webMethods Module for AS4 Installation and User's Guide

Version 10.1

May 2019

WEBMETHODS

This document applies to webMethods Module for EDI 10.1 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-EDI-IUG-101-20230920

Table of Contents

About this Guide

This guide describes how to install, configure, and use webMethods Module for AS4. It contains information for administrators and application developers who want to exchange AS4-specific messages with other trading partners using webMethods Module for AS4.

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

- webMethods Integration Server, Integration Server Administrator, and understand the concepts and procedures in the *webMethods Integration Server Administrator's Guide* for your release.
- webMethods Trading Networks, and understand the concepts and procedures described in the various Trading Networks guides.
- Software AG Designer, and understand the concepts and procedures described in the *Software AG Designer Online Help*.
- My webMethods Server and its interface My webMethods, and understand the concepts and procedures described in *Administering My webMethods Server*.
- *OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features Specification* and *OASIS ebXML Messaging Services Version 3.0: Part 2, Advanced Features*.
- AS4 Profile of the *OASIS ebMS 3.0 Version 1.0 Specification*, *ENTSOG AS4 Profile Version 3.5*, and *ENTSOG AS4 Agreements and Agreement Updates Version 1*.

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.

Convention	Description
[ ]	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

Product Documentation

You can find the product documentation on our documentation website at https://documentation.softwareag.com.

In addition, you can also access the cloud product documentation via https://www.softwareag.cloud. Navigate to the desired product and then, depending on your solution, go to "Developer Center", "User Center" or "Documentation".

Product Training

You can find helpful product training material on our Learning Portal at https://knowledge.softwareag.com.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at https://techcommunity.softwareag.com. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://hub.docker.com/u/softwareag and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at https://empower.softwareag.com. Many services on this portal require that you have an account. If you do not yet have one, you can request it at https://empower.softwareag.com/register. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.

- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

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

What is webMethods Module for AS4?

The ebXML Messaging Services (ebMS) specification was developed to provide organizations a standard to exchange business messages over the Internet. AS4 is a simplified subset of ebMS 3.0 specification that provides guidelines for payload independent exchange of documents using web services.

webMethods Module for AS4 is an implementation of the AS4 ebHandler conformance profile specified in the AS4 Profile. Using Module for AS4, you can exchange messages, independent of payload type, across your trading network.

Features

Module for AS4 includes the following features:

- **Message Exchange Patterns.** A *Message Exchange Pattern* (MEP) defines how messages are exchanged between two trading partners. Module for AS4 supports the following ebMS MEPs for both the sender and the receiver:
  - One-way/Push: Used to send a user message to a trading partner.
  - One-way/Pull: Used to send a pull signal to a trading partner to receive a user message.
  - Two-Way/Sync: Used to send a user message to a trading partner and receive a reply from the same partner.
  - Two-Way/Push-Pull: Used to push a user message to a trading partner followed by a pull signal to receive a reply user message.
  - Two-Way/Push-Push: Used to send a user message to a trading partner and receive a reply user message from the same partner asynchronously.

  For more information about message exchange patterns, see "Configuring Trading Partner Agreements" on page 34.
- **Message Packaging.** The module supports creation of signal and user messages, attachments, and message properties.
- **Message Partition Channels (MPC).** The module supports multiple MPCs for selective pulling by a partner Messaging Service Handler (MSH).
- **Message Persistence.** The module saves all incoming and outgoing messages in the Trading Networks database.
- **Reliable Messaging.** The module supports:
  - **Reception Awareness.** When an initiating MSH does not get a receipt confirming the user message has been received, the MSH can notify the producer.
  - **Message Retry.** When an initiating MSH does not get a receipt confirming the user message has been received, the MSH can resend the user message.
  - **Duplicate Detection.** A responding MSH can detect and eliminate duplicate user messages.

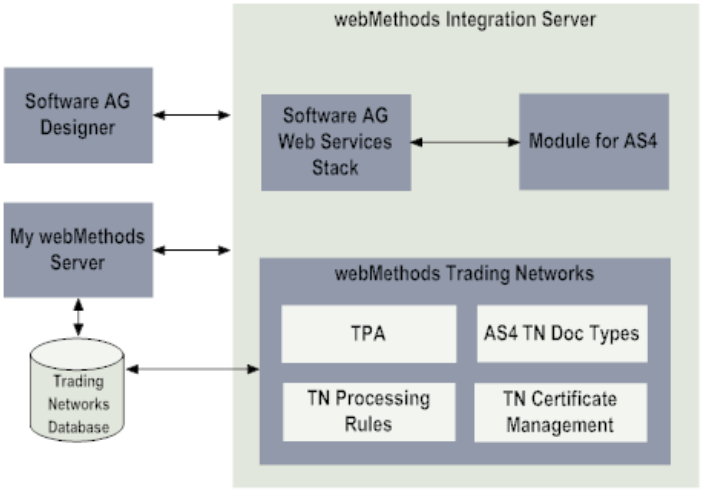
- **Security**. Module for AS4 uses the *Web Services Security 1.1 (WSS 1.1)* security specification and provides the following security features:
  - **Signs and encrypts messages.** The module uses the *Web Services Security X.509 Certificate Token Profile* to sign and encrypt messages. When signing and encrypting messages, you can choose to sign and encrypt either the entire message or parts of the message. When both signature and encryption are required, the message is signed prior to being encrypted.
  - **Authenticates Security UsernameTokens and supports password hashing.** The module uses the *Web Services Security UsernameToken Profile* to authenticate Security UsernameTokens.
  - **Authorizes a message at the P-Mode level.** The module supports authorization to access resources such as Message Partition Channels (MPC), which pull signals access to pull messages.
  - **Supports Secure Sockets Layer (SSL).** The module supports transport level security for transmitting user messages.
- **Customizable Processing Mode (P-Mode) Parameters.** P-Mode parameters are required for interoperability between two trading partners. The parameters define how a message should be processed. P-Modes in Module for AS4 are represented as trading partner agreements (TPA).

  For a list of supported P-Mode parameters, see "Supported P-Mode Parameters" on page 117.
- **Agreement Update.** The module supports agreement update standard that conforms to the ebCore Agreement Update specification. This feature enables in governing updates to certificates.
- **Payload Compression.** The module is capable of providing configurable compression and decompression of application payloads.
- **Splitting and Joining ebMS Messages.** When a large SOAP-rooted, MIME Multipart/Related message is to be sent, you can configure Module for AS4 to split the message into smaller SOAP fragments. Once all the fragments reach the receiving MSH, they are reassembled and stored in the Trading Networks database.
- **Multi-hop Messaging.** The module is capable of transmitting messages across intermediaries, which route and forward messages while maintaining reliability and security. Multi-hop messaging allows the sending MSH to disregard lower-level transport parameters because it is not concerned with the final message destination.
- **Error Handling.** The module reports errors encountered during message processing, as specified in the AS4 ebHandler conformance profile.
- **Large Payload Handling.** During message processing, the module can optimize processing of messages by storing large payloads on the hard disk drive rather than keeping them in memory.
- **Error message bundling.** The module supports the bundling and processing of multiple error messages into a single message.

- **Payload extraction.** The module supports the extraction of the payload from a user message as a separate transaction.

Architecture and Components

The following diagram shows the module's architecture, components, and component relationships. The components are described in the table.



Component	Description
webMethods Integration Server	webMethods Integration Server is the underlying foundation of the webMethods architecture. It processes requests from, and relays responses to a back-end system. Integration Server contains the IS document types and services that you use to create the module's services. The module is compatible with Integration Server 9.0 and above.
Software AG Web Services Stack	Software AG Web Services Stack is a toolkit that handles the complex process of sending and receiving web services requests.
webMethods Module for AS4	webMethods Module for AS4 contains the core WmAS4 package, the WmAS4AU package, and the services for implementing the AS4 functionality.
webMethods Trading Networks	webMethods Trading Networks enables your enterprise to link with other companies (buyers, suppliers, strategic partners) and marketplaces to form a business-to-business trading network. Module for AS4 uses Trading Networks and TN document types to: ■ Recognize user messages and signals being processed

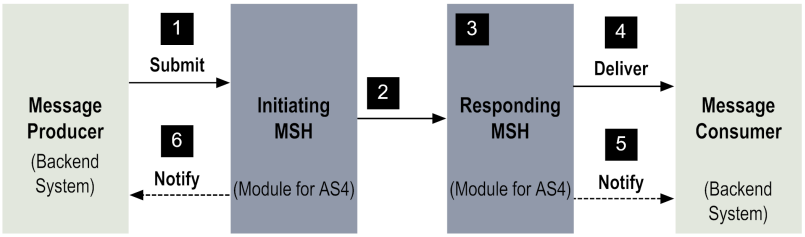
Component	Description
	■ Manage security keys and certificates for your enterprise and partners ■ Create BizDocEnvelopes ■ Save BizDocEnvelopes to the Trading Networks database Customizable Trading Networks processing rules allow for custom processing. The P-Mode parameters defined in the ebMS 3.0 specification are modeled as Trading Networks TPAs. These TPAs can be customized using My webMethods. The TN document types are created automatically when starting the WmAS4 package, and you can view them using My webMethods. For more information about P-Modes, see *OASIS ebXML Messaging Services 3.0 Version 1.0, Part 1, Core Features*. For more information about TPAs and processing rules, see *webMethods Trading Networks Administrator's Guide*.
Trading Networks database	Trading Networks database stores all information about trading networks, such as: trading partner profiles, TN document types, and TPA information. Information and content of the processed documents is stored in the TN database.
My webMethods Server and My webMethods	My webMethods Server is a web-based monitoring and administration user interface for managing webMethods components. You can use My webMethods Server (and its user interface, My webMethods) with Module for AS4 to define and manage Trading Networks partner profiles, TPAs, custom attributes, processing rules, and TN document types. My webMethods can also be used to monitor the message transaction details. At design time, you can 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.
Software AG Designer	Designer is a development environment used at design time to edit the module's services and create customized solutions. Designer also provides tools for testing and debugging the solutions you create.

Messaging Model

The messaging model has the following components:

- **Messaging Service Handler (MSH).** MSH generates or processes AS4-compliant messages. An MSH can be a sender (initiating MSH) or a receiver (responding MSH).
- **Message Producer.** The message producer interacts with an initiating MSH to send a user message.
- **Message Consumer.** The message consumer interacts with a responding MSH to receive a user message.

The following diagram shows the messaging model. The steps are described in the table.



Step	Description
1	The message producer submits data to the initiating MSH.
2	The initiating MSH packages the submitted data as a user message and sends the message to the responding MSH.
3	The responding MSH receives and processes the user message. The responding MSH sends either a receipt or an error to the initiating MSH to indicate if the message was processed successfully.
4	The responding MSH delivers the data packaged as the user message to the message consumer.
5	The responding MSH may optionally notify the message consumer about the status of the received message.
6	The initiating MSH may optionally notify the message producer about the status of the message that was sent.

Message Exchange Patterns

Module for AS4 supports the following one-way and two-way MEPs:

One-Way:

- One-Way/Push
- One-Way/Pull
  - Selective-Pull

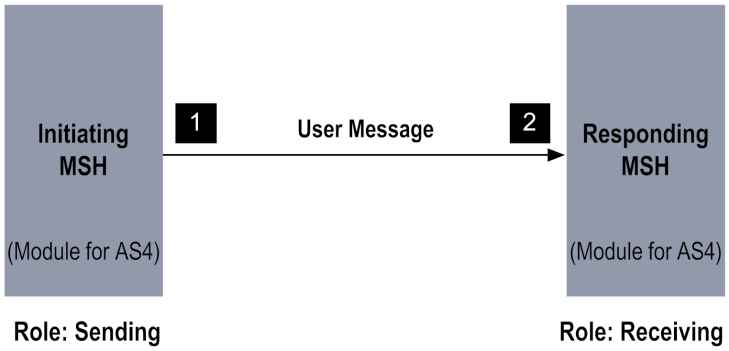
Two-Way:

- Two-Way Sync
- Two-Way/Push-Pull
- Two-Way/Push-Push

Module for AS4 can be used in the role of an initiating MSH or a responding MSH.

One-Way/Push MEP

This MEP is used to send (push) a user message to a trading partner. The user message can be carried over a one-way or a two-way underlying protocol.



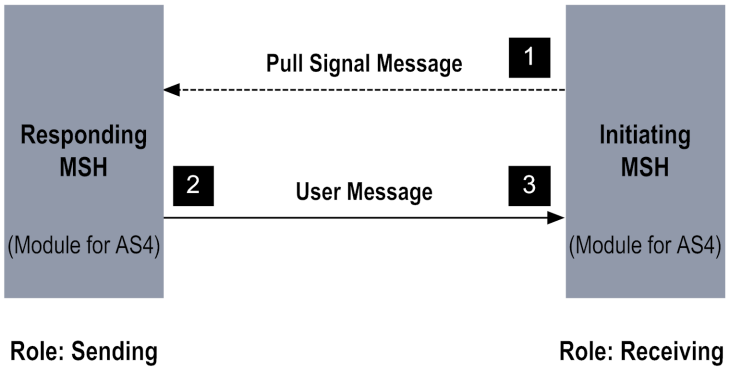
Step	Description
1	The initiating MSH sends the data packaged as a user message to the responding MSH.
2	The responding MSH receives and processes the user message.

When using the One-Way/Push MEP, use the wm.ip.estd.as4.msh:submit service to package the submitted payloads as a user message and send it to the responding MSH. The submit service uses the values of the *toPartyId*, *toPartyIdType*, *fromPartyId*, *fromPartyIdType*, and *pmodeID*. If *pmodeID* is empty, then a tuple of *toPartyId*, *toRole*, *fromPartyId*, *fromRole*, *Service*, *Action*, and *AgreementRef* input parameters to fetch the appropriate TPA. The values in the TPA are then used to generate and send the user message.

For information about configuring the module to use the One-Way/Push MEP, see "Configuring Trading Partner Agreements" on page 34. For more information about the submit service, see "Built-In Services" on page 91.

One-Way/Pull MEP

This MEP is used to send a pull signal to a trading partner to receive a user message. The Initiating MSH sends a pull signal message to the Responding MSH, as illustrated in the diagram.



Step	Description
1	The initiating MSH sends a pull signal message to the responding MSH.
2	The responding MSH sends the message data packaged as a user message to the initiating MSH.
3	The initiating MSH receives the user message for processing.

When using One-Way/Pull, Module for AS4 can play the role of initiating MSH or responding MSH. The role played by the MSH will depend on whether you want to send or receive a user message.

Selective-Pull

Selective pull feature allows you to pull a subset of messages posted on an MPC. The initiating MSH sends a pull signal to the responding MSH along with a pull criteria using requestSM. The responding MSH uses the requestSM parameters to match the messages present in the MPC and sends back the first matching message as a response using requestUM.

Selective pull can be initiated by a business application by providing simple selection item or complex selection item parameters in the wm.ip.estd.as4.msh:sendPull service.

Simple selection, complex selection or both can be used to pull messages from an MPC. Following are the two types of selection:

- Simple selection: Use this option to select messages using any of the following parameters: RefToMessageId, ConversationId, AgreementRef, Service, and Action.
- Complex selection: Use this option to select messages using any of the following parameters: From, To, and MessageProperties.

For information about using selective pull service, see "Built-In Services" on page 91.

Module for AS4 as Initiating MSH

When Module for AS4 plays the role of initiating MSH, the pull transfer is initiated by the wm.ip.estd.as4.msh:sendPull service. The module uses the *mpc* and *pmodeId* input parameters that are

passed to this service to fetch the appropriate TPA and identify the messaging parameters for the transfer.

The parameters specified in the requestSM leg of the TPA are used to generate the pull signal and identify the address of the responding MSH. When the user message is received as a response to the pull signal, the module uses the requestUM leg of the TPA to process the user message.

For more information about configuring the module to use the One-Way/Pull MEP, see "Configuring Trading Partner Agreements" on page 34. For more information about the sendPull service, see "Built-In Services" on page 91.

Module for AS4 as Responding MSH

When using the One-Way/Pull MEP, use the com.ip.estd.as4.msh.submit service to package the submitted payload(s) as a user message and submit it to an MPC. The submit service uses the values of the *toPartyId*, *toPartyIdType*, *fromPartyId*, *fromPartyIdType*, and *pmodeID*. If *pmodeID* is empty, then a tuple of *toPartyId*, *toRole*, *fromPartyId*, *fromRole*, *Service*, *Action*, and *AgreementRef* input parameters to fetch the appropriate TPA. The values in the TPA are then used to identify the messaging parameters. Instead of transferring the message to the initiating MSH, the module stores the message in the MPC specified in the *legs/businessInfo/mpc* TPA parameter of the requestUM leg.

When a pull signal is received, the module determines which TPA to use to process the pull signal by matching the MPC named in the pull signal with the MPC named in the requestSM leg of each TPA that is configured in Trading Networks. The matching MPC is used to process the pull signal. The module retrieves the user message from the MPC specified in the pull signal and sends it to the initiating MSH. If no MPC is specified in the pull signal, the module uses the default MPC to pull the user message.

Module for AS4 uses the Trading Networks database as the MPC queue. When a user message is queued, the message is stored in Trading Networks with a UserStatus of QUEUED, and the corresponding bizDocId is cached. When a pull signal is received, the module first retrieves the bizDocId from the cache First In, First Out, and then retrieves the user message that corresponds to this bizDocId from Trading Networks. The retrieved message is then sent to the initiating MSH.

For more information about configuring the module to use the One-Way/Pull MEP, see "Configuring Trading Partner Agreements" on page 34. For more information about the submit service, see "Built-In Services" on page 91.

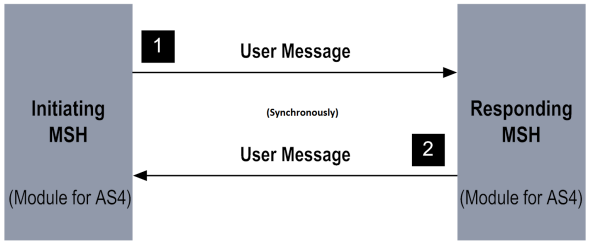
Important:
When purging the transaction analysis in Trading Networks, do not delete user messages in the QUEUED state.

