within the first service specified in the TPA.
service		**String** (optional) Specifies the service to be executed for any instance of the bizdoc. This value must correspond to one of the service values specified in the TPA. If you do not specify a service, ebXML Module uses the first service specified in the TPA.

Output Parameters

messageId		**String** The message ID for the ebXML message.
errorMsg		**Document** Error information.
	errorCode	**String** The ebXML error code.

severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
location	**String** The XPATH of the error location.
errorMessage	**String** The error message.
codeContext	**String** The context of the error message.

wm.ip.ebxml.MSH:statusUsingTPA

Provides the Message Status Request service as specified in the *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification*.

This service constructs an ebXML Message and submits it to the internal queue as a task to be processed.

This service performs the following:

- Sends a Message Status Request message to an MSH
- Responds to a received Message Status Request message with a Message Status Response message

This service requires a valid TPA.

Input Parameters

To	**Document** The host of the message whose status you are requesting.	
	ID	**String** The identification number of the destination.
	IDType	**String** The ID type of the destination.
From	**Document** The sender of the message whose status you are requesting.	
	ID	**String** The identification number of the sender.
	IDType	**String** The ID type of the sender.
CPAId	**String** A valid agreementID for a TPA.	
MessageID	**String** The document ID of the message whose status you are requesting.	

Output Parameters

messageId	**String** The message ID for the ebXML message.
errorMsg	**Document** Error information.

errorCode	**String** The ebXML error code.
severity	**String** The error severity. It can have two values, either `Error` or `Warning`.
location	**String** The XPATH of the error location.
errorMessage	**String** The error message.
codeContext	**String** The context of the error message.

TN Folder (wm.ip.ebxml.TN)

wm.ip.ebxml.TN:getPayloads

Retrieves the business documents carried in the ebXML message.

Input Parameters

bizdoc	**Object** An ebXML BizDocEnvelope object.
Envelope	**Document** (optional) The ebXML Envelope within the bizdoc object.

Output Parameters

Payloads **Document list** Payload(s) to be sent via the message.

mimeHeader	**Document** The MIME header for each payload in a multipart MIME message.
Content-ID	**String** The message ID in the MIME header.
Content-Type	**String** The content-type string in the MIME header.
stream	**Object** A java.io.InputStream object for each payload.
inSignature	**String** Whether the payload object should be signed in the *Signature* element. Possible values are `true` and `false`.
inSMIME	**String** Whether the payload object should be an SMIME object. Possible values are `true` and `false`.
compression	**String** Whether the payload object should be compressed using zip compression. Possible values are `true` and `false`. When set to `true`, the payload is compressed using zip

compression; when set to `false`, the payload is not compressed.

Manifest **Document** The purpose of the Manifest is as follows: to make it easier to directly extract a particular payload associated with this message, and to allow another application to determine whether or not it can process the payload without having to parse it.

role **String** (optional) Identifies some resource that describes the payload object or its purpose. This parameter must have a value that is a valid URI in accordance with the XLINK specification.

Schema **Document list** (optional) Provides a means of identifying a schema (if present) that defines the referenced payload object.

location **String** The URI of the schema.

version **String** (optional) A version identifier of the schema.

Description **Document** (optional) A textual description of the payload object referenced by the Reference element.

text **String** The textual description.

language **String** The language of the description; the value must comply with the rules for identifying languages specified in XML.

wm.ip.ebxml.TN:getTNConversationId

Creates a Trading Networks conversation ID for an ebXML message.

Input Parameters

cid	**String** The conversation ID defined in an ebXML message.
receiverId	**String** The internal ID for the message receiver.
SenderID	**String** (optional) The internal ID for the message sender.

Output Parameters

tnConversationId	**String** The Trading Networks conversation ID.

util Folder (wm.ip.ebxml.util)

wm.ip.ebxml.util:migrateTPA

Migrates the ebXML Module 6.0.1 TPA to the ebXML Module 7.1 TPA. Also creates a backup of the existing 6.0.1 TPA with the agreementId suffixed by `_6-0-1`. For example, if the agreementId of the 6.0.1 TPA is `Sample`, the agreementID of the backed up TPA will be `Sample_6-0-1`.

Input Parameters

senderId	**String** The external ID for the message sender.
senderIdType	**String** The sender's ID type. If it is absent, this service uses the host's ID type.
receiverId	**String** The external ID for the message receiver.
receiverIdType	**String** The receiver's ID type. If it is absent, this service uses the host's ID type.
agreementId	**String** A valid agreementID for a TPA.