Whenever you purge the transaction analysis, use the wm.ip.estd.as4:clearMPC service to manually clear all the BizDocIds that are queued in the cache.

Two-Way/Sync MEP

This MEP is used to synchronously exchange messages between two trading partners. The initiating MSH sends a user message as a request to the responding MSH. The responding MSH replies with a user message to the initiating MSH.

The reply must refer to the request.



Steps	Description
1	The initiating MSH sends a user message as a request to the responding MSH.
2	The responding MSH sends a user message as a reply.

When using Two-Way/Sync, Module for AS4 can play the role of initiating or responding MSH. The role played by the MSH will depend on whether you want to send a user message as a request or reply.

Module for AS4 as Initiating MSH

When Module for AS4 plays the role of initiating MSH, the Two-Way/Sync transfer is initiated by the wm.ip.estd.as4.msh:submit service.

The parameters specified in the requestUM leg of the TPA are used to generate the user message and identify the address of the responding MSH. After the module receives a reply, it uses the replyUM leg of the TPA to process the user message.

If the module does not receive a response within the time-period defined in `as4.twowaysync.requestTimeOut` property in the AS4 configuration file, an error message is generated by the initiating MSH. For more information, see "Configuring AS4 Configuration Properties" on page 52.

For more information about configuring the module to use the Two-Way/Sync MEP, see "Configuring Trading Partner Agreements" on page 34. For more information about built-in service, see "Built-In Services" on page 91.

Module for AS4 as Responding MSH

When Module for AS4 plays the role of responding MSH, use the wm.ip.estd.as4.msh.userMessageSubmit service to submit a reply user message for the corresponding request user message. The refToMessageId of the reply user message should be set to the request user message's MessageId

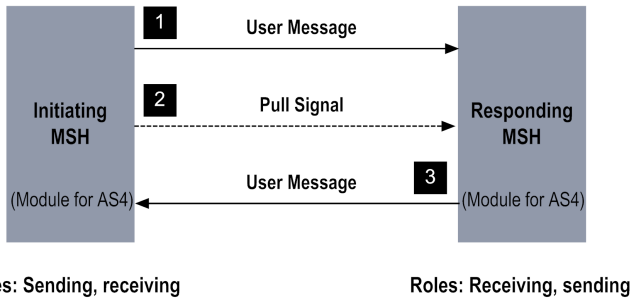
value. The module uses the requestUM leg of the TPA to process the request user message and sends the reply to the initiating MSH.

Module for AS4 uses the Trading Networks database as the MPC queue. When a user message is queued, the message is stored in Trading Networks with a UserStatus of QUEUED, and the corresponding bizDocId is cached.

Two-Way/Push-Pull

This MEP is used to push a request user message to a trading partner followed by a pull signal to receive a reply user message as shown in the diagram. Both the user message and the pull signal are sent by the initiating MSH.

The pulled user message refers to the user message that was pushed.



Steps	Description
1	The initiating MSH sends a user message to the responding MSH.
2	The initiating MSH sends a pull signal to the responding MSH.
3	A user message is submitted at the responding MSH as a response to the request user message.

For more information about configuring the module to use the Two-Way/Push-Pull MEP, see "Configuring Trading Partner Agreements" on page 34. For more information about built-in service, see "Built-In Services" on page 91.

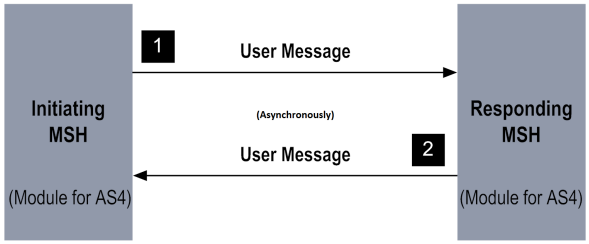
Important:
When purging the transaction analysis in Trading Networks, do not delete user messages in the QUEUED state.

Whenever you purge the transaction analysis, use the wm.ip.estd.as4:clearMPC service to manually clear all the BizDocIds that are queued in the cache.

Two-Way/Push-Push

This MEP is used to asynchronously exchange messages between two trading partners. The initiating MSH sends a user message as a request to the responding MSH. The responding MSH replies with a user message to the initiating MSH.

The reply must refer to the request.



Steps	Description
1	The initiating MSH sends a user message as a request to the responding MSH.
2	The responding MSH sends a user message as a reply.

When using Two-Way/Push-Push, Module for AS4 can play the role of an initiating or a responding MSH. The role played by the MSH will depend on whether you want to send a user message as a request or reply.

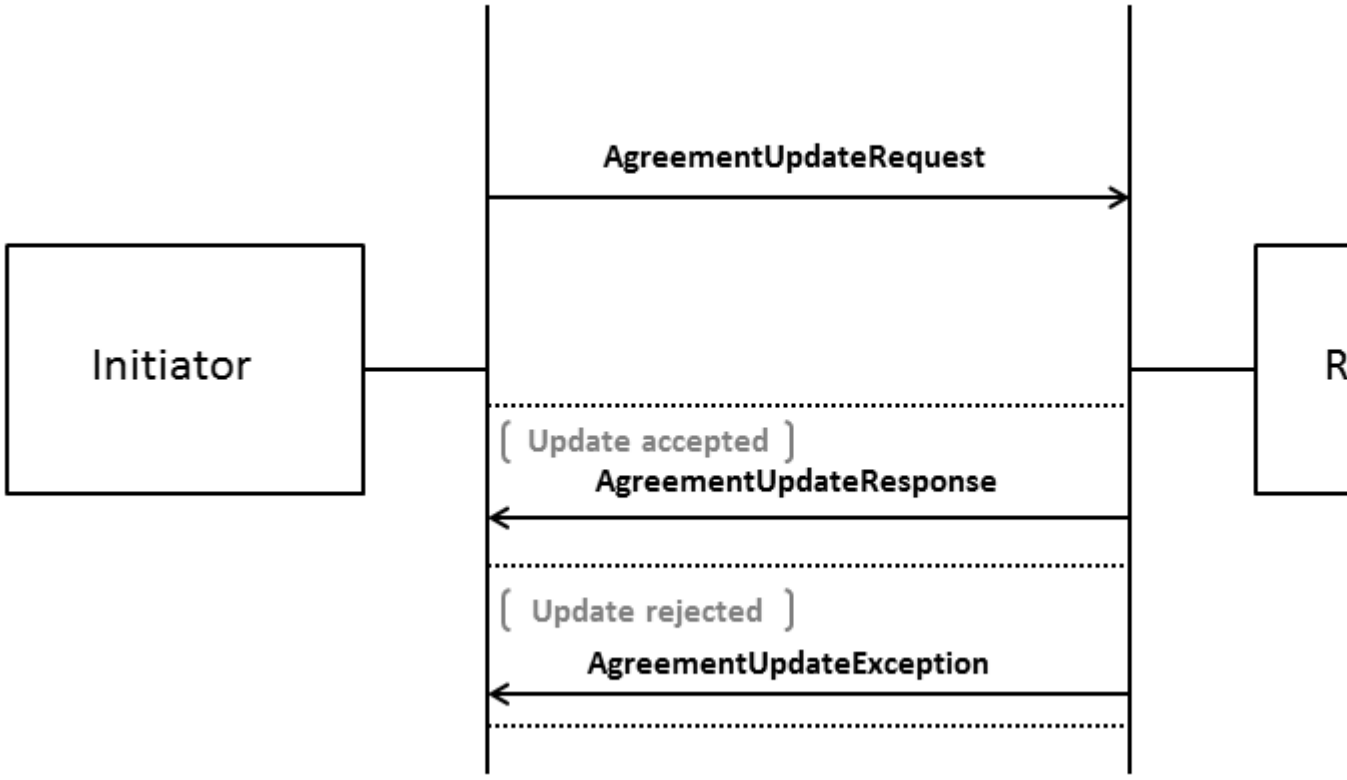
For more information about configuring the module to use the Two-Way/Push-Push MEP, see "Configuring Trading Partner Agreements" on page 34. For more information about built-in service, see "Built-In Services" on page 91.

Agreement Update

The Agreement Update (AU) standard conforms to the ebCore Agreement Update specification. AU enables in governing updates to certificates. In addition, you can also create new agreements based on existing agreements.

When an initiator requests a responder for an update to an existing agreement, the initiator sends an AgreementUpdateRequest document to the responder. An AgreementUpdateRequest includes one or more UpdateRequest elements. An UpdateRequest is an abstract element that has the abstract type UpdateRequestType. For certificate updates, an `AgreementUpdateRequest` must include `CertificateUpdateRequest` to request an update for a specific certificate used in a particular agreement.

The exchange of agreement updates in AS4 user messages between trading partners includes requests, responses, and exceptions. An agreement update exchange is mostly an asynchronous process that may take hours or days. A typical flow of an agreement update exchange between trading partners is as follows:



For more information about AU services, see "Built-In Services" on page 91. For more information about Agreement Update, see *OASIS ebCore Agreement Update Specification Version 1.0*.

The initiating MSH generates the AgreementUpdateRequest message using the wm.estd.as4.au.message:generateUpdateRequest service in the format specified by AU specification document. The initiator submits the generated Agreement Update Request message using the wm.ip.estd.as4.msh:submit service as a payload in the user message to the responder. The submit service recognizes the user message as an Agreement Update message based on the service and action values as specified in the Agreement Update specification, service input, and persists the user message and Agreement Update Message into Trading Networks as two separate transactions and sends them to the responder.

The responding MSH receives the Agreement Update Request message, recognizes it as an AgreementUpdateRequest message based on action and service specified in the user message header and persists it as user message and AgreementUpdateRequest into Trading Networks as two separate transactions and invokes the processing rule that it is associated with. The wm.estd.as4.au.handlers:validate service is used to validate the AgreementUpdateRequest message and generates the AU errors if the message is invalid. If the validation results in errors, the responder uses the errors generated by the validate service, generates the AgreementUpdateException message using the message generation wm.estd.as4.au.message:generateUpdateException service, and sends it as a payload in the user message using the wm.ip.estd.as4.msh:submit service to the initiator. On successful validation of the AgreementUpdateRequest service, the responding MSH uses the wm.estd.as4.au.handlers:process service to process the request message, updates the certificates, and generates new agreements with new certificate ID. Then, sends the AgreementUpdateResponse

message using the message generation wm.estd.as4.au.message.generateUpdateResponse service and sends it as a payload in the user message to the initiator using the wm.ip.estd.as4.msh:submit service.

The wm.estd.as4.au.handlers:validate service validates the inbound or outbound agreement update messages as described in the Agreement Update specification. If the message is invalid or is not according to the Agreement Update specification, then Agreement Update errors are generated.

The wm.estd.as4.au.handlers:process service updates the new certificate identified based on CurrentCertificateIdentifier present in the CertificateUpdateRequest in the initiator's Trading Networks partner profile at the responding MSH side. And, generates new TPAs for all the existing agreements identified based on CurrentAgreementIdentifier by copying the configuration from the existing agreements and configuring the new certificate ID for signing and encrypting. The wm.estd.as4.au.handlers:process service creates a new TPA by replacing the new version either by extracting the version number from update request if UpdateAgreementIdentifier is in the format `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/1.1` or the version numbers are incremented by appending `_v_` string to the current TPA ID.

Example1: If the UpdateAgreementIdentifier is in `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/1.1` format, then the current TPA ID is `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/Request_v_1`, UpdateAgreementIdentifier is `https://entsog.eu/communication/agree_ment_s/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/1.1`, and new agreement ID is `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/Request_v_1.1`.

Example 2: If the UpdateAgreementIdentifier is not in `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/1.1`, then current TPA ID is `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/Request_v_1`, UpdateAgreementIdentifier is `1.1`, and new agreement ID is `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/Request_v_2`.

Example 3: If the current TPA ID does not contain any version format, then current TPA ID is `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/Request`, UpdateAgreementIdentifier is `1.1`, and new agreement ID is `https://entsog.eu/communication/agreements/21X-EU-A-X0A0Y-Z/21X-EU-B-P0Q0R-S/Request_v_1` (starts with 1 and is incremented for the next request).

Module for AS4 supports the following ebCore AU documents:

- AgreementUpdateRequest
- AgreementUpdateResponse
- AgreementUpdateException
- AgreementTerminationException
- AgreementTerminationRequest
- AgreementTerminationResponse

2 Installing webMethods Module for AS4

Overview

This chapter explains how to install, upgrade, and uninstall webMethods Module for AS4.

Requirements

For a list of requirements, supported operating systems, and webMethods products, see *webMethods eStandards Modules System Requirements*, available in the webMethods area of the Software AG Documentation website.

Installing webMethods Module for AS4

To install webMethods Module for AS4:

1. If you are installing Module for AS4 on an already installed webMethods Integration Server, shut down the Integration Server. Ensure that the version of Integration Server is 10.1 or above.

2. Download Software AG Installer from the Empower Product Support website.

3. Start the Installer wizard.

4. Choose the webMethods release that includes the Integration Server on which you want to install the module.

5. Specify the installation directory as follows:

   - If you are installing Module for AS4 on an existing Integration Server, specify the installation directory that contains the host Integration Server.
   - If you are installing both the host Integration Server and Module for AS4 for the first time, specify the directory to use. The default directory name is SoftwareAG.

6. In the product selection list, go to **eStandards > webMethods Module 10.1 for AS4**. For Agreement Update (AU), go to **eStandards > webMethods Module 10.1 for AS4 > Agreement Update**. Select any other required products as indicated in *webMethods eStandards Modules System Requirements*.

7. Select **Install the packages on the default instance as well** to install the package in the default instance.

   WmAS4 package is installed by default in the package repository located in the *Integration Server_directory*\packages directory.

   When installing the module on Integration Server, you can choose to install the package in the default instance. In this case, the package will be installed both in the package repository and the default instance packages directory located in *Integration Server_directory*\instances\default\packages.

You can move the package to an IS Instance of your choice by using the is_instance Script. For more information about using is_instance Script, see *webMethods Integration Server Administrator's Guide*.

Note:

To ensure the seamless operation of Module for AS4 with Integration Server, when migrating your assets from one version of Integration Server to another, or when you create a new instance using ..\IntegrationServer\instances\is_instance.bat, you must copy the following files:

- *AS4Service.aar* to ..*IntegrationServer**instances**default**config**wss**services*\
- *AS4Module.mar* to ..*IntegrationServer**instances**default**config**wss**modules*\

8. After Installer completes the installation, close the wizard.

9. Apply the latest fixes for the webMethods products that Module for AS4 requires as specified in *webMethods eStandards Modules System Requirements*.

10. Start the host Integration Server.

Upgrading the webMethods Module for AS4 from 9.5 to 10.1

Upgrade the module from 9.5 to 10.1 by installing webMethods Module for AS4 10.1 in a separate installation directory, editing the properties in the config files, and migrating the data from webMethods Trading Networks.

Important:
webMethods Module for AS4 10.1 requires the supported webMethods Trading Networks version. You must upgrade all your webMethods products at the same time. For more information about upgrading your webMethods products, see the *Upgrading Software AG Products* for your release.

To upgrade the module from 9.5 to 10.1

1. Shut down all webMethods products and all other applications that are running on the machine on which you want to install webMethods Module for AS4 10.1.

   Important:
   If all products, applications, and business processes are not shut down, Installer cannot replace the key files that are locked by the operating system.

2. Back up the 9.5 WmAS4 installation directory.

3. Install webMethods Module for AS4 10.1 as described in "Installing webMethods Module for AS4 " on page 24. Specify the webMethods Module for AS4 10.1 installation directory for the installation directory.

4. Copy the ...\packages\WmAS4\config\properties.cnf file from your backup installation to your 10.1 installation.

5. Migrate the data from webMethods Trading Networks. For instructions about migrating data from webMethods Trading Networks, see *Upgrading Software AG Products*.

Uninstalling webMethods Module for AS4

To uninstall Module for AS4:

1. Shut down the Integration Server that hosts Module for AS4.

2. Start Software AG Uninstaller as follows:

System	Instructions
Windows	In the **Add or Remove Programs** window, select the **Software AG Products 10.x** *installation_directory*, where *installation_directory* is the Integration Server installation directory in which Module for AS4 is installed.
UNIX	Go to the *installation_directory*/bin directory of the installation that includes the Integration Server on which Module for AS4 is installed and enter `uninstall` (wizard) or `uninstall -console` (console mode).

3. In the product selection list, select **eStandards > webMethods Module 10.1 for AS4**.

   Uninstaller does not delete the configuration files, directory structure, or the files that you have created. To delete those files, go to *Integration Server_directory*\packages and *Integration Server_directory*\instances\default\packages. Delete the WmAS4 package and WmAS4AU package (if installed) directories.

4. Restart the Integration Server from which you uninstalled Module for AS4.

3 Defining Trading Networks Information

Overview

Module for AS4 uses Trading Networks to:

- Manage certificates.
- Define: Partner profiles, trading partner agreements, and document processing rules.

To set up Module for AS4 to work with Trading Networks, complete the following tasks:

Task 1	Define trading partner profiles. Partner profiles identify the trading partners with whom you want to send and receive AS4 messages.
Task 2	Define trading partner agreements (TPA). TPA contains a set of parameters you can use to govern how business documents are exchanged between two trading partners.
Task 3	Add or remove certificates in trading networks for your enterprise and trading partners.
Task 4	Optionally, configure TN processing rules.

Defining Partner Profiles

To use Module for AS4, define a single trading partner profile for your organization (for example, *My Enterprise*), and then you define a trading partner profile for each trading partner with whom you want to exchange AS4 messages.

Defining Your Enterprise Profile

Define the enterprise profile for your enterprise.

To create My Enterprise:

1. In My webMethods: **Administration > Integration > B2B > Partner Administration > Partner Profiles**.
2. Click **Create My Enterprise**.
3. Define the following required fields for your enterprise:

Field	Description
Corporation Name	The name of your enterprise.
Status	■ **Active**: Enables your enterprise profile. ■ **Inactive**: Disables your enterprise profile.

Field	Description
	Прим.: You must enable your enterprise profile before you can exchange documents with trading partners.
External IDs > Add ID	The value for the external ID type that your enterprise uses within documents. When exchanging documents, partners can use a well-known ID scheme such as a Data Universal Numbering System (D-U-N-S) to identify themselves. Select one of the ID types available in the Trading Networks database, and then specify the value that identifies your enterprise for that ID type.