The wm.ip.ebxml:migrateTPA service does not migrate the following TPA parameters of the ebXML Module 6.0.1:

For this parameter in ebXML Module 6.0.1...	Use this parameter in ebXML Module 7.1
tpaDataSchema.OtherParams.AutoReceiptSend	deliveryReceiptRequested
tpaDataSchema.OtherParams.PayloadProcessSvc	ServiceBinding:Service
tpaDataSchema.OtherParams.MimebodyEncoding	SimplePart[0]:mimeparameters
tpaData.OtherParams.OtherDeliveryChannel	Not supported in ebXML Module 7.1 SP1

For this parameter in ebXML Module 6.0.1...	Use this parameter in ebXML Module 7.1
tpaData.OtherParams.UserDefined	Not supported in ebXML Module 7.1 SP1
tpaData.OriginalCPA	Not supported in ebXML Module 7.1 SP1. If you require the TPA in a CPA format, invoke the wm.ip.ebxml.cpa:exportCPA service.

Note:
After executing the wm.ip.ebxml:migrateTPA service, for the reliable messaging feature to work, you must set the values for Retries and RetryInterval parameters for both *ebXML Message Service Version 1.0 Specification* and *ebXML Message Service Version 2.0 Specification* messages. For a list of the TPA parameters and the valid values for the individual parameters, see "Trading Partner Agreement Parameters Version 2. 0" on page 119 and "Trading Partner Agreement Parameters Version 2. 0" on page 119.

B Trading Partner Agreement Parameters Version 1. 0

- "Overview" on page 107
- "TPA Parameters for ebXML Message Service Version 1.0 Specification" on page 107

Overview

The following section provides an overview of the Trading Partner Agreement (TPA) parameters for *ebXML Message Service Version 1.0 Specification* and the default parameter values. The parameters are listed in tables with the respective description for each parameter and its sub-parameter(s). For better visualization, some parent parameters are represented in separate tables by means of cross-references.

TPA Parameters for ebXML Message Service Version 1.0 Specification

The following table lists and describes the TPA parameters for *ebXML Message Service Version 1.0 Specification* . This table helps you to create a TPA manually.

CollaborationProtocolAgreement

Parameters

cpaid	A string that gives the document a unique identifier.
version	A string that is the version of the TPA. It can only have a value of 1.0.
schemaLocation	A URI string that specifies the location of the schema document used to validate the document.
Status	An IS document (IData object) that indicates the state of the process that creates the CPA. For a list of sub-parameters and their descriptions, see "Status" on page 108.
PartyInfo	An IS document (IData object) that describes the messaging capabilities of a trading partner identified by the "PartyId" document. For a list of sub-parameters and their descriptions, see "PartyInfo" on page 108.
Packaging	An IS document (IData object) that provides specific information about how the Message Header and payload constituent(s) are packaged for

	transmittal over the transport, including information about document-level security packaging security features. For a list of sub-parameters and their descriptions, see "Packaging" on page 112.
Routing	An IS document (IData object) used for routing configuration in a multi-hop messaging scenario. For a list of sub-parameters and their descriptions, see "Routing" on page 113.
CustomParams	An IS document (IData object) that contains webMethods implementation-specific parameters. For a list of sub-parameters and their descriptions, see "CustomParams" on page 114.

Status

value

A string that records the current state of composition of the CPA. As per the CPP-CPA 1.0 specification, it can have three possible values: `proposed`, when the CPA is still being negotiated by the Parties; `agreed`, when the contents of the CPA is accepted by both Parties; and `signed`, when the CPA is signed by the Parties. webMethods ebXML Module

PartyInfo

PartyId

An IS document (IData object) that provides Party identification.

Parameter	Description
body	A string that provides a unique identifier for the Party.
type	A URI string that defines a namespace for the value of the PartyId.

CollaborationRole

An IS document (IData object) that identifies a role for the Party.

Parameter	Description
Role	An IS document (IData object) that identifies the role that the Party can support.

Parameter	Description
	■ **name**A string that is the name attribute of Role.
ServiceBinding	An IS document (IData object) that provides service binding information for the message receiver.
channelId	An IDREF string that refers to the default DeliveryChannel for message delivery.
packageId	An IDREF string that refers to the default Packaging element to be used with the ServiceBinding element.
Service	An IS document (ID object) that identifies the service bound to a Role element. Parameters: ■ **body** A string that is the name of the service to be bound. ■ **type** A string that is a type attribute for Service.
Override	An IS document (IData object) that specifies delivery and packaging information for actions that need channelId and packaging other than the default values specified by ServiceBinding/channelId and ServiceBinding/packaging. Parameters: ■ **action**A string that is the action the message receiver takes. ■ **channelId**A string that is the ID for the DeliveryChannel used for the message identified in action. ■ **packageId**A string that is the ID for the Packaging used for the message identified in action.

DeliveryChannel

An IS document (IData object) that defines the characteristics of each channel used for message delivery.

Parameter	Description
channelId	A string that is the ID attribute for a DeliveryChannel.
transportId	A string that identifies the Transport element for a DeliveryChannel. Its value equals that of the transportId value of the identified Transport element.
docExchangeId	A string that identifies the DocExchange element for a DeliveryChannel. Its value equals that of the docExchangeId value of the identified DocExchange element.
Characteristics	An IS document (IData object) that contains security definitions for a delivery channel.

Parameter	Description
	■ **syncReplyMode**A string that defines the content of a synchronous reply. It can have one of four values: `none`, `responseOnly`, `signalsAndResponse`, and `signalsOnly`. When the value is set to none, the reply is asynchronous. At present, ebXML Module interprets the values of `responseOnly`, `signalsAndResponse`, and `signalsOnly` as synchronous, and does not specifically handle each of these values. ■ **nonrepudiationOfOrigin** A Boolean string than can have one of two values: `true` for requesting a digital signature from the message sender, and `false` for requesting no digital signature. A value of `true` results in a digest comparison of the received acknowledgment with the original message. ■ **ackRequested** A string that can have one of three values: `Signed` for requesting a transport-level signed acknowledgment, `Unsigned` for requesting a transport-level unsigned acknowledgment, and `None` for requesting no transport-level acknowledgment.

Transport

An IS document (IData object) that defines communication protocols for specific ebXML message actions. Supported communication protocols are HTTP, HTTPS, and SMTP.

Parameter	Description
transportId	A string that is the ID attribute for each Transport element.
SendingProtocol	An IS document that identifies the communication protocol that the sender uses to send an ebXML message to its intended receiver. The SendingProtocol:body parameter can be set to one of the following values: `Preferred`, `Primary HTTP`, `Secondary HTTP`, `Primary HTTPS`, `Secondary HTTPS`, `Primary Email`, or `Secondary Email`. Each of these values correspond to a delivery method defined in the profile of the receiver.
ReceivingProtocol	An IS Document that identifies the communication protocol using which a Party can receive ebXML messages from the other Party.
Endpoint	An IS document (IData object) that contains the address of the message receiver. ■ **uri** A URI string that is the electronic address of the message receiver. ■ **type** A string that identifies the purpose of the Endpoint and can have one of five values: `login` for the initial message, `request` for

Parameter	Description
	request messages, `response` for response messages, `error` for error messages, and `allPurpose` for any messages.

DocExchange

An IS document (IData object) that contains information related to document exchange.

Parameter	Description
docExchangeId	A string that is the ID used to reference this element from other parts of the CPA.
ebXMLBinding	An IS document (IData object) that contains specific properties of the *ebXML Message Service Version 1.0 Specification*. ■ **version** A string that is the version of the ebMS specification. ■ **ReliableMessaging** An IS document that contains information about reliable ebXMl message exchange. ■ **deliverySemantics** A String that can have one of two values: `OnceAndOnly Once` for reliable delivery, and `BestEffort` for normal delivery. ■ **deliveryReceiptRequested** A String that can have one of three values: `Signed` for a signed receipt, `Unsigned` for an unsigned receipt, and `None` for no receipt. Applicable only for *ebXML Message Service Version 1.0 Specification*. ■ **messageOrderSemantics** A string that can have one of two values: `Guaranteed` for processing the messages in the order in which they were sent, and `Not Guaranteed` for not following the order in which they were sent. ■ **Retries** A string that specifies the number of retries that are permitted. ■ **RetryInterval** A string that specifies in seconds the interval between retries after a timeout. webMethods ebXML Module ■ **PersistDuration** A string that provides the length of time (in seconds). In case of nonreliable messaging, this parameter is used for computing the value of the **TimeToLive** element. ■ **NonRepudiation** An IS document (IData object) that contains information about the XML Digital Signature of the message.

Parameter	Description
	■ **Protocol** An IS document (ID object) that contains information about the technology used for the digital signature. ■ **body** A string that is the name of the technology used for the digital signature. ■ **HashFunction** A string that specifies the algorithm used for the message digest. ■ **SignatureAlgorithm** A string that specifies the algorithm used for the digital signature. ■ **DigitalEnvelope** An IS document (IData object) that contains encryption options for symmetric encryption. ■ **Protocol** An IS document (ID object) that defines the security protocol that is used. ■ **body** A string that is the name of the security protocol. ■ **EncryptionAlgorithm** A string that specifies the encryption algorithm.

Packaging

id

A string that is the ID used to reference this element from other parts of the CPA.

SimplePart

An IS document (IData object).

Parameter	Description
id	A string that is the Content-ID for the MIME part. This ID is also used to match a particular payload to a packaging specification. This ID must be unique within the Packaging object.
mimetype	A string that is the Content-Type for the MIME part.
mimeparameters	A string that specifies the payload encoding.
NamespaceSupported	An IS document that identifies the schema for the MIME part.
location	A URI for retrieval of the schema associated with the namespace.
version	A version value for the namespace.

CompositeList

An IS document (IData object) that contains security encapsulation and MIME multiparts.

Parameter	Description
Encapsulation	An IS document (IData object) that contains information about MIME security options. ■ **id** A string that is the ID for the MIME part security encapsulation. This ID can be referenced from a later element in the sequence. ■ **mimetype** A string that is the Content-Type for the MIME part. ■ **mimeparameters** A string that contains other parameters for Content-Type. ■ **Constituent** An IS document (IData object) that identifies the contents and order of payloads. ■ **idref** A string that is the ID of a previous element in the sequence or a SimplePart id.