4. Click **Save and Close**.

Defining Trading Partner Profiles

Define a trading partner profile for each trading partner with whom you want to exchange messages and files.

To create a trading partner profile

1. In My webMethods: **Administration > Integration > B2B > Partner Administration > Partner Profiles > Add Profile**.

2. Complete the following required fields for your trading partner:

Field	Description
Corporation Name	The name of your partner's enterprise.
External IDs > Add ID	The value for the external ID type that your partner uses within documents. When exchanging documents, partners typically use a well-known ID scheme, such as a D-U-N-S number, to identify themselves within a document. You must select one of the ID types available in the Trading Networks database, and then specify the value that identifies your partner's enterprise for that ID type.

3. Click **Save**.

For more information about creating trading partner profiles, see *webMethods Trading Networks Administrator's Guide*.

Defining Trading Partner Agreements

The Trading Partner Agreement (TPA) schema used by Module for AS4 is based on Processing Modes (P-Modes), the P-Modes control how AS4 messages are processed. For more information about P-Modes, see *OASIS ebXML Messaging Services 3.0 Version 1.0, Part 1, Core Features*.

Every TPA is uniquely identified by a sender, a receiver, and an agreement ID. During a transaction between trading partners, the module retrieves the appropriate TPA based on this information and uses the TPA to process the business document being exchanged. Each TPA is associated with an MEP; therefore, for every MEP between a pair of trading partners, you must define a separate TPA.

Note:
With ENTSOG, the agreement ID and TPA are optional. For more information about the optional *pmodeId*, see "wm.ip.estd.as4.msh:submit" on page 93.

Based on the message type, the module uses the appropriate leg (requestUM, requestSM, or replyUM) to process the AS4 message. For more information about configuring MEPs and legs, see "Configuring Trading Partner Agreements" on page 34.

Define TPAs in My webMethods on the **Administration > Integration > B2B > Trading Partner Agreements > Agreement Details** page. For more information about defining TPAs, see *webMethods Trading Networks Administrator's Guide*.

Managing Certificates

Trading Networks manages certificates for your enterprise and trading partners. Trading Networks certificate sets consist of sign/verify and encrypt/decrypt authentication certificates, these certificates are used by Module for AS4 to sign, unsign, encrypt, and decrypt ebMS 3.0 messages. For more information about managing certificates, see *webMethods Trading Networks Administrator's Guide*.

Defining TN Processing Rules

When WmAS4 starts, it registers the following TN document types:

ebMS.3.0 UserMessage

ebMS.3.0 Receipt Signal

ebMS.3.0 PullRequest Signal

ebMS.3.0 Error Signal

ebMS 3.0 Message Fragment

SOAP Fault

When WmAS4AU starts, it registers the following TN document types:

Agreement Update Request

Agreement Update Response

Agreement Update Exception

Agreement Termination Request

Agreement Termination Response

Agreement Termination Exception

For each of these document types, the module also registers a default processing rule named *AS4 default rule* and *AU default rule* if the WmAS4AU package is installed. This default processing rule is not mapped to any other processing rule.

You can configure TN processing rules to process any registered TN document type. For more information about configuring processing rules, see *webMethods Trading Networks Administrator's Guide*.

4 Configuring Module for AS4

Overview

Before you can exchange messages with your trading partners, you must:

1. Add and configure trading partner agreement (TPA) and parameters between two trading partners, and then configure TPA parameters.
2. Optionally, configure the AS4 configuration properties.

Configuring Trading Partner Agreements

To add and configure a trading partner agreement:

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements**.
2. Click **Add TPA** and enter the TPA details. For more information about TPA configuration, see *webMethods Trading Networks Administrator's Guide*.
3. From the list of IS document types in the WmAS4 package, type com.wm.estd.as4.documents.pmode:PMode.
4. In the TPA Data panel, define the following parameters:

TPA Parameter	Description
agreement	URI of the location that contains the partner agreement.
mepBinding	Select one of the following MEP type: ■ `One-Way/Push` ■ `One-Way/Pull` ■ `Two-Way/Sync` ■ `Two-Way/Push-Pull` ■ `Two-Way/Push-Push`
initiator	Information that identifies the initiator of the message exchange. Configure the following initiator parameters:

Parameter Name	Description
id	External ID of the initiating MSH. The contents of this parameter map to the element Messaging/UserMessage/PartyInfo/From/PartyId.
type	External ID type of the initiating MSH, as defined in the trading partner profile. For example, D-U-N-S.

TPA Parameter	Description	
	role	URI of the location where the agreement is stored
	host	Whether the partner profile is the host or not for a document exchange. If you set this parameter to 'true', then the certificates associated with this partner is selected. Configure this parameter only when the Trading Networks partner profile is not the enterprise profile during a document exchange between two partners. **Note:** Ensure that you configure either the **initiator** or the **responder** TPA parameter as the host.
	authorization	Optional. User credentials the pull signal receiver needs to authorize the pull signal. Specify username and password.
responder	Information that identifies the responder. Configure the following responder parameters:	
	Parameter	**Description**
	id	External ID of the responding MSH, as defined in the trading partner profile.
	type	External ID type of the responding MSH, as defined in the trading partner profile. For example, D-U-N-S.
	role	URI of the location where the agreement is stored.
	host	Whether the partner profile is the host or not for a document exchange. If you set this parameter to 'true', then the certificates associated with this partner is selected. Configure this parameter only when the Trading Networks partner profile is not the enterprise profile during a document exchange between two partners. **Note:** Ensure that you configure either the **initiator** or the **responder** TPA parameter as the host.
legs	Defines processing, transportation binding, and business information parameters. Add one or more legs by clicking **Add**. Click **Insert** to add a leg before the current leg. Configure the following leg parameters:	
	Parameter	**Description**

TPA Parameter	Description
label	MEP leg name. Select one of the following: ■ `requestUM`: Select when using One-Way/Push, One-Way/Pull, Two-Way/Sync, Two-Way/Push-Pull, and Two-Way/Push-Push. ■ `requestSM`: Select when using One-Way/Pull. ■ `replyUM`: Select when using Two-Way/Sync, Two-Way/Push-Pull, and Two-Way/Push-Push.
protocol	Identifies the type of transport between two MSHs and information related to the MSHs. Configure the following parameters: ■ `address`: Endpoint URI of the responding MSH. For example, http://*host*:*port*/ws/msh/receive ■ `username`: Username to authenticate the access to the partner's endpoint URI. ■ `password`: Password to authenticate the access to the partner's endpoint URI. ■ `addActorOrRoleAttribute`: Specifies whether the role attribute will be added to the message header. For more information, see "Configuring Multi-Hop Messaging at an Endpoint" on page 50
businessInfo	Identifies message exchange service information for a pair of trading partners. Configure the following parameters: ■ `service`: URI of the service that processes the message. ■ `serviceType`: Name of the service type that indicates how the sender and receiver will interpret the `service` element. If you do not enter a value for this parameter, the `service` parameter must be a URI. ■ `action`: URI of the element that identifies an operation or an activity within a service. ■ `properties`: Properties required in the message. If the required properties are missing, the processing of the message stops and an error is returned. Click Add or Insert and configure the following parameters: `name`, `description` and, `required`.

TPA Parameter	Description
	■ `mpc`: Message Partition Channel (MPC) identifier. The user message or signal will use this MPC. ■ `extendedInfo`: Configure multiple message exchange `service`, `service type`, and `action`.

5. Click **Save and Close**.

Note:errorHandling and **receptionAwareness** are optional TPA parameters.

Configuring Optional Features

Using My webMethods, you can configure TPAs to implement any or all of the module's optional features. The table below describes the options available and where to find more information about them.

When you want to...	Configure this parameter...	As described in...
Report errors	*legs/errorHandling*	"Configuring Error Handling" on page 38
Secure your AS4 message exchange	*legs/security*	"Configuring Security" on page 39
Confirm messages are delivered successfully	*legs/receptionAwareness*	"Configuring Reception Awareness" on page 45
Compress payloads	*payloadService/compressionType*	"Configuring Payload Compression" on page 46
Extract attachments	*payloadService/extractAttachment*	"Configuring Extract Attachments" on page 47
Splitting and joining messaging	*splitting*	"Configuring Splitting and Joining ebMS Messages" on page 48
Employ multi-hop messaging	*legs/protocol*	"Configuring Multi-Hop Messaging" on page 49
Store large payloads on the hard disk drive instead of in memory	Not applicable	"Configuring Large Payload Handling" on page 51
Validate agreement update message parameters	*agreementUpdateParameters*	"Configuring Agreement Update Parameters" on page 51

Configuring Error Handling

When you want Module for AS4 to report errors, configure the error handling TPA parameters in the requestUM or replyUM leg for a user message and in the requestSM leg for a pull signal.

To configure error handling

1. In My webMethods: **Applications > Administration > Integration > B2B > Trading Partner Agreements > Trading Partner Agreement Details**.

2. Select the TPA for which you want to configure error handling.

3. In the TPA Data panel, configure the `errorHandling/report` parameters as follows:

   a. `senderErrorsTo` Address or comma-separated list of addresses to which to send the ebMS errors that are generated by the sending MSH.

   b. `receiverErrorsTo` Address or comma-separated list of addresses to which to send the ebMS errors that are generated by the receiving MSH.

   c. `asResponse` Select if processing errors at the receiver end are reported on the back channel of erroneous messages. Specify one of the following:

      - `true`—Errors are reported on the back channel. This is the default.
      - `false`—Errors are reported to the *receiverErrorsTo* address.

   d. `missingReceiptNotifyProducer` Whether an error signal notification is generated and sent to the producer when a receipt is not received from the partner for the message sent. The error message is sent to the address configured in the `senderErrorsTo` parameter.

      - `false`—Error reports are not generated. This is the default.
      - `true`—Error reports are generated.

Note:
In the requestUM leg, when *missingReceiptNotifyProducer* is set to `true`, *sendReceipt* must be set to `true` and *replyPattern* can be set to `response` or `callback`. Where, for `response`: no additional settings are required and for `callback`: set *replyTo* and *replyPattern*.

- For `One-Way/Push`, either `response` or `callback` can be set.
- For `One-Way/Pull`, only `callback` can be set.
- For `Two-Way/Push-Pull`, only `response` can be set.

missingReceiptNotifyProducer cannot be configured in the replyUM leg.

Note:
Bundling of multiple errors within a single message is supported.

Configuring Security

To secure your AS4 message exchange, configure the security TPA parameters in the requestUM or replyUM leg for a user message and in the requestSM leg for a pull signal.

To configure security

1. In My webMethods **Applications >Administration > Integration > B2B > Trading Partner Agreements**.

2. Select the TPA for which you want to configure security.

3. In the TPA Data panel, configure the *security* parameters as follows:

   a. *enablesecurity* Enable or disable security

      - `true`— Security is enabled.
      - `false`— Security is disabled. This is the default.

   b. *includeTimeStamp* Includes time stamp in the security header. The default value is `false`.

      - `true`— Time stamp will be included in the security header.
      - `false`— Time stamp will not be included in the security header.

   c. *x509* Information required to sign and encrypt an AS4 message using the WSS X.509 Certificate Token Profile. Configure the following parameters:

Parameter	Value
sign	Specify if the message will be signed and which parts of the message will be signed. When you use your own policy file for security settings, the values of the *sign* parameters are ignored. Configure the following parameters:

Parameter	Value
enableSign	Whether signing is enabled. Specify one of the following: - `true`—Signing is enabled. - `false`—Signing is disabled. This is the default.
certificateId	Optional. The certificate ID to use for signing or verifying a signature for a message. The partner certificate ID must

Parameter	Value	
		be used while receiving a message. You can view the enterprise *certificateId* for the configured certificates in the My webMethods Server: **Partner Profiles > Certificates > Certificate ID** column.
	element	The XPath of each element that needs to be signed. For example, to sign the Timestamp element, add /soapenv:Envelope/soapenv:Header/wsse:Security/wsu:Timestamp Click **Add** or **Insert** to configure multiple element paths that need to be signed. Specify:
	attachments	Whether signing of the attachments for a message is enabled. Specify one of the following: ■ `true`—Signing of the attachments of a message is enabled. ■ `false`—Signing of the attachments of a message is disabled. This is the default.
	signReceipt	Whether signing of the receipt is enabled. Specify one of the following: ■ `true`—Signing of the receipt is enabled. ■ `false`—Signing of the receipt is disabled. This is the default. **Note:** If *enableSign* is set to `false` and *signReceipt* is set to `true`, the receipt will not be signed.
	receiptCertificateID	The certificate ID to use for signing the receipt.
	signReceiptBody	Whether signing the receipt body is enabled. Specify one of the following: ■ `true`—Signing of the receipt body is enabled.

Parameter	Value
	■ `false`—Signing of the receipt body is disabled. This is the default. **Note:** This parameter can be enabled only if *signReceipt* is enabled. **Parameter** / **Value** *element* — The path to the element that needs to be signed.
encrypt	Whether the message will be encrypted and which parts of the message will be encrypted. **Note:** When you use your own policy file for security settings, the values of the *encrypt* parameters are ignored. **Parameter** / **Value** *enableEncrypt* — Whether encryption is enabled. Specify one of the following: ■ `true`—Encryption is enabled. ■ `false`—Encryption is disabled. This is the default. *certificateId* — The certificate ID to use for encrypting or decrypting a message. The partner certificate ID must be used for encryption while sending a message. The enterprise certificate ID must be used for decryption. You can view the enterprise *certificateId* for the configured certificates in the My webMethods Server: **Partner Profiles > Certificates > Certificate ID** column. *element* — The XPath of each element that needs to be encrypted. For example, to encrypt the Timestamp element, add /soapenv:Envelope/soapenv:Header/wsse:Security/wsu:Timestamp Click **Add** and configure the following parameters: **Parameter** / **Value**

Parameter	Value		
		element	Each element to be encrypted. Click **Add** or **Insert** to add elements to be encrypted.
	attachments	Whether encrypting the attachments of a message is enabled. Specify one of the following: ■ `true`—Encrypting of the attachments of a message is enabled. ■ `false`—Encrypting of the attachments of a message is disabled. This is the default.	
	encryptBody	Whether to encrypt the body of a message or not. Specify one of the following: ■ `true`—Enables encryption of the message body. This is the default. ■ `false`—Disables encryption of the message body.	
algorithmSuite	Specifies the algorithm suite to be used for signing and encrypting. For more information about algorithm suites, see"Using Algorithm Suites" on page 44.		

d. *`usernameToken`* Information needed to authenticate the AS4 message. Configure the following parameters:

- *username* User name to authenticate the message.
- *password* Password to authenticate the message.
- *hashpassword* Whether password hashing is enabled. Specify one of the following:

  `true`—Password hashing is enabled.

  `false`—Password hashing is disabled. This is the default.

e. *`policyFile`* Optional. Absolute path of the policy file. The policy file must adhere to the format specified in *OASIS WS-SecurityPolicy*.

f. *`pmodeAuthorize`* Whether to authorize the messages on the MEP leg for processing. Specify one of the following:

- `true`—Messages are authorized for processing.

- `false`—Messages are not authorized for processing. This is the default.

g. *`receipt`* Information that identifies how receipts are handled. Configure the following parameters:

- *sendReceipt* Whether a receipt (Receipt ebMS signal) is sent. Specify one of the following:

  `true`—A receipt is sent.

  `false`—A receipt is not sent. This is the default.

- *replyPattern* Specifies the reply pattern of the receipt signal. Specify one of the following:

  `response`—The module sends the receipt on the back channel. This reply pattern can only be used with the One-Way/Push MEP. This is the default.

  `callback`— The module sends the receipt signal as a separate request.

- *replyTo* Specifies the endpoint URL to which the receipt is sent. You must configure this parameter when *replyPattern* is set to callback.

- *nonRepudiation* Whether the hash values for the digests in the user message should be included in the receipt. Specify one of the following:

  `true`—Hash values are included in the receipt.

  `false`—Hash values are not included in the receipt. This is the default.

4. Click **Save** or **Save and Close**.

Configuring Certificates in Trading Networks

Configure the certificates for your enterprise and partners in Trading Networks and reload Module for AS4 after configuring the certificates. For more information about configuring certificates in Trading Networks, see *webMethods Trading Networks Administrator's Guide*.

Note:
Reload Module for AS4 whenever you update a certificate in Trading Networks for the security settings to take effect.

Configuring Receipts

Follow this procedure to configure Module for AS4 to handle receipts.

To configure receipts:

1. In My webMethods: **Applications >Administration > Integration > B2B > Trading Partner Agreements**.

2. Select the TPA for which you want to configure receipts.

3. Configure the `receipt` parameters as follows:

   - *sendReceipt* Whether a receipt (Receipt ebMS signal) is sent. Specify one of the following:

     `true`—Send a receipt.

     `false`—Do not send a receipt. This is the default.

   - *replyPattern* Specifies the reply pattern of the receipt signal. Specify one of the following:

     `response`—The module sends the receipt on the back channel. This reply pattern can only be used with the One-Way/Push MEP. This is the default.

     `callback`— The module sends the receipt signal as a separate request.

   - *replyTo* Specifies the endpoint URL to which the receipt is sent. You must configure this parameter when *replyPattern* is set to callback.

   - *nonRepudiation* Whether receipts are configured for Non-repudiation of Receipt (NRR). Specify one of the following:

     `true`—Receipts are configured for NRR. The generated receipt includes ds:Reference elements that contain digests of the original message parts.

     `false`—Receipts are configured for reception awareness. The generated receipt contains a copy of the user message or message fragment structure of the received AS4 message. This is the default.

     Note:
     When the *sendReceipt* parameter is false, the *nonRepudiation* parameter is ignored.