Composite	An IS document (IData object) that contains information about MIME multiparts.
	■ **id** A string that is the ID for the MIME part security encapsulation. This ID can be referenced from a later element in the sequence.
	■ **mimetype** A string that is the Content-Type for the MIME part.
	■ **mimeparameters** A string that contains other parameters for Content-Type.
	■ **Constituent** An IS document (IData object) that identifies the contents and order of payloads.
	  ■ **idref** A string that is the ID of a SimplePart or Encapsulation.
	  ■ **excludedFromSignature** A string that indicates whether this payload is signed by the XML Signature. It can have a value of `true` or `false`.

Routing

previousMSH

The previous MSH from which the message is sent.

Parameter	Description
ids	A string that provides a unique identifier for the previousParty MSH.
idTypes	A string that is a type attribute for ids.

nextMSH

The next MSH to which the message should be sent.

Parameter	Description
ids	A string that provides a unique identifier for the next Party MSH.
idTypes	A string that is a type attribute for ids.

senderMSH (Message Type)

An IS document (IData object) that identifies the original MSH sender.

Parameter	Description
ids	A string that provides a unique identifier for the sender Party MSH.
idTypes	A string that is a type attribute for ids.

receiverMSH (Message Type)

An IS document (IData object) that identifies the original MSH receiver.

Parameter	Description
ids	A string that provides a unique identifier for the receiver Party MSH.
idTypes	A string that is a type attribute for ids.

CustomParams

AutoAckSend

Enables or disables the sending of automatic acknowledgments.

- Set to `false` to disable the sending of automatic acknowledgments when messages are processed successfully. You can explicitly send acknowledgments using the wm.ip.ebxml.MSH:sendAck service.
- Set to `true` to enable webMethods ebXML Module

Automatic acknowledgments can also be controlled with the webMethods ebXML Module

payload\description

Specifies descriptive text for the payload to be included in the MessageHeader element of a SOAP message. The ID field in this parameter refers to the payload ID in the SimplePart section of the TPAs.

timestampFormat

A string that specifies the timestamp format in the ebXML message.

- Set this property to `default1` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:SSS'Z' (for example, 2008-05-20T13:20:10.663Z). This is the default setting.
- Set this property to `default2` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:mm (for example, 2008-05-220T13:20:10).
- Set this property to `default3` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:mm, where yyyy-MM-dd'T'HH:mm:ss is the local time, and -HH:mm is the time zone offset. (For example, "2008-05-20T13:20:10-05:00" represents "20 May 2008 13:20:10 GMT-05:00".)
- Set this property to any other valid Java timestamp format.

> **Note:**
> For more information about the timestamp formats above, see the *XML Schema Part 2: Datatypes* document available at http://www.w3.org/TR/xmlschema-2

timezone

A string that specifies the timezone in the ebXML message.

- Set this property to `GMT` to use the GMT timezone. This is the default setting.
- Set this property to `local` to use the timezone set on your local machine.
- Set this property to any other valid Java timezone format.

Default Parameter Values

In Trading Networks, when you specify the initialization service (wm.ip.ebxml.cpa:initTPA1), the TPA IS document type parameters listed in the following table are populated with the default values indicated.

Parameter Name	Default Value
version	`1.0`

Parameter Name	Default Value
schemaLocation	`http://www.ebxml.org/namespaces/tradePartner http://ebxml.org/project_teams/trade_partner/cpp-cpa-v1_0.xsd`
Status	`agreed`
PartyInfo.CollaborationRole.ServiceBinding:channelId	`DC1`
PartyInfo.CollaborationRole.ServiceBinding:packageId	`PKG1`
PartyInfo.DeliveryChannel:channelId	`DC1`
PartyInfo.DeliveryChannel:docExchangeId	`DX1`
PartyInfo.DeliveryChannel.Characteristics:syncReplyMode	`none`
PartyInfo.DeliveryChannel.Characteristics:nonrepudiationOfOrigin	`False`
PartyInfo.DeliveryChannel.Characteristics:ackRequested	`None`
PartyInfo.DocExchange:docExchangeId	`DX1`
PartyInfo.DocExchange.ebXMLBinding:version	`1.0`
PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging:deliverySemantics	`OnceAndOnlyOnce`
PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging:deliveryReceiptRequested	`None`
PartyInfo.DocExchange.ebXMLBinding.ReliableMessaging:messageOrderSemantics	`NotGuaranteed`
Packaging:id	`PKG1`
Packaging.SimplePart:id	`SP1`
Packaging.SimplePart.mimetype	`text/xml`
Packaging.CompositeList.Composite:id	`C1`
Packaging.CompositeList.Composite.Constituent:idref	`SP1`

Note:
PartyInfo parameter settings are applicable to both sender and receiver parties.

Ensure the following points, when you create the TPA structure manually:

- The value of PartyInfo.DeliveryChannel:docExchangeId parameter is the same as the PartyInfo.DocExchange:docExchangeId parameter, in this case, DX1.
- The value of Packaging.CompositeList.Composite.Constituent:idref parameter is the same as the Packaging.SimplePart:id parameter, in this case, SP1.
- The values of PartyInfo.CollaborationRole.ServiceBinding:channelId parameter and PartyInfo.CollaborationRole.ServiceBinding:packageId parameter are the same as the

PartyInfo.DeliveryChannel:channelId parameter and Packaging:id parameter respectively, in this case DC1 and PKG1 respectively.

To configure the TPA manually, for exchange of payload messages, you need to configure the following parameters:

Parameter	Default Value
PartyInfo.CollaborationRole.ServiceBinding.Service:body	`util:testService`
PartyInfo.CollaborationRole.ServiceBinding.Service:type	`webMethods`
PartyInfo.CollaborationRole.ServiceBinding.Override:action	`testAction`
PartyInfo.CollaborationRole.ServiceBinding.Override:channelId	`DC1`
PartyInfo.CollaborationRole.ServiceBinding.Override:packageId	`PKG2`
Packaging:id	`PKG2`
Packaging.SimplePart:id	`SP2`
Packaging.SimplePart:mimetype	`Application/xml`
Packaging.CompositeList.Composite.Constituent[0]:idref	`SP1`
Packaging.CompositeList.Composite.Constituent[1]:idref	`SP2`

To create the TPA structure manually for message exchange with payloads, configure the following parameters:

1. **Define the Service Element**. Specify a value for the PartyInfo.CollaborationRole.ServiceBinding.Service:body parameter. This value is used as the value of the Service element, ebXML Message Header of the payload messages.

   For example, if the value of PartyInfo.CollaborationRole.ServiceBinding.Service:type is `webMethods`, the ebXML Module interprets the service as a webMethods Integration Server service and invokes it to process the message payloads.

2. **Define the Action Element.** Specify a value for PartyInfo.CollaborationRole.ServiceBinding.Override:action parameter. This value is used as the value of the Action element in the ebXML Message Header of the payload message.

3. **Associate the Delivery Channel and Packaging for the Payload Message Exchange**. Specify values for PartyInfo.CollaborationRole.ServiceBinding.Override:channelId and PartyInfo.CollaborationRole.ServiceBinding.Override:packageId parameters.

   If the messaging requirements specified in the TPA for the payload message is not met by the DocExchange element DX1 associated with the default deliveryChannel DC1, you have to define a new DocExchange element say DX2 and a corresponding deliveryChannel DC2. Associate the Action element that you specified in step 2 with this deliveryChannel DC2.

4. **Define the Packaging.SimplePart:id Parameter.** Specify a value for Packaging.SimplePart:id parameter, say SP2, that attributes for the payload of the ebXML message.

5. **Define the Packaging:id Parameter.** Specify a value for the Packaging:id parameter, say PKG2.

6. **Associate the CompositeList parameters to the Message.** Specify the corresponding Packaging.CompositeList.Composite.Constituent[0]:idref parameter and Packaging.CompositeList.Composite.Constituent[1]:idref parameters to refer to the corresponding ebXML message, SP1 for the SOAP envelope and SP2 for the ebXML message with payload. Also, specify the value of the PartyInfo.CollaborationRole.ServiceBinding.Override:packageId parameter to PKG2.

C Trading Partner Agreement Parameters Version 2. 0

- "Overview" on page 119
- "TPA Parameters for ebXML Message Service Version 2.0 Specification" on page 119

Overview

The following section provides an overview of the Trading Partner Agreement (TPA) parameters for *ebXML Message Service Version 2.0 Specification* and the default parameter values. The parameters are listed in tables with the respective description for each parameter and its sub-parameter(s). For better visualization, some parent parameters are represented in separate tables by means of cross-references.

TPA Parameters for ebXML Message Service Version 2.0 Specification

The following table lists and describes the TPA Parameters for *ebXML Message Service Version 2.0 Specification* . This table helps you to create a TPA manually.

CollaborationProtocolAgreement

Parameters

schemaLocation	A string that provides a URI for the schema that describes the structure of the external information.
cpaid	A URI string that supplies a unique identifier for the document.
version	A string that is the version of the schema to which the CPA conforms.
Status	An IS document (IData object) that identifies the state of the process that creates the CPA. For a list of sub-parameters and their descriptions, see "Status" on page 120.
PartyInfo	An IS document (IData object) that records the organization whose capabilities are described in the CPA and includes all details about the Party. For a list of sub-parameters and their descriptions, see "PartyInfo" on page 120.
SimplePart	An IS document (IData object) that provides a repeatable list of the constituent parts identified by the MIME content-type value.