4. Click **Save** or **Save and Close**.

Using Algorithm Suites

After installing Module for AS4, you can use the following algorithm suites:

Algorithm Suite Name	Signing Algorithm	Encryption Algorithm
Default	sha1	tripledes-cbc
Basic128	sha1	aes128-cbc
TripleDes	sha1	tripledes-cbc
Basic128Rsa15	sha1	aes128-cbc
TripleDesRsa15	sha1	tripledes-cbc
Basic192Sha256	sha256	aes192-cbc
Basic128Sha256	sha256	aes128-cbc

Algorithm Suite Name	Signing Algorithm	Encryption Algorithm
TripleDesSha256	sha256	tripledes-cbc
Basic128Sha256Rsa15	sha256	aes128-cbc
TripleDesSha256Rsa15	sha256	tripledes-cbc
Advanced128Sha256GCM	sha256	aes128-gcm
Advanced256Sha256GCM	sha256	aes256-gcm
Basic256	sha1	aes256-cbc
Basic192	sha1	aes192-cbc
Basic256Rsa15	sha1	aes256-cbc
Basic192Rsa15	sha1	aes192-cbc
Basic256Sha256	sha256	aes256-cbc
Basic192Sha256	sha256	aes192-cbc
Basic256Sha256Rsa15	sha256	aes256-cbc
Basic192Sha256Rsa15	sha256	aes192-cbc

Configuring Reception Awareness

If reception awareness is enabled, an initiating MSH sends a user message to the responding MSH. If the responding MSH does not send a receipt within a specified time period, the initiating MSH retries sending the message. This continues until the receipt is received or the value specified in the *maxRetries* parameter is reached. The activity log in Trading Networks is updated with a message about each retry.

Duplicate detection detects a duplicate user message with the same UserMessage/MessageInfo/MessageId as a previous message. When a duplicate message is received, the message is flagged as duplicate and UserStatus in Trading Networks transaction analysis page is updated to `DUPLICATE_RECEIVED`.

Configure Module for AS4 to retry sending messages by configuring the reception awareness TPA parameters in the requestUM leg.

To configure reception awareness

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements > Trading Partner Agreement Details**.

2. Select the TPA for which you want to configure reception awareness.

3. In the TPA Data panel, specify the following `receptionAwareness` parameters:

   a. `enabled`: Select `true` to enable and `false` to disable reception awareness. The default is set to false.

   b. `retry`: Select parameters that control push retry.

   - *enabled*: Select one of the following:

     `true`—Enabled.

     `false`—Disabled. This is the default.

   - Configure the following *retryParameters* that controls push retry:

     maxRetries: Type the maximum number of times the module will try resending a message.

     period: Type the length of time, in seconds, the module waits between retry attempts.

   c. Select the following `duplicateDetection` parameters:

   - *enabled*: Select *true* to enable and *false* to disable duplicate detection.
   - *checkWindow* Optional. Length of time, in seconds (S) or days (D), that a message ID is retained in the cache and checked for duplicate incoming message IDs (for example, 4320000S or 5D).
   - *maxSize* Optional. The maximum size, in bytes, of an incoming duplicate message that the module will save. When the size of the incoming duplicate message exceeds this value, the duplicate message is not saved in the Trading Networks database and an error is logged.

4. Click **Save** or **Save and Close**.

Note:
Duplicate detection will start only after the *duplicateDetection* parameter is enabled in Trading Networks. All messages received before enabling *duplicateDetection* are not considered.

Configuring Payload Compression

Module for AS4 is capable of providing configurable compression and decompression of application payloads. AS4 messages that contain compressed or application payloads are built according to the SOAP with Attachments specification. Each compressed payload is carried in its own MIME body part.

When you want to send a compressed payload, set the PMode.PayloadService.CompressionType P-Mode parameter to application/gzip, as described below. Enabling this P-Mode parameter indicates to the sending MSH that the outgoing message payloads should be compressed before sending. Therefore, the receiving MSH must decompress the payload part before it delivers the message to the receiver.

Note:
When the incoming message contains a compressed payload part(s), the receiving MSH decompresses the payload irrespective of the value of the TPA's CompressionType P-Mode parameter.

To configure the module to compress the payload of an AS4 message

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements > Trading Partner Agreement Details**.
2. Select the TPA for which you want to configure compression.
3. In the TPA data panel, configure the *payloadService > compressionType* parameter. Specify one of the following:
   - `none`—Payload compression is disabled. This is the default.
   - `application/gzip`—Payload compression is enabled.
4. Click **OK**.

Note:
Payload compression is applied before applying splitting or signing and encrypting.

Configuring Extract Attachments

Configure Module for AS4 to extract attachments in ebMS 3.0 user message as a separate transaction.

To configure the module to extract attachments from the ebMS 3.0 user message

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements**.
2. Select the TPA for which you want to extract attachments for payloads.
3. In the TPA data panel, configure the *payloadService > extractAttachment* parameter. Specify one of the following:
   - `false`—Attachments in ebMS 3.0 user message are not extracted. This is the default.
   - `true`—Attachments in ebMS 3.0 user message are extracted as a separate transaction.
4. Click **OK**.

Note:
- *extractAttachment* is supported only for XML and EDI document types.
- The extracted attachment is identified as XML or EDI if a matching document type exists in Trading Networks, otherwise, the document type is identified as *Unknown*.

Configuring Splitting and Joining ebMS Messages

Message splitting and joining involves two related operations. During the Send operation of a large user message, the sending MSH splits the message into multiple fragments. Each fragment is wrapped in its own SOAP message with a MessageFragment header extension element and then sent to the receiving MSH in a separate thread from a thread pool. The thread pool properties (that is, the as4.throttling properties) can be configured for optimal performance. For more information, see "Configuring AS4 Configuration Properties" on page 52.

During the Receive operation, the presence of a MessageFragment header element indicates that the message content is part of a larger message. Once all the fragments have been received, they are reassembled and joined into a single message again.

The set of fragments that make up a user message belong to one group. Each group is assigned one of the following statuses:

Value	Description
ACTIVE	One or more fragments of the group was received and there are still more fragments to come.
COMPLETE	All fragments in the group were received.
EXPIRED	A group fragment was received after the time set for the *joinInterval* parameter elapsed.
REJECTED	There was a problem during processing of one of the fragments in the group.

When a user message is split and then separate fragments are sent, the TPA parameters for security and reception awareness apply at the fragment level, not the source message level. That is, the message fragments are individually secured. In addition, message acknowledgments, retries, and duplicate detection apply at the fragment level.

Note:
Receipts cannot be generated for joined user messages.

Module for AS4 provides built-in services to manage fragment groups. For more information, see "Built-In Services" on page 91.

To configure splitting and joining of ebMS messages

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements > Trading Partner Agreement Details**.

2. Select the TPA for which you want to configure splitting and joining.

3. In the TPA data panel, configure the `splitting` parameter as follows:

Parameter	Value
enabled	Optional. Whether or not splitting is enabled. Specify one of the following: ■ `true`—Enabled. ■ `false`—Disabled. This is the default.
fragmentSize	The size of each fragment, in bytes. For example, if *fragmentSize* is defined as 1000 bytes and the message is 2050 bytes, the message is split into two fragments of 1000 bytes each, and one fragment of 50 bytes. If the message is 900 bytes, one fragment of 900 bytes is sent.
joinInterval	The maximum time, in seconds, to expect and process additional fragments after the first fragment is received.

Configuring Empty Conversation ID

Allows empty conversation ID. allowEmptyConversationId parameter is added in com.wm.estd.as4.documents.pmode:PMode document. By default, this parameter is not set. If the value is not set for this parameter in the Trading Partner Agreement, then the property as4.message.emptyConversationId is used.

Value	Meaning
True	Empty conversation Id is allowed.
False	Empty conversation Id is not allowed.

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements >Trading Partner Agreement Details**

2. Select the TPA for which you want to configure compression.

3. In the TPA data panel, configure the allowEmptyConversationId parameter.

Configuring Multi-Hop Messaging

Multi-hop is used by organizations that use intermediaries (MSHs in this case) for message delivery. Messages flow through a multi-hop path, a connection of public or private networks and clouds. With intermediaries responsible for routing functions, the communicating parties can ignore message destination details.

Module for AS4 supports forwarding messages using both standard and custom routing functions.

When Module for AS4 acts as the sender or receiver endpoint, a TPA is required. When the module acts as an intermediary, a TPA is not required. Multi-hop configuration at the intermediary involves configuring a routing function only.

Configuring Multi-Hop Messaging at an Endpoint

A TPA must be configured for multi-hop messaging when Module for AS4 acts as the sender or receiver endpoint.

To configure multi-hop messaging for an endpoint

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements > Trading Partner Agreement Details**.

2. Select the TPA for which you want to configure multi-hop messaging.

3. In the TPA data panel, configure the *protocol* parameters as follows:

Parameter	Value
address	The endpoint address of the next MSH.
addActorOrRoleAttribute	Specifies whether or not the role attribute with a value of `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/part2/200811/nextmsh` will be added to the ebMS 3.0 messaging header of an outbound user message. Valid values are: ■ `true`—The role attribute is added to the message header. ■ `false`—The role attribute is not added to the message header.

Configuring Multi-Hop Messaging at an Intermediary

When Module for AS4 is the intermediary, only a routing function needs to be configured.

To configure multi-hop messaging at an intermediary

1. Open the *Integration Server_directory*\Instances*Instance_Name*\Packages\WmAS4\config\config.cnf file in a text editor.

2. Configure the as4.multihop.routingfunction property.

   For more information, see "Configuring AS4 Configuration Properties" on page 52.

Configuring Large Payload Handling

When a large payload is sent and received as a MIME attachment, Module for AS4 can store the payload on the hard disk drive rather than keeping it in memory. This results in more efficient message processing.

To configure the module to handle large payloads

1. In the *Integration Server_directory*\Instances*Instance_Name*\Packages\WmAS4\config\config.cnf file, configure the as4.payload.large.enable and the as4.payload.large.threshold properties. For more information, see "Configuring AS4 Configuration Properties" on page 52.

2. In Integration Server Administrator, on the **Settings > Extended** screen, click **Edit Extended Settings**.

3. Specify values for the following Integration Server configuration parameters:

   - `watt.server.tspace.location`
   - `watt.server.tspace.max`
   - `watt.server.tspace.timeToLive`

   For more information about defining Integration Server configuration properties, see *webMethods Integration Server Administrator's Guide*.

4. Click **Save Changes**.

5. Restart Integration Server.

Configuring Agreement Update Parameters

Configure Module for AS4 with `agreementUpdateParameters` to validate the AgreementUpdate message parameters: `au:RespondBy`, `au:ExpireBy`, `au:TerminateBy`, and `au:ActivateBy` against `agreementUpdateParameters` if passed in a request message.

These values should be configured in days. For example, if `respondBy` is configured with `5` days, then the response for the received request is sent within 5 days. If the request message expects a response within `4` days or lesser with `au:RespondBy` set to `4` days along with the `au:CreatedAt` value, then validate service generates a `RespondByRejected` error.

Note:
If `agreementUpdateParameters` are not configured, the validation of agreement updates is skipped by the validation service. For more information, see "wm.estd.as4.au.handlers:validate" on page 113.

To configure the module to validate agreement updates

1. In My webMethods: **Administration > Integration > B2B > Trading Partner Agreements > Trading Partner Agreement Details**.

2. Select the TPA for which you want to configure agreement update parameters.

3. In the TPA data panel, configure the following *agreementUpdateParameters* parameters:

Value	Description
respondBy	Number of days by which a response is expected.
expireBy	Number of days after which agreement is not valid.
terminateBy	Number of days by which agreement is terminated.
activateBy	Number of days by which a change is expected to be implemented and activated for use.

4. Click **OK**.

Configuring AS4 Configuration Properties

Use the AS4 configuration properties to control some of the module functions.

To configure AS4 properties

1. Open the *Integration Server_directory*instances*Instance_Name*\packages\WmAS4\config\config.cnf file in a text editor.

2. Configure the following properties:

Property	Definition
as4.cacheTPAs	Whether TPAs are cached. Valid values are: ■ true—The module caches TPAs. ■ false—The module does not cache TPAs. This is the default. **Note:** To improve performance in your production environment, Software AG recommends setting this property to true.

Property	Definition
as4.message.emptyConversationId	Whether empty conversation IDs are supported. Valid values are: ■ `true` - Empty conversation IDs are supported. ■ `false` - Empty conversation IDs are not supported. This is the default.
as4.default.requestTimeOut	The length of time, in milliseconds, the module waits for a response to an AS4 request. The default is 30000.
as4.message.securityHeader.timeToLive	The length of time, in seconds, that the ebMS message security header is valid. The default is 3600.
as4.multihop.routingFunction	The routing service Module for AS4 invokes to determine the address of the next MSH. When this property is not configured, the module invokes the default wm.ip.estd.as4.util:routingFunction service to determine this address.
as4.nonprimary.httpPorts	Comma-separated list of non-primary HTTP ports expecting AS4 messages. Configure this property on the receiver to allow incoming AS4 messages passage through non-primary HTTP ports of Integration Server.
as4.nonprimary.httpsPorts	Comma-separated list of non-primary HTTPS ports expecting AS4 messages. Configure this property on the receiver to allow incoming AS4 messages passage through non-primary HTTPS ports of Integration Server.
as4.peppol.proxy.enable	Indicates whether the HTTP proxy server is enabled. Valid values are: ■ `true`—The HTTP proxy server is enabled. ■ `false`—The HTTP proxy server is disabled.
as4.peppol.proxyAlias	The HTTP proxy server alias value.
as4.twowaysync.requestTimeOut	The amount of time, in milliseconds, the module waits for responding MSH to submit a reply user message in Two-Way/Sync . The default is 3000.

Property	Definition
as4.payload.large.enable	Whether large payload handling is enabled. Valid values are: ■ `true`—Enabled. ■ `false`—Disabled. This is the default. **Note:** A payload is considered large when its size is greater than the value specified in the as4.payload.large.threshold property.
as4.payload.large.threshold	The payload size in bytes over which a document is considered large. For example, if as4.payload.threshold is set to 1000000, the module considers any payload greater than 1,000,000 bytes as large. **Note:** This property should be configured to 16384 or greater. If no value is specified, or if the value configured is less than 16384, the module sets this property to 16384.
as4.pmode.validation	Whether TPA parameters are validated. Valid values are: ■ `true`—The module validates a TPA before using it to send or process a user message or pull signal. This is the default. ■ `false`—The module does not validate a TPA before using it to send or process a user message or pull signal. **Note:** To improve performance in your production environment, Software AG recommends setting this property to false once you have finished configuring and verifying all TPA parameters.
as4.rg.http.url	Comma-separated list of URLs of the Reverse HTTP Gateway Server. Configure this property if your Integration Server is configured to receive messages from external clients through a Reverse HTTP Gateway

Property	Definition
	Server over HTTP. Use the values configured for **Host** and **Port** in Integration Server Administrator on the Security > Ports > View Internal Server Details screen. Use the following format: `hostname:port` where *hostname* is the IP address or the name of the machine, and *port* is the HTTP port number. For example, workstation4.webmethods.com:5555
as4.rg.https.url	Comma-separated list of URLs of the Reverse HTTP Gateway Server. Configure this property if your Integration Server is configured to receive messages from external clients through a Reverse HTTP Gateway Server over HTTPS. Use the values configured for **Host** and **Port** in Integration Server Administrator on the Security > Ports > View Internal Server Details screen. Use the following format: `hostname:port` where *hostname* is the IP address or the name of the machine, and *port* is the HTTPS port number. For example, workstation5.webmethods.com:6666.
as4.throttling.keepAliveTime	The maximum length of time an idle thread pool waits to receive a new task before terminating. The default is 100 seconds.
as4.throttling.maxThreadPoolCount	The maximum number of threads allowed in the thread pool. The default value is the number of processors available to the JVM.
as4.throttling.minThreadPoolCount	The minimum number of threads allowed in the thread pool, including idle threads. The default value is 1.
as4.throttling.queueSize	The size of the thread pool queue into which the tasks to be executed are submitted. The default is 20.

3. Click **Save**.

4. Restart Integration Server.

5 Viewing Information About AS4 and AS4AU Transactions

Overview

This chapter describes viewing AS4 and AS4AU transactions, as well as user statuses that Module for AS4 sets while processing transactions.

Viewing AS4 and AS4AU Transactions

You can view the following types of information about your AS4 and AS4AU transactions:

- Sender and receiver of the message
- Date and time the message was received in Trading Networks
- User status of the message
- Document type
- Related messages, for example, the receipt for a user message or fragment
- Processing status
- On the **Contents** tab, the following will be displayed based on the type of AS4 or AS4AU message being sent:
  - ebMS envelope or fragment envelope, which contains the message envelope
  - Secured ebMS message when security is enabled
  - Payload or fragment data
- On the **Activity Log** tab, the steps that the message goes through during inbound or outbound processing

For more information about viewing transactions, see *webMethods Trading Networks User's Guide*.

User Status for AS4/AS4AU Messages

The following tables list the user statuses that the module sets while processing AS4 or AS4AU messages. You can view this information in the Transactions panel of the Transactions page in My webMethods.

Status Values for User Messages at the Sending Side

User Status	Meaning
SUBMITTED	Using the submit service, a user message was submitted to the module for processing.
SENT	The user message was sent to the trading partner.
TOBE_FRAGMENTED	The user message was submitted to the split operation.

User Status	Meaning
FRAGMENTED	The user message was split into fragments successfully.
SEND_ERROR	An error occurred while sending the user message to the trading partner. If retry is enabled, the message transmission will be retried according to the settings defined in the TPA's *retry* parameter.
ERROR_RECEIVED	An error signal or SOAP fault was received from the trading partner. If retry is enabled, the message transmission will be retried according to the settings defined in the TPA's *retry* parameter.
WAIT FOR RECEIPT	The message was sent successfully and the module is waiting for the partner to send a receipt signal.
RECEIPT_RECEIVED	The receipt signal was received for the user message that was sent.
ATTEMPTS_EXHAUSTED	A receipt signal was not received. Retry is enabled and the retry attempts were exhausted.
RETRYING:n	A receipt signal was not received. Retry is enabled and the module has attempted to resend the message *n* number of times. For example, RETRYING:2 indicates that the module is retrying to send the message for the second time.
QUEUED	The message was submitted and it is queued in an MPC, waiting for a pull signal.
PROCESSED	The message was processed successfully.
RESPONSE_RECEIVED	The user message response was received for the requested message.
CONFIRMED	The agreement update message is validated and processed successfully.
REJECTED	The agreement update message is not validated successfully.
VALIDATION_ERROR	An error occurred during the validation of the agreement update message.
IGNORED	The agreement update message is ignored when there is a validation error in the response message.