	For a list of sub-parameters and their descriptions, see "SimplePart" on page 126.
Packaging	An IS document (IData object) that provides specific information about how the Message Header and payload constituent(s) are packaged for transmittal over the transport, including information about document-level security packaging security features. For a list of sub-parameters and their descriptions, see "Packaging" on page 126.
Routing	An IS document (IData object) that contains routing information for messaging. For a list of sub-parameters and their descriptions, see "Routing" on page 128.
CustomParams	An IS document (IData object) that contains webMethods implementation-specific parameters.For a list of sub-parameters and their descriptions, see *"CustomParams" on page 129* .

Status

value

A string that records the current state of composition of the CPA. As per the CPP-CPA 2.0 specification, it can have three possible values: `proposed`, when CPA is still being negotiated by the Parties; `agreed`, when the contents of the CPA is accepted by both Parties; and `signed`, when CPA is signed by the Parties. ebXML Module supports `proposed` and `agreed`.

PartyInfo

partyName

A string that indicates the common name of the organization.

defaultMshChannelId

A string that identifies the default DeliveryChannel to be used for sending standalone Message Service Handler level messages.

defaultMshPackageId

A string that identifies the default Packaging to be used for sending standalone Message Service Handler level messages.

PartyId

An IS document that provides an identifier that shall be used to logically identify the Party.

Parameter	Description
body	A string that provides a unique identifier for the Party.
type	A string that provides a scope or namespace for the PartyId content.

CollaborationRole

An IS document that associates a Party with a specific role in the Business Collaboration. It identifies which role the Party is capable of playing in the Process Specification document referenced by the CPA.

Parameter	Description
Role	An IS document that identifies which role in the Process Specification the Party is capable of supporting. ■ **name** A string that gives a name to the Role. Its value is taken from a name attribute of one of the BinaryCollaborationRole elements. ■ **xlink:type** A string that has a fixed value of `simple`. This identifies the element as being an [XLINK] simple link. ■ **xlink:href** A string that identifies the location of the element or attribute within Process Specification.
ServiceBinding	An IS document that identifies a DeliveryChannel for all business Message traffic that is to be sent or received by the Party in the context of the Process Specification. ■ **Service** An IS document that determines the value of the Service element in the ebXML Message Header. ■ **body** A string that is the name of the service to be bound. ■ **type** A string indicating that the Parties sending and receiving the Message know how to interpret the value of the Service. When specified as webMethods, ebXML Module interprets the message processing service to be a webMethods IS service. ■ **CanSend** An IS document that identifies an action message that a Party is capable of sending. For a list of sub-parameters and their descriptions, see the following table.

CanSend

Parameter	Description
ThisPartyActionBinding	An IS document that defines one or more DeliveryChannel elements for Messages for a selected action and the Packaging for those Messages. ■ **id** A string that is the ID attribute for ThisPartyActionBinding. ■ **action** A string that identifies the business document exchange to be associated with the DeliveryChannel identified by the ChannelId children. ■ **packageId** An [XML] IDREF that identifies the Packaging element to be associated with the Message identified by the action. ■ **BusinessTransactionCharacteristics** An IS document that describes the security characteristics and other attributes of the DeliveryChannel. ■ `isNonRepudiationRequired` A Boolean string. If its value is set to `true`, the DeliveryChannel must specify that the Message is to be digitally signed. ■ `isNonRepudiationReceiptRequired` A Boolean string. If its value is set to `true`, a digest comparison of the received acknowledgment is performed against the original message. ■ `ChannelId` A string that identifies one or more DeliveryChannels that can be used for sending or receiving the respective action messages.

DeliveryChannel

An IS document that defines the DeliveryChannel's characteristics.

Parameter	Description
channelId	An [XML] ID that uniquely identifies the DeliveryChannel for reference.
docExchangeId	An [XML] IDREF that refers to the DocExchange of the DeliveryChannel.
MessagingCharacteristics	An IS document that describes the attributes associated with messages delivered over a DeliveryChannel. ■ **syncReplyMode** A string that defines the content of a synchronous reply. It can have one of four values: `none`,

Parameter	Description
	`mshSignalsOnly`, `responseOnly`, `signalsAndResponse`, and`signalsOnly`. When the value is set to `none`, the reply is asynchronous. At present, `mshSignalsOnly`, `responseOnly`, `signalsAndResponse and signalsOnly` are all interpreted as synchronous, and ebXML Module does not handle each of these values specifically. ■ **ackRequested** A string that determines whether ackRequested in the SOAP Header is to be present. ■ **ackSignatureRequested** A string that determines how the signed attribute in the ackRequested element in the SOAP Header is to be set. ■ **duplicateElimination** A string that determines whether the duplicateElimination in Message Header under SOAP Header is to be present. ■ **actor** A URI string that specifies the value for the actor attribute in ackRequested.

Transport

An IS document that defines the communication protocols for specific ebXML message actions.