Status Values for Message Fragments at the Sending Side

User Status	Meaning
SUBMITTED	A message fragment was submitted to the module for sending.
SENT	The message fragment was sent to the trading partner.
SEND_ERROR	An error occurred while sending the message fragment to the trading partner. If retry is enabled, the message fragment transmission will be retried according to the settings defined in the TPA's *retry* parameter.
ERROR_RECEIVED	An error signal or SOAP fault was received from the trading partner. If retry is enabled, the message fragment transmission will be retried according to the settings defined in the TPA's *retry* parameter.
WAIT FOR RECEIPT	The message fragment was sent successfully and the module is waiting for the partner to send a receipt signal.
RECEIPT_RECEIVED	The receipt signal was received for the message fragment that was sent.
ATTEMPTS_EXHAUSTED	The receipt signal was not received. Retry is enabled and the retry attempts were exhausted.
RETRYING:n	The receipt signal was not received. Retry is enabled and the module has attempted to resend the message fragment *n* number of times. For example, RETRYING:2 indicates that the module is retrying to send the message fragment for the second time.

Status Values for Pull Signals at the Sending Side

User Status	Meaning
SEND	The module received a request to send a pull signal, and the pull signal was sent to the trading partner.
SEND_ERROR	The pull signal could not be sent to the trading partner.
SENT	The pull signal was sent to the trading partner.
ERROR_RECEIVED	The trading partner responded to the pull signal with an error signal or a SOAP fault.

Status Values for User Messages at the Receiving Side

User Status	Meaning
RECEIVED	The user message was received and it will be processed next.
ROUTED	The user message was forwarded to the next MSH instead of being processed.
PROCESSED	The message was processed successfully.
MESSAGE_IN_ERROR	An error occurred during processing of the received message. Based on the configuration of the *errorHandling* parameters, either an error signal or a SOAP fault was returned to the message sender. For more information, see "Configuring Error Handling" on page 38.
DUPLICATE_RECEIVED	Duplicate detection is enabled, and the message received was a duplicate.
RESPONSE_RECEIVED	The user message response was received for the requested message.
RESPONSE_SENT	The user message response was sent to the requested message.
SUBMITTED	Using the submit service, a user message was submitted to the module for processing.
SENT	The user message was sent to the trading partner.
CONFIRMED	The agreement update message is validated and processed successfully.
REJECTED	The agreement update message is not validated successfully.
PROCESSING_ERROR	An error occurred during the processing of the agreement update message.
VALIDATION_ERROR	An error occurred during the validation of the agreement update message.
IGNORED	The agreement update message is ignored when there is a validation error in the response message.

Status Values for Message Fragments at the Receiving Side

User Status	Meaning
RECEIVED	A message fragment was received and is being processed.

User Status	Meaning
PROCESSED	The message fragment was processed successfully.
MESSAGE_IN_ERROR	An error occurred while processing the received message fragment.

Status Values for Pull Signals at the Receiving Side

User Status	Meaning
RECEIVED	A pull signal was received and is being processed.
PROCESSED	The pull signal was processed successfully.
MESSAGE_IN_ERROR	An error occurred while processing the received pull signal.

6 Logging and Error Handling

Overview

Module for AS4 supports the following error handling capabilities for all MEPs:

- The responding MSH is able to report errors to the initiating MSH as either an ebMS error message or as a SOAP fault
- Errors can be sent on the back channel of the underlying transport protocol
- Errors can be reported to a third-party address

A list of error codes and supporting information appears at the end of this chapter.

Error Message Logging

The module uses the Integration Server logging mechanism to log messages. To view the server log, use Integration Server Administrator. For more information about viewing the server log, see *webMethods Integration Server Administrator's Guide*.

The module adds Error, Warn, Info, and Debug messages to the server log, using the format *AS4.000n.nnnn*, where:

- AS4 is the product code that indicates the message is issued by the module.
- *000n* is the major error code, where *n* can be any of the following values:
  - 0—The module encountered a general error.
  - 1—The module encountered an error while receiving an AS4 message.
  - 2—The module encountered an error while sending an AS4 message.
- *nnnn* is the minor error code.

Error Codes for AS4

0000.3005 Error relating bizdoc with Document ID {0} to bizdoc with Document ID {1} as {2}.

Explanation: An error occurred while retrieving the user message related to the receipt or error signal from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3004 Error updating Processing Status to {0} and User Status to {1} for bizdoc with Document ID {2} and Internal ID {3}. Error message: {4}.

Explanation: Error updating Processing Status and User Status for a message document in Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3003 Error retrieving partnerId from Trading Networks for External Id Value {0}, Type {1}. Error message: {2}.

Explanation: An error occurred while retrieving the partnerId parameter from Trading Networks.

Action: Ensure the partner profile in Trading Networks is configured correctly.

0000.3002 Error while saving bizdoc for message with Message Id {0} in Trading Networks. Error message: {1}.

Explanation: An error occurred while saving the message in Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3001 Validation error for TPA with AgreementID {0}: The mandatory keystore parameters required for Security are not set in the partner TPA.Details: {1}

Explanation: One of the security parameters specified in the error message is not configured in the TPA.

Action: Configure the mandatory security parameters in the TPA and try again.

0000.3000 Error occurred during startup of WmAS4 package. Error message: {0}.

Explanation: An error occurred while starting the WmAS4 package.

Action: See the error message for details, fix the reported issue, and restart Integration Server.

0000.0006 The message partition channel {0} is empty.

Explanation: A pull request was received for the specified MPC but this MPC contains no messages.

Action: Ensure that there are messages in the specified MPC.

0000.0103 Message with MessageId {0} does not comply with the security parameters configured in the TPA. Error message: {1}.

Explanation: The processor determined that the message's security methods, parameters, scope or other security policy-level requirements or agreements were not satisfied.

Action: Ensure the security parameters in the partner TPA are configured correctly.

0000.0102 Decryption failed for message with MessageId {0}. Error message: {1}.

Explanation: The security module could not decrypt the encrypted data reference that the security header intended for the SOAP actor.

Action: Confirm that the private key configured for the Enterprise in the Integration Server keystore is correct.

0000.0101 Signature verification failed for message with MessageId {0}. Error message: {1}.

Explanation: The security module could not validate the signature in the security header that was intended for the SOAP actor.

Action: Ensure the public certificates configured for the trading partner in the Integration Server keystore are correct.

0000.3042 Validation error for TPA with AgreementID {0}: If security needs to be enabled, one of the following parameters needs to be set in the TPA: policy file, userNameToken, sign, or encrypt.

Explanation: Security is enabled but the required security parameters are not configured in the TPA.

Action: Configure at least one of the parameters specified in the error message.

0000.3008 Internal error. Error processing input \n {0}.

Explanation: An internal error occurred.

Action: Contact Software AG Global Support.

0000.3009 Error retrieving TPA for senderID {0} receiverID {1} agreementID {2} from Trading Networks. Error message: {3}.

Explanation: An error occurred while retrieving the TPA from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3011 Error retrieving bizdoc with Internal ID {0} from Trading Networks. Error message: {1}.

Explanation: An error occurred while retrieving the bizdoc from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3012 Error retrieving bizdoc with Document ID {0} from Trading Networks. Error message: {1}.

Explanation: An error occurred while retrieving the bizdoc from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3013 Error retrieving content part {0} from bizdoc with Internal ID {1} from Trading Networks. Error message: {2}.

Explanation: An error occurred while retrieving content part from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3014 Error retrieving message with MessageId {0} referenced by {1} {2} from Trading Networks. Error message: {2}.

Explanation: An error occurred while retrieving the message from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3015 Error persisting message {0} in Trading Networks. Error message: {1}.

Explanation: An error occurred while persisting the message in Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3017 Error while retrieving leg with label {0} from TPA with AgreementID {1} for service {2} and action {3}.

Explanation: An error occurred while retrieving the specified leg from the designated TPA.

Action: Ensure that the leg specified in the message is configured in the designated TPA.

0000.3020 Error while retrieving TPAs. Error message: {0}.

Explanation: An error occurred while retrieving the module's TPAs from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3021 Error retrieving TPA for mpc {0} and Agreement ID {1} from Trading Networks.

Explanation: The module was unable to retrieve the TPA while generating or processing a pull signal.

Action: Ensure at least one TPA is configured in Trading Networks with the Message Partition Channel specified in the message.

0000.3022 Authorization information for {0} is not configured for the TPA with AgreementID {1}.

Explanation: An error occurred while retrieving authorization information for the initiator or responder from the TPA with the agreement ID specified in the error message.

Action: Ensure that the required authorization information is configured in the TPA.

0000.3023 Error while retrieving content for the MIME attachment referenced by Content-ID {0}. Error message: {1}.

Explanation: An error occurred while retrieving the content for the MIME attachment.

Action: For a generated user message, ensure the partInfo and partContent parameters of the submit service are valid. For a received user message, consult with the trading partner who sent the message to resolve the error.

0000.3024 Error retrieving content from the SOAP Body referenced by id {0}. Error message: {1}.

Explanation: An error occurred while retrieving the payload content from the SOAP body.

Action: For a generated user message, ensure the partInfo and partContent parameters of the submit service are valid. For a received user message, consult with the trading partner who sent the message to resolve the error.

0000.3025 Error while retrieving content for the external resource referenced by the URL {0}. Error message: {1}.

Explanation: An error occurred while retrieving payload content from the URL.

Action: For a generated user message, ensure the partInfo and partContent parameters of the submit service are valid. For a received user message, consult with the trading partner who sent the message to resolve the error.

0000.3028 Validation error for TPA with AgreementID {0}. Parameter {1} is not configured.

Explanation: In the specified TPA, a required parameter has not been configured.

Action: In the specified TPA, configure the required parameter.

0000.3029 Validation error for TPA with AgreementID {0}. Leg with label {1} is not configured.

Explanation: The specified leg is not configured in the TPA.

Action: In the TPA, configure the leg specified in the message.

0000.3030 Validation error for TPA with AgreementID {0}. Parameter {1} is not configured in leg with label {2}.

Explanation: In the designated TPA, a mandatory leg parameter is not configured for the leg specified.

Action: In the TPA, configure the leg parameter specified in the message.

0000.3031 Validation error for TPA with AgreementID {0}. Parameter protocol/address within leg with label {1} is an invalid URL.

Explanation: In the designated TPA, the protocol/address value is not configured with a valid URL for the leg specified.

Action: In the TPA, configure the protocol/address parameter as a valid URL for the leg specified.

0000.3032 Validation warning for TPA with AgreementID {0}. Parameter businessInfo/service within leg {1} must be configured as a URI.

Explanation: In the specified TPA leg, the businessInfo/serviceType parameter is not configured and the businessInfo/service parameter is not configured as a URI.

Action: In the specified TPA leg, ensure that either the businessInfo/serviceType parameter is configured or the businessInfo/service is configured with a valid URI.

0000.3033 Validation error for TPA with AgreementID {0}. Parameter businessInfo/service within leg {1} must be configured with the value "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service".

Explanation: In the specified TPA leg, the businessInfo/action parameter is configured with the value "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test", but the businessInfo/service parameter is not configured as "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service" as required.

Action: In the specified TPA leg, ensure that the values of the businessInfo/action parameter and the businessInfo/service parameter are consistent with the ebMS version 3.0 specification.

0000.3034 Internal Error: Error during initialization of logger, JournalLog Handler could not be retrieved.

Explanation: Logger initialization failed.

Action: Contact Software AG Global Support.

0000.3035 Error loading AS4 config file "packages/WmAS4/config/config.cnf". Error message: {0}.

Explanation: There was an error loading the config.cnf file.

Action: Ensure the config.cnf file is at the specified location. If it is not present, contact Software AG Gobal Support.

0000.3036 Failed to retrieve TPA for fromPartyId {0}, fromPartyIdType {1}, toPartyId {2}, toPartyIdType {3}, and pmodeId {4}. Error message: {5}.

Explanation: The TPA for the specified parameters could not be retrieved.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3037 Error while loading policy file {0}.

Explanation: The specified policy file, which is used for security purposes, could not be loaded.

Action: Ensure that the content of the policy file has the correct syntax.

0000.3038 Error retrieving message for RefToMessageId {0} from Trading Networks. Error message: {1}.

Explanation: An error occurred while retrieving the user message from Trading Networks.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3039 Error while processing the {0} message with messageId {1}. Error message: {2}.

Explanation: An error occurred while retrieving the TPA for the specified message.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3040 P-Mode authorization for pull signal failed for the mpc {0}. The authorization parameters do not match.

Explanation: P-Mode authorization failed because the authorization check failed.

Action: In the TPA, ensure the Initiator username and password are configured correctly.

0000.3041 P-Mode authorization for pull signal failed. The input message has PMode authorization parameters, but the PMode authorization in the TPA is disabled.

Explanation: The input message contains P-Mode authorization parameters, but P-Mode authorization in the TPA is disabled.

Action: In the TPA, set the security/pmodeAuthorize parameter to true to enable P-Mode authorization.

0000.3058 Validation error for TPA with AgreementID {0}. Parameter errorHandling/report/asResponse within leg must be set to false to enable splitting.

Explanation: The errorHandling/report/asResponse parameter within the leg is not set to false as required to enable splitting.

Action: Set errorHandling/report/asResponse to false and configure errorHandling/report in the leg.

0000.3059 Validation error for TPA with AgreementID {0}. Parameter pmode/splitting/fragmentSize should be greater than zero

Explanation: Parameter pmode/splitting/fragmentSize is not greater than zero.

Action: Set the value of fragmentSize to a positive, non-zero integer.

0000.3060 Validation error for TPA with AgreementID {0}. Parameter security/receipt/replyPattern within leg with leg requestUM must be configured to callback for splitting

Explanation: The security/receipt/replyPattern parameter within the requestUM leg is not configured to callback for splitting.

Action: Within the requestUM leg, configure security/receipt/replyPattern as callback and configure security/receipt/replyTo with the address to which the receipt should be sent.

0000.3043 Error while processing the security for the message. Error message {0}.

Explanation: An error occurred while processing the security of the message.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3045 Error while setting the algorithm suite for security. Error message {0}.

Explanation: An error occurred while configuring the security parameters.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3044 Error while retrieving the Integration Server keystore details for keystore alias {0} and key alias {1}.

Explanation: An error occurred while retrieving the Integration Server keystore details.

Action: Ensure the specified ISKeystoreAlias and keyAlias parameters are configured correctly in Integration Server.

0000.3049 Error updating UserStatus of bizdoc with Internal ID {0} to {1}. Error Message: {2}.

Explanation: An error occurred while updating the user status of the bizdoc.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3051 Error adding content for {0} to SOAP Body. Error message: {1}.

Explanation: An error occurred while adding an input payload to the SOAP Body.

Action: Ensure that the payload has XML-structured content.

0000.3052 The partInfo input parameter is not configured for partContent [{0}]. The specified partContent will not be added as a payload part in the outbound ebMS message.

Explanation: There is no value for the partInfo input parameter, which acts as metadata for the partContent parameter. The module cannot resolve the location for partContent, so partContent is ignored and omitted from the message.

Action: Provide a value for the partInfo parameter for the partContent specified in the warning message.

0000.3027 Validation warning for TPA with AgreementID {0}.Parameter {1} is not configured in leg with label {2}.

Explanation: Validation of the TPA identified by the AgreementID has failed for reasons specified by the error message.

Action: See the error message and rectify the TPA accordingly.

0000.3036 Encountered error while retrieving PMode for fromPartyId {0}, fromPartyIdType {1}, toPartyId {2}, toPartyIdType {3}, fromRole {4}, toRole {5}, service {6}, action {7}, agreementRef {8}, and pmodeId {9}. Error details: {10}

Explanation: The TPA for the specified parameters could not be retrieved.

Action: See the error message for details and take appropriate action to resolve the problem.

0000.3053 Validation error for TPA with AgreementID {0}. Parameter errorHandling/report/receiverErrorsTo must be configured in leg with label requestUM, when the errorHandling/report/asResponse parameter is configured as false.

Explanation: The errorHandling/report/asResponse parameter is configured as false (that is, errors are be reported on the back channel); however, the address to which errors should be reported is not configured in the requestUM leg.

Action: In the requestUM leg, configure the errorHandling/report/receiverErrorsTo parameter to indicate the address to which errors should be reported.

0000.3054 Validation error for TPA with AgreementID {0}. Parameter errorHandling/report/receiverErrorsTo within leg with label requestUM contains an invalid URL.

Explanation: The errorHandling/report/receiverErrorsTo parameter within the leg with label {1} contains an invalid URL.

Action: Ensure that the errorHandling/report/receiverErrorsTo parameter within the requestUM leg in the TPA specified by the error message contains a list of comma separated valid URLs.

0000.3055 Validation error for TPA with AgreementID {0}. Parameter security/receipt/replyTo must be configured in leg with label requestUM, when the security/receipt/sendReceipt is configured as true and the security/receipt/replyPattern parameter is configured as callback.

Explanation: The security/receipt/sendReceipt is configured as true and the security/receipt/replyPattern parameter is configured as callback, but the security/receipt/replyTo parameter in the requestUM leg is not configured.

Action: Configure the security/receipt/replyTo parameter in the requestUM leg for the TPA specified by the error message.

0000.3056 Validation error for TPA with AgreementID {0}. Parameter security/receipt/replyTo within leg with label requestUM contains an invalid URL.

Explanation: The security/receipt/replyTo parameter within the requestUM leg contains an invalid URL.

Action: Configure the security/receipt/replyTo parameter within the requestUM leg of the TPA specified by the error as a valid URL.

0000.3057 Validation error for TPA with AgreementID {0}. Parameter security/receipt/replyPattern within leg with label requestUM must be configured to callback for One-Way/Pull

Explanation: The security/receipt/replyPattern parameter in the requestUM leg is not configured as callback, as required for the One-Way/Pull MEP.

Action: Configure the security/receipt/replyPattern parameter as callback.

0000.3066 There is no requestUM leg with mpc {0}. The mpc name configured in the requestSM leg and requestUM leg should be same

Explanation: The mpc configured in the requestSM leg of the TPA does not match the mpc configured in the request UM leg of the TPA.

Action: Edit the TPA so that the value of BusinessInfo/mpc in the requestSM leg and the value of BusinessInfo/mpc in requestUM leg are the same.

0000.3067 TPA configuration Error. Password should not be null or empty for the username {0}.

Explanation: The username is set but the password is null or empty.

Action: Set the password parameter to a non-empty value.

0000.3074 as4.payload.large.threshold is configured as {0}. The property must be configured to a value greater than 16KB. The module will use a threshold value of 16KB instead.

Explanation: The as4.payload.threshold property is less than 16KB.

Action: Configure the as4.payload.threshold property to greater than 16KB.

0001.3000 An error occurred while generating Receipt for the received UserMessage which has messageId {0}. Error message {1}.

Explanation: An error occurred while generating the receipt for the received user message.

Action: See the error message for details and take appropriate action to resolve the problem.

0001.3001 Error while processing the received message with messageType {0}, messageId {1}. Error message: {2}.

Explanation: An error occurred while processing the received message.

Action: See the error message for details and take appropriate action to resolve the problem.

0001.3002 Non-Repudiation check for received Receipt with messageId {0} failed. URI {1} in the UserMessage with messageId {2} does not have a match.

Explanation: The non-repudiation check failed.

Action: Consult with the trading partner to ensure the partner sent the correct non-repudiation information.

0001.3004 Received a duplicate message with messageId {1}.

Explanation: A duplicate message was detected.

Action: Notify the trading partner that the message sent was a duplicate.

0001.3005 Invalid ebMS Header error. Error Message: The Messaging element must contain at least one of the child elements UserMessage or SignalMessage.

Explanation: The inbound message's ebMS message header does not contain a UserMessage or SignalMessage element.

Action: Notify the partner that the message header was incomplete.

0001.3006 Invalid ebMS Header error. Error message: The SignalMessage element must contain at least one of the child elements PullRequest or Error or Receipt.

Explanation: The SignalMessage element of the inbound message does not contain at least one of these elements: PullRequest, Error, or Receipt.

Action: Notify the trading partner that a message with an invalid SignalMessage element was received.

0001.3007 EBMS processing error. Error Message: Either the UserMessage/PartyInfo element is missing from the ebMS header or the element is encrypted.

Explanation: The module is unable to retrieve the UserMessage/PartyInfo element from the inbound message's ebMS header.

Action: Notify the message sender.

0001.3008 EBMS processing error. Error Message: Either the UserMessage/PartyInfo/From element is missing from the ebMS header or the element is encrypted.

Explanation: The module is unable to retrieve the UserMessage/PartyInfo/From element from the inbound message's ebMS header.

Action: Notify the message sender.

0001.3009 EBMS processing error. Error Message: Either the UserMessage/PartyInfo/From/PartyId element is missing from the ebMS header or the element is encrypted.

Explanation: The module is unable to retrieve the From/PartyId element from the inbound message's ebMS header.

Action: Notify the message sender.

0001.3010 Inconsistent value error. Error Message: The UserMessage/PartyInfo/From/PartyId element cannot be empty.

Explanation: The module is unable to retrieve a value from the PartyId element of the inbound message's ebMS header.

Action: Notify the message sender.

0001.3011 EBMS processing error. Error Message: Either the UserMessage/PartyInfo/To element is missing from the ebMS header or the element is encrypted.

Explanation: The module is unable to retrieve the UserMessage/PartyInfo/To element from the inbound message's ebMS header.

Action: Notify the message sender.

0001.3012 EBMS processing error. Error Message: Either the UserMessage/PartyInfo/To/PartyId element is missing from the ebMS header or the element is encrypted.

Explanation: The module is unable to retrieve the To/PartyId element from the inbound message's ebMS header.

Action: Notify the message sender.

0001.3013 Inconsistent value error. Error Message: The UserMessage/PartyInfo/From/PartyId element cannot be empty.

Explanation: The module is unable to retrieve a value from the From/PartyId element from the inbound message's ebMS header.

Action: Notify the message sender.

0001.3014 EBMS processing error. Error Message:The UserMessage/CollaborationInfo/AgreementRef element is required by the module to identify the agreement for processing an inbound message.

Explanation: The UserMessage/CollaborationInfo/AgreementRef element is missing.

Action: Notify the message sender.

0001.3015 EBMS processing error. Error Message:The UserMessage/CollaborationInfo/AgreementRef value is required by the module to identify the agreement for processing an inbound message.

Explanation: The UserMessage/CollaborationInfo/AgreementRef value is missing.

Action: Notify the message sender.

0001.3016 Feature not supported error. Error Message: Processing of Bundled messages is not currently supported.

Explanation: Processing of bundled messages is not supported currently.

Action: Contact Software AG Global Support for any inquiries regarding support of this feature.

0001.3018 Invalid ebMS Header error. Error Message: {0}.

Explanation: Validation of the ebMS header has failed due to the reason specified in the error message.

Action: Notify the message sender.

0001.3019 EBMS validation error. Error Message: {0}.

Explanation: The EBMS header validation process failed for reasons other than an invalid header.

Action: Report the problem described in the error message to Software AG Global Support.

0001.3020 Inconsistent value error: Error Message: The content of the From/PartyId element MUST be a URI if the PartyId/@type attribute is not present.

Explanation: The content of the PartyId element is not a valid URI as defined in the specification.

Action: Notify the message sender that the PartyId element does not conform to ebMS specification.

0001.3021 Inconsistent value error: Error Message: The content of the To/PartyId element MUST be a URI if the PartyId/@type attribute is not present.

Explanation: The content of the PartyId element is not a valid URI as defined in the specification.

Action: Notify the message sender that the PartyId element does not conform to ebMS specification.

0001.3022 Inconsistent value error: Error Message: The content of the AgreementRef element MUST be a URI if the AgreementRef/@type attribute is not present.

Explanation: The content of the AgreementRef element is not a valid URI as defined in the specification.

Action: Notify the message sender that the AgreementRef element does not conform to ebMS specification.

0001.3023 Inconsistent value error: Error Message: The content of the Service element MUST be a URI if the Service/@type attribute is not present.

Explanation: The content of the Service element is not a valid URI as defined in the specification.

Action: Notify the message sender that the Service element does not conform to ebMS specification.

0001.3025 EBMS processing warning. Warning Message: Either the From/PartyId/@type attribute is not present or attribute is empty.

Explanation: The module is unable to extract the From/PartyId/@type attribute value from the ebMS header of the inbound message.

Action: Notify the message sender to set the From/PartyId/@type attribute with the appropriate value.

0001.3026 EBMS processing warning. Warning Message: Either the To/PartyId/@type attribute is not present or attribute is empty.

Explanation: The module is unable to extract the To/PartyId/@type attribute value from the ebMS header of the inbound message.

Action: Notify the message sender to set the From/PartyId/@type attribute with the appropriate value.

0001.3027 Incoming Message does not comply with the security parameters configured in the TPA. Error message: {0}.

Explanation: The content of the element specified in the incoming message is not compatible with the expected content based on the associated TPA.

Action: Notify the message sender.

0001.3028 Processing mode mismatch error. Error Message: The expected content of the element {0} is {1}, the actual content is {2}.

Explanation: The content of the element specified in the error message is not compatible with the expected content based on the associated TPA.

Action: Notify the message sender.

0001.3029 Processing mode mismatch error. Error Message: The Messaging/eb:UserMessage/eb:MessageProperties element is expected with the required properties {0}.

Explanation: The content of the element specified in the error message is not compatible with the expected content based on the associated TPA.

Action: Notify the message sender.

0001.3030 Processing mode mismatch error. Error Message: The Messaging/eb:UserMessage/eb:MessageProperties element is expected to contain the required property {0}.

Explanation: The content of the element specified in the error message is not compatible with the expected content based on the associated TPA.

Action: Notify the message sender.

0001.3031 Security processing warning. Warning Message: Skipping security processing for the message of type {0} with messageId {1} as the TPA leg is not available.

Explanation: The module cannot process the received message's security headers because it can't retrieve a TPA leg to process the received Error or Receipt signal. If the

SignalMessage/MessageInfo/RefToMessageId element is present, however, the module continued to process the message.

Action: Check the error logs to see if there was an error resolving the TPA or retrieving a TPA leg.

0001.3033 EBMS processing error. Error Message:The UserMessage/CollaborationInfo/Service element is missing or encrypted.

Explanation: The UserMessage/CollaborationInfo/Service element is missing or encrypted.

Action: Notify the message sender.

0001.3034 EBMS processing error. Error Message:The UserMessage/CollaborationInfo/Service value is missing.

Explanation: The UserMessage/CollaborationInfo/Service value is missing.

Action: Notify the message sender.

0001.3035 EBMS processing error. Error Message:The UserMessage/CollaborationInfo/Action element is missing or encrypted.

Explanation: The UserMessage/CollaborationInfo/Action element is missing or encrypted.

Action: Notify the message sender.

0001.3036 EBMS processing error. Error Message:The UserMessage/CollaborationInfo/Action value is missing.

Explanation: The UserMessage/CollaborationInfo/Action value is missing.

Action: Notify the message sender.

0001.3037 No ebMS Error generated for the message in error {0}.

Explanation: A message was received in error but the module did not generate an ebMS error.

Action: None.

0001.3038 Unable to retrieve errorHandling parameters for the message in error {0}. Error message: the TPA leg is not available.

Explanation: The module was unable to retrieve the TPA leg for the message in error, so it cannot report the generated ebMS error.

Action: None.

0001.3039 Error while decompressing the received payload part {0}, Error message: {1}

Explanation: Decompressing the payload failed.

Action: Ensure the incoming payload is formatted correctly, is compressed, and is not corrupted.

0001.3040 The received fragment with groupID {0} has the MessageSize set. MessageSize has already been received by a previous fragment of this group. Rejecting this fragment group.

Explanation: The fragment group is rejected. More than one fragment in the group has MessageSize configured.

Action: Consult with the trading partner to ensure that only one fragment in the group has MessageSize configured.

0001.3041 The received fragment with groupID {0} has the FragmentCount set. FragmentCount has already been received by a previous fragment of this group. Rejecting this fragment group.

Explanation: The fragment group is rejected. More than one fragment in the group has the FragmentCount parameter configured.

Action: Consult with the trading partner to ensure that only one fragment in the group has FragmentCount configured.

0001.3042 The received fragment with groupID {0} has the MessageHeader set.MessageHeader has already been received by a previous fragment of this group. Rejecting this fragment group.

Explanation: The fragment group is rejected. More than one fragment has MessageHeader configured.

Action: Consult with the trading partner to ensure that only one fragment in the group has MessageHeader configured.

0001.3043 The received fragment with groupID {0} has the Action parameter set. Action parameter has already been received by a previous fragment of this group. Rejecting this fragment group.

Explanation: The fragment group is rejected. More than one fragment in the group has the Action parameter configured.

Action: Consult with the trading partner to ensure that only one fragment in the group has the Action parameter configured.

0001.3044 The received fragment with groupID {0} has the CompressionInfo parameter set. CompressionInfo parameter has already been received by a previous fragment of this group. Rejecting this fragment group.

Explanation: The fragment group is rejected. More than one fragment in the group has CompressionInfo configured.

Action: Consult with the trading partner to ensure that only one fragment in the group has CompressionInfo configured.

0001.3045 The fragment received is a duplicate fragment. A frament with the same messageID is received before.

Explanation: The fragment group is rejected. The group contains duplicate fragments.

Action: Notify the trading partner that duplicate fragments were received.

0001.3046 The received fragment with groupID {0} has the FragmentNum value greater than the FragmentCount. Rejecting this fragment group.

Explanation: The fragment group is rejected. At least one of the fragments received has a FragmentNum value that exceeds the value of FragmentCount.

Action: Consult with the trading partner to ensure that the FragmentNum value for all fragments is not greater than the FragmentCount.

0001.3047 The received fragment with groupID {0} has the FragmentCount value which is lesser than the FragmentCount received by a previous fragment. Rejecting this fragment group.

Explanation: The fragment group is rejected. The value of FragmentCount for this fragment is less than the value of FragmentCount for a fragment received previously.

Action: Consult with the trading partner to ensure the correct FragmentCount values were sent.

0001.3048 The received fragment belongs to the group {0}. This is a rejected group. The fragment is rejected.

Explanation: The fragment is rejected. It belongs to a rejected group.

Action: Consult with the trading partner and take appropriate action.

0001.3049 The received fragment belongs to the group {0}. This is a completed group. The fragment is rejected.

Explanation: The fragment is rejected. It belongs to a completed group.

Action: Consult with the trading partner and take appropriate action.

0001.3050 The received fragment belongs to the group {0}. The JoinInterval time has expired. All the fragments of this group will be rejected.

Explanation: The fragment is rejected. It belongs to an expired group.

Action: Consult with the trading partner and take appropriate action.

0001.3093 No PMode is configured for senderID {0} and receiverID {1} with agreementRef {2}.

Explanation: The TPA for the specified parameters could not be retrieved.

Action: See the error message for details and take appropriate action to resolve the problem.

0001.3094 Duplicate key found while updating cache {0} with key {1} for PMode ID {2}. PMode ID {3} has same cache entry.

Explanation: There are duplicate key entries in the sender ID, receiver ID, from role, to role, agreementRef, service and action PMode parameters within the cache.

Action: Check the related TPAs and modify it accordingly.

0001.3095 Unable to retrieve certificate for hostInternal ID {0}, partnerInternal ID {1}, and signCertificate ID {2}.

Explanation: The signing certificate is not configured for the respective partners.

Action: Make sure to configure the signing certificate in the partner profiles of the respective partners in Trading Networks.

0001.3096 No private key is configured for hostInternal ID {0}, partnerInternal ID {1}, and signCertificate ID {2}.

Explanation: The signing certificate's private key is not configured for the host.

Action: Make sure to configure the private key in the partner profiles for the host in Trading Networks.

0001.3097 Unable to retrieve certificate for hostInternal ID {0}, partnerInternal ID {1}, and encryptCertificate ID {2}.

Explanation: The encryption certificate is not configured for the respective partners.

Action: Make sure to configure the encrypt certificate in the partner profiles of the respective partners in Trading Networks.

0001.3098 No private key is configured for hostInternal ID {0}, partnerInternal ID {1}, and encryptCertificate ID {2}.

Explanation: The decryption certificate's private key is not configured for the host.

Action: Make sure to configure the private key in the partner profiles for the host in Trading Networks.

0001.3099 Encountered validation error for TPA with AgreementID {0}. Either security/receipt/replyPattern must be set to response or receptionAwareness/retry must be enabled when errorHandling/report/missingReceiptNotifyProducer is enabled.

Explanation: The TPA parameters for MEP are configured incorrectly.

Action: In the TPA, configure the correct paramters for the specified MEP.

0001.3101 Encountered validation error for TPA with Agreement ID {0}. Both requestUM and replyUM legs must be configured for Two-Way/Push Push MEP.

Explanation: You must configure both, requestUM and replyUM legs for Two-Way/Push Push MEP in TPA.

Action: Configure both, requestUM and replyUM legs for Two-Way/Push Push MEP in TPA.

0001.3103 Unable to create bizdoc for the doc id {0}, payload id {1}, and mimeType {2}. Details: {3}

Explanation: Document is not recognized by the system as it may not be supported.

Action: Ensure that you have the respective document type installed and enabled.

0001.3104 Encountered validation error for TPA with AgreementID {0}. The security/receipt/sendReceipt parameter must be configured to true if the errorHandling/report/missingReceiptNotifyProducer parameter is configured as true.

Explanation: The TPA parameters for MEP are configured incorrectly.

Action: In the TPA, configure the correct paramters for the specified MEP.

0002.3001 The wm.ip.estd.as4.msh:submit service failed to process the input pipeline {0}. Error message {1}.

Explanation: The submit service failed to send the message.

Action: See the error message for details and take appropriate action to resolve the problem.

0002.3003 Error while processing the retry parameters. The module will continue as though retry parameters are not set. Error message: {0}.

Explanation: The retry parameters specified are invalid. The module will continue as though no retry parameters are set.

Action: See the error message for details and take appropriate action to resolve the problem.

0002.3011 There was an error while processing the content for payload {0}. Error message: {1}

Explanation: When executing the wm.ip.estd.as4.msh:submit service, processing part of the content provided in the partContent input parameter failed.

Action: See the error message for details and take appropriate action. Also, verify the partContent mapping.

0002.3012 There was an error while large payload handling for the payload {0}. Error message: {1}

Explanation: Handling the large payload failed.

Action: See the error message for details and take appropriate action.

0002.3013 Unsupported content format for payload {0}. The module provides support for payload content as stream only

Explanation: The wm.ip.estd.as4.msh:submit service failed because part of the content provided in the partContent input parameter is not formatted correctly.

Action: Ensure that the content provided in the partContent input parameter is a stream.

0002.3017 {0} is a required input field. Specify a value for this field

Explanation: No value is set for the mandatory field specified in the error message.

Action: Define a value for the field specified in the error message.

0002.3018 One or more of the entries of partContent input passed to the 'submit' service is null

Explanation: One or more of the entries of the partContent parameter is missing input for the partInfo parameter.

Action: Ensure that input is specified for the partInfo parameter for each of the partContent entries.

0002.3020 An error in the input to the 'submit' service. The number of entries in the partContent and partInfo inputs do not match

Explanation: The number of entries in the partContent and partInfo are not the same.

Action: Ensure that the number of entries in the partContent and partInfo parameters are the same.

0002.3030 Encountered error while sending message using PModeID {0} with Part ID {1}. Compression of the SOAP body for the AS4 message is not supported.

Explanation: The AS4 message cannot be delivered as a SOAP body because the compression of the payload is enabled.

Action: Send the payload as an attachment or disable the compression.

0002.3031 Encountered error while sending message using PModeID {0}. PartInfo should not be empty if there are multiple partContent fields.

Explanation: The AS4 message cannot be delivered because partInfo is empty.

Action: Add partInfo for each of the partContent fields.

0003.3061 Compression type PMode ID mismatch for message Id {0} with payload ID {1}. Expected {2} but received {3}.

Explanation: There is compression type PMode ID mismatch due to unexpected behavior.