Parameter	Description
transportId	A string that is the ID for each Transport element.
TransportSender	An IS Document that identifies the transport protocol that the sender uses to send an ebXML message to its intended receiver. The TransportSender.TransportProtocol:body parameter can be set to one of the following values: `Preferred`, `Primary HTTP`, `Secondary HTTP`, `Primary HTTPS`, `Secondary HTTPS`, `Primary Email`, or `Secondary Email`. Each of these values correspond to a delivery method defined in the profile of the receiver.
TransportReceiver	An IS document that specifies the transport protocol used for receiving the messages. ■ **Endpoint** An IS document that specifies a logical address where messages can be received. ■ **uri** A URI string that identifies the address of a resource. ■ **type** A string that identifies thepurpose of the Endpoint.

DocExchange

An IS document that provides information that the Parties must agree on regarding exchange of documents between them.

Parameter	Description
DocExchangeId	A string that provides a unique identifier that can be referenced from elsewhere in the CPA.
ebXMLSenderBinding	An IS document that describes properties related to sending messages with the ebXML Message Service.

- **version** A string that is the version of the ebXML Message Service specification.
- **ReliableMessaging** An IS document that contains the properties of reliable ebXML Message exchange.
  - **Retries** A string that defines the permitted number of retries of sending a reliably delivered message following a timeout.
  - **RetryInterval** A string that defines the interval between retries.

    Note:ebXML Module calculates the value of the TimeToLive element based on the values you enter for the Retries and the RetryInterval parameters for reliable messaging.
  - **MessageOrderSemantics** A string that can have one of two values: `Guaranteed` for preserving the order in which the messages were sent, and `NotGuaranteed` for no need to preserve the order.
- **SenderNonRepudiation** An IS document that conveys the message sender's requirements for non-repudiation.
  - **NonRepudiationProtocol** An IS document defining the technology that will be used to digitally sign a Message.
    - **body** A string that is the name of NonRepudiationProtocol.
  - **HashFunction** A string that identifies the algorithm used to compute the digest of the Message being signed.
  - **SignatureAlgorithm** An IS document that identifies the algorithm used to compute the value of the digital signature.

Parameter	Description
	■ **body** A string that is the ID attribute for SignatureAlgorithm.
ebXMLReceiverBinding	An IS document that describes properties related to receiving messages with the ebXML Message Service. ■ **PersistDuration** A string that provides the length of time (in seconds). In case of non reliable messaging, this parameter is used for computing the value of the TimeToLive element. ■ **ReceiverDigitalEnvelope** An IS document that conveys the receiver's requirements for message encryption using the digital-envelope method. ■ **DigitalEnvelopeProtocol** An IS document that identifies the message encryption protocol to be used. ■ **body** A string that is the name of DigitalEnvelopeProtocol. ■ **EncryptionAlgorithm** An IS document that identifies the message encryption algorithm to be used. ■ **body** A string that is the name of the EncryptionAlgorithm. ■ **minimumStrength** A string that defines the effective strength the encryption algorithm must provide in terms of effective or random bits.

OverrideMshActionBinding

An IS document that overrides the DeliveryChannel specified by PartyInfo's defaultMshChannelId for standalone MSH level messages such as Acknowledgment, Error, StatusRequest, StatusResponse, Ping, and Pong.

Parameter	Description
action	A string that identifies the Message Service Handler level action whose delivery is not to use the default DeliveryChannel for Message Service Handler actions.
channelId	A string that specifies the DeliveryChannel to be used instead.

SimplePart

id

A string that is the content ID for the MIME part.

mimetype

A string that is the content-type for the MIME part.

mimeparameters

A string that provides the values of any significant MIME parameter(s) needed to understand the processing demands of the content-type.

NamespaceSupported

An IS document that identifies the schema for the MIME part.

location

A URI for retrieval of the schema associated with the namespace.

version

A version value for the namespace.

Packaging

id

A string that is referred to by packageId under ThisPartyActionBinding.

CompositeList

An IS document that specifies the way in which the simple parts are combined in MIME multiparts or encapsulated in security-related MIME content-types.

Parameter	Description
Encapsulation	An IS document that indicates the use of MIME security mechanisms. ■ **id** A string that provides a way to refer to the Encapsulation if it needs to be mentioned as a subelement of some later element in the sequence.

Parameter	Description

	■ **mimetype** A string that is the content-type for the MIME part.
	■ **mimeparameters** A string that provides the values of any significant MIME parameter(s) needed to understand the processing demands of the content-type.
	■ **Constituent** An IS document that indicates the contents and order of the packages in Encapsulation.
	  ■ **idref** A string that is the ID of a previous Composite, Encapsulation or SimplePart.
	  ■ **SignatureTransforms** An IS document defining the transforms that must be applied to the source data before a digest. For a list of sub-parameters and their descriptions, see the following table.