Action: Either update the TPA parameter payloadService>compressionType with the respective compression type or ask your partner to send the payload accordingly.

Error Codes for AS4 Agreement Update

0001.8000 [AU] Encounterd error during the start-up of the WmAS4AU package. Error details: {0}

Explanation: An error occurred while starting the WmAS4AU package.

Action: See the error message for details, fix the reported issue, and restart Integration Server.

0001.8002 [AU] Failed to load pmode for sender id {0}, receiver id {1}, and agreement ref {2}.

Explanation: An error occurred while loading pmode using the input provided.

Action: Ensure that the TPA is configured with the input specified or specify correct input.

0001.8003 [AU] Encountered validation error while generating {0} message. Parameter {1} is required.

Explanation: An error occurred while generating the AU message because the mandatory parameter {1} is not configured.

Action: See the error message for details, type the value of parameter {1}, and execute the service again.

0001.8004 [AU] Encountered validation error while generating {0} message. Parameter {1} should not be empty.

Explanation: An error occurred while generating the AU message because the parameter {1} is empty.

Action: See the error message for details, type the value of parameter {1}, and execute the service again.

0001.8005 [AU] Validation error for TPA with AgreementID {0}. Parameter value configured in 'businessInfo/action' is incorrectly set to {1} when 'businessInfo/Service' is {2}.

Explanation: An error occurred while validating the TPA because the parameter 'businessInfo/action' is incorrectly configured.

Action: In the specified TPA, configure the appropriate parameter.

0001.8006 [AU] The bizdoc and message input parameters are missing. Either bizdoc or message is required.

Explanation: The required input parameters are missing.

Action: Ensure that either the bizdoc or message input parameter is specified.

0001.8007 [AU] fromParty, toParty, and pmodeId input parameters are missing.

Explanation: The fromParty, toParty, and pmodeId input parameters are required when the AU message is processed for validation.

Action: Specify the required parameters when the AU message is passed as string in the message for validation.

0001.8008 [AU] Unable to retrieve request bizdoc for the input response message validation.

Explanation: Request bizdoc is required for the response message validation.

Action: Ensure that the request bizdoc is available for validating the response message.

0001.8009 [AU] Cannot recognize the message type.

Explanation: Message type is not recognized as an AU message.

Action: Specify a valid AU message.

0001.8010 [AU] Schema validation failed for the AU message. Details: {0}

Explanation: AU message is not valid as per schema.

Action: See the error message for details, fix the reported issue.

0001.8011 [AU] Encounterd error while retreving bizdoc using internalId {0}.

Explanation: An error occurred while retrieving the bizdoc from database.

Action: See the error message for details and fix the reported issue.

0001.8012 [AU] Encounterd error during agreement update validation or processing for certificate. Pmode parameters legs/security/enableSecurity, legs/security/x509/sign/enableSign, and legs/security/x509/encrypt/enableEncrypt are disabled in leg.

Explanation: An error occurred during certification validation or processing due to disabled legs/security/enableSecurity, legs/security/x509/sign/enableSign and legs/security/x509/encrypt/enableEncrypt Pmode parameters. These parameters must be configured.

Action: Configure the parameters in the TPA.

0001.8013 [AU] Encounterd error during agreement update validation or processing for certificate. Pmode parameter legs/security/x509/sign/certificateId and legs/security/x509/encrypt/certificateId are not configured in leg.

Explanation: An error occurred during certification validation or processing due to missing configuration for legs/security/x509/sign/certificateId and legs/security/x509/encrypt/certificateId Pmode parameter. These parameters must be configured with correct values.

Action: Configure the parameters in the TPA.

0001.8014 [AU] Processing failed due to a missing bizdoc input parameter.

Explanation: Processing of the request or response message failed due to a missing bizdoc input parameter.

Action: Specify a valid bizdoc input parameter.

0001.8015 [AU] Error encountered while parsing message. Agreement Update message contains multiple payloads.

Explanation: Message with the Agreement Update should contain a single Agreement Update/Termination payload.

Action: Ensure that the message has a single Agreement Update payload.

0001.8016 [AU] Processing failed. X509Data is either missing or present more than once.

Explanation: Processing failed either because X509Data is missing in the message or is present multiple times.

Action: Ensure that the message has only one X509Data.

0001.8017 [AU] Agreement Update Request either does not have a specialization for the UpdateRequest abstract element or has a wrong specialization as substitute.

Explanation: As the UpdateRequest is an abstract element, the initiator must use a non-abstract element that substitutes the abstract element.

Action: Provide the correct specialization for the abstract element in the input message. For example, the Certificate Update Request is a specialization of the Agreement Update Request to update the X.509 certificates.

0001.8018 [AU] Processing the message has failed because the ds:X509Certificate is missing.

Explanation: Processing the message has failed because the ds:X509Certificate is missing in the request message.

Action: Ensure that the request message has the correct ds:X509Certificate element.

0001.8019 [AU] Processing failed. Unable to retrieve message from bizdoc.

Explanation: Failed to extract payload from bizdoc either due to improper or missing information.

Action: Ensure that the processed bizdoc has correct payload information.

7 Peppol in AS4 estandards

What is Peppol?

Peppol is a set of artifacts and specifications enabling cross-border eProcurement. The use of Peppol is governed by a multi-lateral agreement structure owned and maintained by OpenPEPPOL. Peppol enables trading partners to exchange standards-based electronic documents over the Peppol network (based on a 4-corner model). These documents include e-Orders, e-Advance Shipping Notes, eInvoices, eCatalogues, Message Level Responses, and so on.

Peppol access Points connect users to the Peppol network and electronic exchange documents based on the Peppol specifications. Buyers and suppliers can choose their preferred single access point provider to connect to all Peppol participants already on the network. This feature enables you to connect once and connect to all.

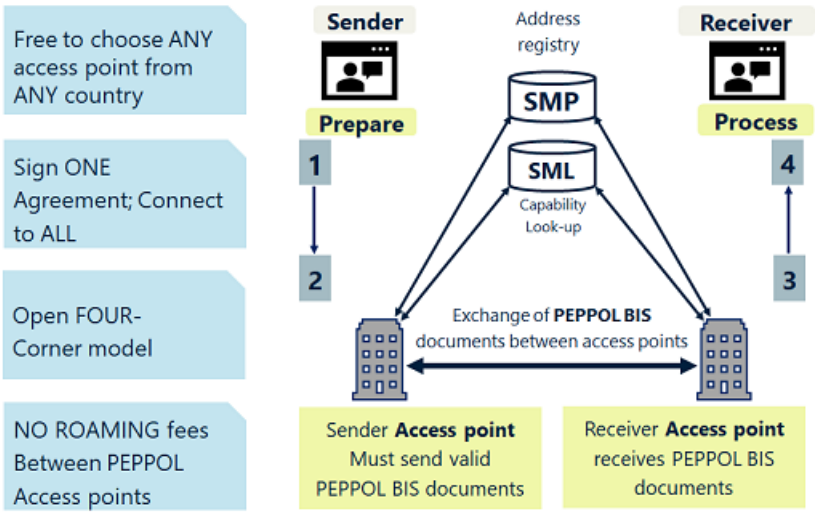
Feature

Peppol capabilities are:

- Peppol eDelivery Network - the network.
- Peppol Business Interoperability Specifications (BIS) - the document specifications.
- Peppol Transport Infrastructure Agreements (TIA) - the legal framework that defines the network governance.

Peppol eDelivery Network

The Peppol uses the eDelivery Network to connect different eProcurement systems by establishing common business processes and technical standards. This service provides an interoperable and secure network connecting all access points using the same electronic messaging protocol and formats and applying digital signature technologies to secure message content.



Peppol SMPs (publishing the capabilities of Peppol participants)

To deliver electronic documents from a sender to the correct recipient, all Peppol access points need to know about each other. To do this Peppol maintains one centralized service, called the Service Metadata Locator (SML). The Peppol SML defines which Service Metadata Publisher (SMP) to use for finding out the delivery details of any Peppol participant.

Peppol SML (central registration system for addressing)

To deliver electronic documents from a sender to the correct recipient, all Peppol access points need to know about each other. To do this Peppol maintains one centralized service called the Service Metadata Locator (SML). The Peppol SML defines which Service Metadata Publisher (SMP) to use for finding out the delivery details of any Peppol participant.

System Assets

The startup service of AS4 module creates the assets.

The corresponding TPA data have the default values as per the Peppol specification. These are required for submitting the payload in the peppol network. The agreement id for the default TPA is *Peppol*, and for sender and receiver the value is *Unknown*.

The following parameters are listed in the TPA:

Parameter	TPA
IS Document Type	com.wm.estd.as4.documents.pmode:PMode_V1
agreement	urn:fdc:peppol.eu:2017:agreements:tia:ap_provider
mepBinding	One-Way/Push
initiator/role	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/initiator
responder/type	urn:fdc:peppol.eu:2017:identifiers:ap
responder/role	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/responde
legs[0]/label	requestUM
legs[0]/businessInfo/mpc	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultMPC
legs[0]/security/receipt/sendReceipt	true
payloadService/validate	both

Note:

When you create any custom TPA, the agreement id must be *Peppol* and IS Document Type value must be `com.wm.estd.as4.documents.pmode:PMode_V1`.

Ensuring Peppol Compliant for AS4 module

The following are the pre-requisites before using the module:

- Organization must have Peppol participant identifier(Peppol ID).
- Configure the Peppol ID in the enterprise TN profile in IDType urn:fdc:peppol.eu:2017:identifiers:ap.

  Note:
  When adding externalID value with IDType urn:fdc:peppol.eu:2017:identifiers:ap toenterprise or any partner, it must be CNAME(ParticipantID) format, where CNAME is thecname from the certificate. It must not have spaces when adding externalID in the abovespecific format.
- Configure your certificate in the enterprise TN profile. You must register with Service MetadataPublisher(SMP) provider with capabilities like endpoint url, document types supported so onfor the exchange of document in Peppol network. The SMP server must support OpenPeppolSMP specification.
- You must have the details of recipient Peppol Participant Identifier (PPID). For moreinformation on SMP providers, see https://peppol.eu.

To send the message using as4 in Peppol network run the services in the following order:

1. wm.ip.estd.as4.peppol:getReceiverDetails
2. wm.ip.estd.as4.peppol:addPartnerOrUpdateCertificate
3. wm.ip.estd.as4.peppol:constructSubmitInput
4. wm.ip.estd.as4.msh:submit

For more information about Peppol services, see "Summary of Elements for the WmAS4 Package" on page 92.

A Built-In Services

Overview

This chapter describes the built-in services that Module for AS4 supports.

Summary of Elements for the WmAS4 Package

The following services are provided with the WmAS4 package:

Element	Description
wm.ip.estd.as4.msh:submit	Enables the sender to submit a payload to the initiating MSH.
wm.ip.estd.as4.msh:userMessageSubmit	Enables the sender to submit a user message to the initiating MSH.
wm.ip.estd.as4.msh:sendPull	Enables the receiver to send a pull signal to the responding MSH.
wm.ip.estd.as4.util.pull:clearMPC	Removes bizDocIDs that are cached in Module for AS4.
wm.ip.estd.as4.util.pull:getMPCContents	Returns the list of bizDocIDs queued in the specified MPC.
wm.ip.estd.as4.util.multihop:routingFunction	Returns the information needed to forward a received message in a multi-hop exchange.
wm.ip.estd.as4.util.retry:cancelRetry	Cancels the Retry that is scheduled for a given user message.
wm.ip.estd.as4.util.split:getGroupList	Retrieves a list of group IDs of the message fragments that have a given status.
wm.ip.estd.as4.util.split:clearGroup	Resets the status of a group.
wm.ip.estd.as4.peppol:getReceiverDetails	Retrieves the details endpoint address, recipient certificate of recipient participant.
wm.ip.estd.as4.peppol:addPartnerOrUpdateCertificate	Creates the recipient partner profile in Trading Networks given the certificate of the recipient.
wm.ip.estd.as4.peppol:constructSubmitInput	Constructs the input for the wm.ip.estd.as4.msh:submit service in order to send the business document.
wm.ip.estd.as4.peppol:validateContent	Validates Peppol Billing BIS 3.0 Invoices and Credit Notes.

wm.ip.estd.as4.msh:submit

Enables the sender to submit a payload to the initiating MSH, which packages the payload into a user message and sends it to the responding MSH. The MEP settings in the P-Mode control whether the message is sent immediately or queued in an MPC until a pull request is received.

Note:
The endpointUrl input field is applicable for Peppol bound documents only.

Input Parameters

refToMessageId	**String** Optional. Identifier used in Two-Way MEPs where the responding MSH user message refers to the initiating MSH user message.
messageId	**String** Optional. Unique identifier of the user message. If no value is specified, Module for AS4 generates a unique *messageId*. When the message is generated, the value of this parameter maps to Messaging/UserMessage/MessageInfo/MessageId.
toPartyId	**String**. External ID of the party receiving the message, as defined in the trading partner profile.
toPartyIdType	**String**. External ID type of the party receiving the message, as defined in the trading partner profile.
toPartyIdRole	**String** Optional. The initiator or responder role of the party in the message exchange.
fromPartyId	**String**. External ID of the party sending the message, as defined in the trading partner profile.
fromPartyIdType	**String**. External ID type of the party sending the message, as defined in the trading partner profile.
fromPartyIdRole	**String** Optional. The initiator or responder role of the party in the message exchange.
conversationId	**String** Optional. Identifier of the set of related messages that make up a conversation between two parties. If no value is specified, Module for AS4 generates a unique ID. The value of this property maps to Messaging/UserMessage/CollaborationInfo/ConversationId.
pmodeId	**String** Optional. TPA agreement ID to use for the submit request.
	Note:

If *pmodeId* is empty, then a tuple of From/PartyId, From/Role, To/PartyId, To/Role, Service, Action, and AgreementRef is used to identify the TPA agreement.

agreementRef **String** Optional. AgreementRef value to use for the submit request.

property **Document List** Optional. List of name-value pairs that are sent along with the message.

These pairs map to the zero or more Property child elements within Messaging/UserMessage/MessageProperties.

Parameter	Description
name	**String**. Name of property.
value	**String**. Value of property.

partInfo **Document List** Optional. Metadata for the *partContent* parameter.

The values specified map to the Messaging/UserMessage/PayloadInfo/PartInfo element.

Parameter	Description
type	**String**. Way in which the *partContent* needs to be packaged. Valid values are: ■ MIME PART—*partContent* is packaged as MIME attachment. ■ SOAP BODY—*partContent* is packaged in the SOAP body. ■ EXTERNAL REFERENCE—*partContent* is an external reference.
id	**String**. ID of the *partContent*.
schemaLocation	**String**. URI of the schema. The value of this parameter maps to the *location* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaVersion	**String**. Version identifier of the schema. The value of this parameter maps to the *version* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaNamespace	**String** Optional. Target namespace of the schema.

The value of this parameter maps to the *namespace* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.

description **String**. Description of the *partContent*.

The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/Description element.

property **Document List**. List of name value pairs that are sent along with the message.

The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties element.

Note:
If you want the final AS4 payload of MIME type application/octet-stream, then add the property *name* to be of MIME type and *value* as application/octet-stream.

Parameter	Description
name	**String**. Name of the property. The value of this parameter maps to the name attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.
value	**String**. Value of the property. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.

partContent **InputStream**. Array of all the MIME attachments. Each element within the array must have a corresponding *partInfo*.

Note:
If *partInfo* is not specified for *partContent*, then SOAP BODY is considered as the default *partInfo* type.

soapAction **String** Optional. Value of the *action* parameter in the MIME type.

service **String** Optional. Name of service used to identify the RequestUM leg to be used for the current transaction.

The module uses the *service* and *action* parameters to determine which RequestUM leg to use for the current transaction. The module compares the

value of this parameter to the value of *leg/businessInfo/service* that is configured in all RequestUM legs of the TPA. The RequestUM leg that contains a match for both the *service* and *action* parameters is the leg that will be used for the current transaction.

When there is only one RequestUM leg configured in the TPA, the *service* parameter does not have to be defined.

action **String** Optional. Name of action used to identify the RequestUM leg to be used for the current transaction.

The module uses the *service* and *action* parameters to determine which RequestUM leg to use for the current transaction. The module compares the value of this parameter to the value of *leg/businessInfo/action* that is configured in all RequestUM legs of the TPA. The RequestUM leg that contains a match for both *service* and *action* is the leg that will be used for the current transaction.

When there is only one RequestUM leg configured in the TPA, the *action* parameter does not have to be defined.

endpointUrl **String**. Endpoint address of the recipient partner.

The endpointUrl input field validates only in Peppol bound document. Therefore, the *originalSender* and *finalRecipient* properties must be specified when you submit the message in Peppol network.

Output Parameters

error **Document**. Any error that was encountered while processing the submission.

Parameter	Description
code	**String**. ebms error code, if any.
description	**String**. Description of the error.

wm.ip.estd.as4.msh:userMessageSubmit

Enables the sender to submit a user message to the responding MSH, which packages the payload into a user message and sends it to the initiating MSH.

Input Parameters

refToMessageId **String**. Identifier that refers to the request message's *messageId*.

messageId **String** Optional. Unique identifier of the user message. If no value is specified, Module for AS4 generates a unique *messageId*.

When the message is generated, the value of this parameter maps to

Messaging/UserMessage/MessageInfo/MessageId

.

partInfo **Document List**. Metadata for the *partContent* parameter.

The values specified map to the Messaging/UserMessage/PayloadInfo/PartInfo element.

Parameter	Description
type	**String**. Way in which the *partContent* needs to be packaged. Valid values are: ■ MIME PART—*partContent* is packaged as MIME attachment. ■ SOAP BODY—*partContent* is packaged in the SOAP body. ■ EXTERNAL REFERENCE—*partContent* is an external reference.
id	**String**. ID of the *partContent*.
schemaLocation	**String**. URI of the schema. The value of this parameter maps to the *location* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaVersion	**String**. Version identifier of the schema. The value of this parameter maps to the *version* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaNamespace	**String**. Target namespace of the schema. The value of this parameter maps to the *namespace* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
description	**String**. Description of the *partContent*. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/Description element.
property	**Document List**. Optional. List of name value pairs that are sent along with the message.