Composite	An IS document that contains information about MIME multiparts.
	■ **id** A string identifier for the Composite element.
	■ **mimetype** A string that is the content-type for the MIME part.
	■ **Constituent** An IS document that indicates the contents and order of the packages in Composite.
	  ■ **idref** A string that is the ID of a previous Encapsulation or SimplePart.
	  ■ **excludedFromSignature** A string indicating that this Constituent is to be excluded as part of the ebXML message signature.

SignatureTransforms

Parameter	Description
Transform	An IS document that is an ordered list with a sequence of transforms. ■ **Algorithm** A string that specifies the algorithm to be applied. ■ **XSLT** A string that contains an embedded XSLT stylesheet. ■ **XPath** A string that contains an XPath expression. It filters what part of a document is signed.

Routing

previousMSH

The previous MSH from which the message is sent.

Parameter	Description
ids	A string that provides a unique identifier for the previous Party MSH.
idTypes	A string that is a type attribute for IDs.

nextMSH

The next MSH to which the message is sent.

Parameter	Description
ids	A string that provides a unique identifier for the next Party MSH.
idTypes	A string that is a type attribute for IDs.

senderMSH

An IS document that identifies the original sender MSH.

Parameter	Description
ids	A string that provides a unique identifier for the sender Party MSH.
idTypes	A string that is a type attribute for IDs.

receiverMSH

An IS document that identifies the original receiver MSH.

Parameter	Description
ids	A string that provides a unique identifier for the receiver Party MSH.
idTypes	A string that is a type attribute for IDs.

CustomParams

AutoAckSend

Enables or disables the sending of automatic acknowledgments.

- Set to `false` to disable the sending of automatic acknowledgments when messages are processed successfully. You can explicitly send acknowledgments using the wm.ip.ebxml.MSH:sendAck service.
- Set to `true` to enable ebXML Module to automatically send acknowledgments to the message sender if the messages were processed successfully. This is the default setting.

Automatic acknowledgments can also be controlled with ebXML Module configuration parameter wm.ebxml.autoAckSend. For more information, see "Configuring webMethods ebXML Module" on page 23.

payload\description

Specifies descriptive text for the payload to be included in the MessageHeader element of a SOAP message. The ID field in this parameter refers to the payload ID in the SimplePart section of the TPAs.

timestampFormat

A string that specifies the timestamp format in the ebXML message.

- Set this property to `default1` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:SSS'Z' (for example, 2008-05-20T13:20:10.663Z). This is the default setting.
- Set this property to `default2` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:mm (for example, 2008-05-220T13:20:10).
- Set this property to `default3` to set the timestamp format to yyyy-MM-dd'T'HH:mm:ss-HH:mm, where yyyy-MM-dd'T'HH:mm:ss is the local time, and -HH:mm is the time zone offset. (For example, "2008-05-20T13:20:10-05:00" represents "20 May 2008 13:20:10 GMT-05:00".)
- Set this property to any other valid Java timestamp format.

> **Note:**
> For more information about the timestamp formats above, see the *XML Schema Part 2: Datatypes* document available at http://www.w3.org/TR/xmlschema-2

timezone

A String that is the valid Java time zone. This value overwrites the global setting provided by the wm.ebxml.timestamp property in the WmebXML/config/config.cnf file.

Example: `GMT-05:00` for EST time.

Default Parameter Values

In Trading Networks, when you run the initialization service (wm.ip.ebxml.cpa:initTPA2), the TPA IS document type parameters listed in the following table are populated with the default values indicated.

Parameter Name	Default Value
version	`2.0`
schemaLocation	`http://www.oasis-open.org/committees/ebxml-cppa/schema/cpp-cpa-2_0.xsd`
Status	`Agreed`
PartyInfo:defaultMshChannelId	`DC1`
PartyInfo:defaultMshPackageId	`PKG1`
PartyInfo.DeliveryChannel:channelId	`DC1`
PartyInfo.DeliveryChannel:docExchangeId	`DX1`
PartyInfo.DeliveryChannel.MessagingCharacteristics:syncReplyMode	`none`
PartyInfo.DeliveryChannel.MessagingCharacteristics:ackRequested	`never`
PartyInfo.DeliveryChannel.MessagingCharacteristics:ackSignatureRequested	`never`
PartyInfo.DeliveryChannel.MessagingCharacteristics:duplicateElimination	`never`
PartyInfo.DeliveryChannel.MessagingCharacteristics:actor	`urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH`
PartyInfo.DocExchange.ebXMLSenderBinding:version	`2.0`
PartyInfo.DocExchange.ebXMLSenderBinding.ReliableMessaging:MessageOrderSemantics	`NotGuaranteed`
SimplePart:id	`SP1`
SimplePart:mimetype	`text/xml`
Packaging:id	`PKG1`
Packaging.CompositeList.Composite:id	`C1`
Packaging.CompositeList.Composite.Constituent:idref	`SP1`

For information about creating the TPA structure manually for message exchange with payloads, see the "Default Parameter Values" on page 130. The parameters listed may be different for the TPA Structure 2.0.