The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties element.

Note:
If you want the final AS4 payload of MIME type application/octet-stream, then add the property *name* to be of MIME type and *value* as application/octet-stream.

Parameter	Description
name	**String**. Name of the property. The value of this parameter maps to the name attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.
value	**String**. Value of the property. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.

property **Document List**. Optional. List of name-value pairs that are sent along with the message.

These pairs map to the zero or more Property child elements within Messaging/UserMessage/MessageProperties.

Parameter	Description
name	**String**. Name of property.
value	**String**. Value of property.

partContent **InputStream**. Optional. Array of all the MIME attachments. Each element within the array must have a corresponding *partInfo*.

Note:
If *partInfo* is not specified for *partContent*, then SOAP BODY is considered as the default *partInfo* type.

Output Parameters

The output of the service will be an input for existing wm.ip.estd.as4.msh:submit service.

wm.ip.estd.as4.msh:sendPull

Enables the receiver to send a pull signal to the responding MSH. The pulled user message is returned on the back channel of the pull request.

Input Parameters

messageId **String**. Optional. Unique identifier of the pull message.

mpc **String**. MPC from which to pull the queued message.

pmodeId **String**. TPA agreement ID to use for the pull request.

soapAction **String**. Optional. Value of the *action* parameter in the MIME message that was sent.

simpleSelectionItems **Document**. Optional. Select a list of simple elements.

Parameter	Description
element	**String** Name of the element that has to be retrieved. This parameter refers to *RefToMessageId*, *ConversationId*, *AgreementRef*, *Service*, *Action*
elementvalue	**String**. Value of the element.
attributeList	**Document** Select a list of attributes. The variables are: ■ *element*: **String**. Name of the attribute. ■ *elementValue*: **String**. Value of the element

complexSelectionItems **Document** Optional. Select a list of complex elements.

Parameter	Description
element	**String**. Name of the element that has to be retrieved. This parameter refers to *From*, *To*,*MessageProperties*.
elementValue	**String**. Value of the element
attributeList	**Document**. A list of attributes. The variables are: ■ *element*: **String**. Name of the attribute ■ *elementValue*: **String**. Value of the attribute

childItems	**Document**. A list of child items.
	■ *element*: **String**. Name of the child item.
	■ *elementvalue*: **String**. Value of the child item.
	■ *attributeList*
	  ■ *element*: **String**. Name of the attribute.
	  ■ *elementvalue*: **String**. Value of the attribute.

Output Parameters

error **Document**. Specifies an error message while processing a document.

Parameter	Description
code	**String**. ebms error code, if any.
description	**String**. Description of the error.

wm.ip.estd.as4.peppol:getReceiverDetails

Retrieves the endpoint address and recipient certificate of the recipient Peppol participant. This service retrieves the details of both production and test participants.

Note:
When you enable as4.peppol.proxy.enable property and do not configure the default HTTP proxy server in Integration server, you must set the HTTP proxy alias server for the as4.peppol.proxyAlias property in the configuration file located here: <Install_Dir>\IntegrationServer\instances<Instance_Name>\packages\WmAS4\config\config.cnf.

Input Parameters

participantID **String**. Peppol participant identifier. For example, `9915:test`.

isProduction **String**. Retrives the details of the participants. Valid values are:

- `true`. Retrieves the details from production environment.
- `false`. Retrieves the details from test environment.

documentType **Document List**. Document to be exchanged. The list of profile names of various document types. If any of the document type is not available, then provide the corresponding Peppol document type identifier value.

processIdentifier **Document List**. List of the Peppol process identifier value. If any of the process identifier is not available, then provide the input as `{Peppol Identifier Scheme}::{Peppol Identifier Value}`.

For example, if `cenbii-procid-ubl` is the Peppol Identifier Scheme and `urn:`www.cenbii.eu:profile:bii01:ver1.0 is the Peppol Identifier Value, then the processIdentifier value must be `cenbii-procid-ubl::urn:`www.cenbii.eu:profile:bii01:ver1.0.

Output Parameters

endpointUrl **String**. The endpoint recipient URL.

certificate	**String**. Holds the complete signing X509 certificate of the recipient access point. A byte array that represents the signing X509 certificate (a java.security.cert.X509Certificate).
description	**String**. A human-readable description of the endpoint.
activationDate	**String**. The activation date of the endpoint. A missing activation date must be interpreted as `valid since forever`.
expirationDate	**String**. The expiration date of the endpoint. A missing expiration date must be interpreted as `valid until eternity`.
contactUrl	**String**. Represents a link to human readable contact information. It can be an email address.
informationUrl	**String**. A URL to human-readable documentation. For example, a website contains XML Schemas, WSDLs, Schematrons, and other relevant resource links.

wm.ip.estd.as4.peppol:addPartnerOrUpdateCertificate

Creates the recipient partner profile in Trading Networks with the certificate of the recipient. Partner name is created with corporation as the CNAME of the certificate and the organisation unit as Peppol participant identifier. Also, the externalID value is the format CNAME(ParticipantID) and configured under IDType `urn:FDC:peppol.eu:2017:identifiers:ap` in the corresponding partner profile. For example: `POP000143(0151:79000024733)`. The certificate is configured for encrypt/decrypt and sign/verify in the corresponding profile. If the partner already exists in Trading Networks, the certificate is updated in the corresponding Trading Networks partner profile by default.

The caCertificateChain input field is optional. Depending on the participant certificate passed as input, the corresponding root, and intermediate certificates provided by Peppol is used by default as it varies based on production or test environment.

Input Parameters

participantID	**String**. Peppol participant identifier. For example 9915:test
certificate	**String**. A byte array that represents the signing X509 certificate (a java.security.cert.X509Certificate) of the recipient access point.
caCertificateChain	**String**. Optional. A byte array that represents the X509 certificate (a java.security.cert.X509Certificate) of the issuer of the certificate.
forceUpdateCertificate	**Boolean**. Optional. Indicates whether to update the certificate in the Trading Networks partner profile. Default is true.

Output Parameters

toPartyId	**String**. PartyId of the recipient partner.
partnerId	**String**. InternalID of the recipient Trading Networks partner profile.
errors	**String**. Specifies an error message while processing a document.

wm.ip.estd.as4.peppol:constructSubmitInput

Constructs the input for the wm.ip.estd.as4.msh:submit service in order to send the business document. This service is used when submitting the message to Peppol network.

Input Parameters

refToMessageId	**String**. Identifier that refers to the *messageId* of the request message.
messageId	**String**. Optional. Unique identifier of the user message. If no value is specified, Module for AS4 generates a unique *messageId*. When the message is generated, the value of this parameter maps to Messaging/UserMessage/MessageInfo/MessageId.
fromPartyId	**String**. External ID of the party sending the message, as defined in the trading partner profile.
toPartyId	**String**. PartyId of the recipient partner. This can be retrieved from the wm.ip.estd.as4.peppol:addPartner service.
conversationId	**String**. Optional. Identifier of the set of related messages that make up a conversation between two parties. If no value is specified, Module for AS4 generates a unique ID. The value of this property maps to Messaging/UserMessage/CollaborationInfo/ConversationId.
pmodeId	**String**. Optional. TPA agreement ID to use for the submit request.
processIdentifier	**Document List** . List of the Peppol process identifier value. Specify the required Peppol document type identifier value if the identifier value is not in the dropdown list. For example, { Peppol Identifier Scheme}::{Peppol Identifier Value}.
documentType	**Document List** . List of document type to be exchanged. Specify the required Peppol document type identifier value if the document type is not present in the dropdown list.
endpointUrl	**String**. Endpoint address of the recipient. It retrieves from the wm.ip.estd.as4.peppol:getReceiverDetails service.

property **Document List** . Optional. List of name-value pairs that are sent along with the message.

These pairs map to the zero or more property child elements within

Messaging/UserMessage/MessageProperties.

Parameter	Description
name	**String**. Name of property.
value	**String**. Value of property.
type	**String**. Type of property.

partInfo **Document List**. Metadata for the *partContent* parameter.

The values specified map to the Messaging/UserMessage/PayloadInfo/PartInfo element.

Parameter	Description
type	**String**. Way in which the *partContent* needs to be packaged. Valid values are: ■ MIME PART—*partContent* is packaged as MIME attachment. ■ SOAP BODY—*partContent* is packaged in the SOAP body. ■ EXTERNAL REFERENCE—*partContent* is an external reference.
id	**String**. ID of the *partContent*.
schemaLocation	**String**. URI of the schema. The value of this parameter maps to the *location* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaVersion	**String**. Version identifier of the schema. The value of this parameter maps to the *version* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaNamespace	**String**. Target namespace of the schema. The value of this parameter maps to the *namespace* attribute of the

Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.

description **String**. Description of the *partContent*.

The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/Description element.

property **Document List**. Optional. List of name value pairs that are sent along with the message.

The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties element.

Note:
If you want the final AS4 payload of MIME type application/octet-stream, then add the property *name* to be of MIME type and *value* as application/octet-stream.

Parameter	Description
name	**String**. Name of the property. The value of this parameter maps to the name attribute of the Messaging/ UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.
value	**String**. Value of the property. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.

partContent **InputStream**. Optional. Array of all the MIME attachments. Each element within the array must have a corresponding *partInfo*.

Note:
If *partInfo* is not specified for *partContent*, then SOAP BODY is considered as the default *partInfo* type.

soapAction **String**. Optional. Value of the action parameter in the MIME type.

Output Parameters

refToMessageId	**String**. Optional. Identifier used in Two-Way MEPs where the responding MSH user message refers to the initiating MSH user message.
messageId	**String**. Optional. Unique identifier of the user message. If no value is specified, Module for AS4 generates a unique *messageId*. When the message is generated, the value of this parameter maps to Messaging/UserMessage/MessageInfo/MessageId.
toPartyId	**String**. External ID of the party receiving the message, as defined in the trading partner profile.
toPartyIdType	**String**. External ID type of the party receiving the message, as defined in the trading partner profile.
toPartyIdRole	**String**. Optional. The initiator or responder role of the party in the message exchange.
fromPartyId	**String**. External ID of the party sending the message, as defined in the trading partner profile.
fromPartyIdType	**String**. External ID type of the party sending the message, as defined in the trading partner profile.
fromPartyIdRole	**String**. Optional. The initiator or responder role of the party in the message exchange.
conversationId	**String**. Optional. Identifier of the set of related messages that make up a conversation between two parties. If no value is specified, Module for AS4 generates a unique ID. The value of this property maps to Messaging/UserMessage/CollaborationInfo/ConversationId.
pmodeId	**String**. Optional. TPA agreement ID to use for the submit request. **Note:** If *pmodeId* is empty, then a tuple of From/PartyId, From/Role, To/PartyId, To/Role, Service, Action, and AgreementRef is used to identify the TPA agreement.
agreementRef	**String**. Optional. AgreementRef value to use for the submit request.
property	**Document List**. Optional. List of name-value pairs that are sent along with the message. These pairs map to the zero or more Property child elements within Messaging/UserMessage/MessageProperties.

Parameter	Description

name	**String**. Name of property.
value	**String**. Value of property.

partInfo **Document List**. Optional. Metadata for the *partContent* parameter.

The values specified map to the Messaging/UserMessage/PayloadInfo/PartInfo element.

Parameter	Description
type	**String**. Way in which the *partContent* needs to be packaged. Valid values are: ■ MIME PART—*partContent* is packaged as MIME attachment. ■ SOAP BODY—*partContent* is packaged in the SOAP body. ■ EXTERNAL REFERENCE—*partContent* is an external reference.
id	**String**. ID of the *partContent*.
schemaLocation	**String**. URI of the schema. The value of this parameter maps to the *location* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaVersion	**String**. Version identifier of the schema. The value of this parameter maps to the *version* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
schemaNamespace	**String**. Optional. Target namespace of the schema. The value of this parameter maps to the *namespace* attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/Schema element.
description	**String**. Description of the *partContent*. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/Description element.
property	**Document List**. List of name value pairs that are sent along with the message. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties element. **Note:**

If you want the final AS4 payload of MIME type application/octet-stream, then add the property *name* to be of MIME type and *value* as application/octet-stream.

Parameter	Description
name	**String**. Name of the property. The value of this parameter maps to the name attribute of the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.
value	**String**. Value of the property. The value of this parameter maps to the Messaging/UserMessage/PayloadInfo/PartInfo/PartProperties/Property element.

partContent **InputStream**. Array of all the MIME attachments. Each element within the array must have a corresponding *partInfo*.

Note:
If *partInfo* is not specified for *partContent*, then SOAP BODY is considered as the default *partInfo* type.

soapAction **String**. Optional. Value of the *action* parameter in the MIME type.

service **String**. Optional. Name of service used to identify the RequestUM leg to be used for the current transaction.

The module uses the *service* and *action* parameters to determine which RequestUM leg to use for the current transaction. The module compares the value of this parameter to the value of *leg/businessInfo/service* that is configured in all RequestUM legs of the TPA. The RequestUM leg that contains a match for both the *service* and *action* parameters is the leg that is used for the current transaction.

When there is only one RequestUM leg configured in the TPA, the *service* parameter does not have to be defined.

action **String**. Optional. Name of action used to identify the RequestUM leg to be used for the current transaction.

The module uses the *service* and *action* parameters to determine which RequestUM leg to use for the current transaction. The module compares the value of this parameter to the value of *leg/businessInfo/action* that is configured in all RequestUM legs of the TPA. The RequestUM leg that contains a match for both *service* and *action* is the leg that is used for the current transaction.

	When there is only one RequestUM leg configured in the TPA, the *action* parameter does not have to be defined.
endpointUrl	**String**. Endpoint address of the recipient partner. It retrieves from the wm.ip.estd.as4.peppol:getReceiverDetails service.

wm.ip.estd.as4.peppol:validateContent

The payload is validated if it is Invoice or CreditNote for both inbound and outbound processing of documents. If the validation has to be done for any other documents, you must configure the XML schema and the schematron in the file: `<Install_Dir>\IntegrationServer\instances\default\packages\WmAS4\resources\peppol\validate.cnf` with key as the corresponding document identifier and value as the comma separated values of schema names and the schematron names. XML schema must be located at `<Install_Dir>\IntegrationServer\instances\default\packages\WmAS4\resources\peppol\xsd\maindoc` and the schematron files must be located at `<Install_Dir>\IntegrationServer\instances\default\packages\WmAS4\resources\peppol` location.

Validates Peppol billing BIS 3.0 invoices and credit notes.Validates in the following order:

1. The UBL 2.1 XML Schema from OASIS.
2. The CEN rules from https://docs.peppol.eu/poacc/billing/3.0/files/CEN-EN16931-UBL.sch.
3. The Peppol rules from https://docs.peppol.eu/poacc/billing/3.0/files/PEPPOL-EN16931-UBL.sch. For the latest rules, see https://docs.peppol.eu/poacc/billing/3.0/.

Input Parameters

action	**String**. Peppol document type identifier.
content	**InputStream**. A byte array that represents the invoice or creditNote payload.

Output Parameters

errors	**String**. Any error that was encountered during the generation of the message.

wm.ip.estd.as4.util.pull:clearMPC

For the MPC specified, removes all bizDocIDs that are cached in Module for AS4. When the transaction analysis in Trading Networks is purged, you must invoke this service to clear the cache as well.

Input Parameters

mpcName	**String**. Name of the MPC from which to clear the content.
bizDocIDs	**String** . Optional. List of bizDocIDs to be cleared. If this parameter is null, all the contents of the MPC are cleared.

Output Parameters

None.

wm.ip.estd.as4.util.pull:getMPCContents

Returns the list of bizDocIDs queued in the specified MPC.

Input Parameters

mpcName	**String**. Name of the MPC from which to retrieve contents.

Output Parameters

results	**String []**. List of bizDocIDs queued in the specified MPC.

wm.ip.estd.as4.util.multihop:routingFunction

Returns the information needed to forward a received message in a multi-hop exchange. This is the default routing service that the intermediary MSH (Module for AS4) invokes to determine which MSH the received message should be forwarded to next.

This service uses the ebMS PartyInfo element and its sub-elements to determine the forwarding address. You can customize this service by editing the mappings in the WmAS4\config\RoutingTable.properties.

Input Parameters

routingInput	**Document**. A document reference to the com.wm.estd.as4.documents.multihop.routingFunction:RoutingInput document type, which specifies the subset of the header content of the ebMS user message, specifically, the eb3:UserMessage element.

Output Parameters

result	**Document**. A reference to com.wm.estd.as4.documents.multihop.routingFunction:NextDestination, which specifies the destination details for the message to be forwarded.

wm.ip.estd.as4.util.retry:cancelRetry

Cancels the Retry that is scheduled for a given user message.

Usually, when Retry is enabled and either the user message fails to be sent or no receipt is received for the user message, the module tries to resend the user message after a configured time interval. The cancelRetry service cancels retrying to send the user message.

Input Parameters

msgId	**String**. The message ID of the user message whose retry needs to be cancelled. If the user message is split into fragments, specifying the message ID of the user message will cancel all retries of all the fragments of the user message.

Output Parameters

taskCancelled	**String**. The status of the cancellation. When *taskCancelled* is true, retry was cancelled successfully. When *taskCancelled* is false, retry was not cancelled successfully.

wm.ip.estd.as4.util.split:getGroupList

Retrieves a list of group IDs of the message fragments that are in a given state.

When message splitting is enabled, each fragment group at the receiver side has a status of either ACTIVE, COMPLETE, EXPIRED, or REJECTED. The getGroupList service retrieves a list of the group IDs of those fragments that are in the specified status.

Input Parameter

status	**String**. Optional. The status of the fragments you want to be listed. Valid values are: ■ `ACTIVE`—One or more fragments of the group was received and there are still more fragments to come. ■ `COMPLETE`—All fragments in the group were received.

- EXPIRED—A group fragment was received after the time set for the *joinInterval* parameter elapsed.
- REJECTED—There was a problem during processing of one of the fragments in the group.

Output Parameters

groupIds **String[]**. Optional. The list of all the group IDs that have the specified status.

wm.ip.estd.as4.util.split:clearGroup

Resets the status of a group.

Input Parameters

status **String**. Optional. Specifies the status of the groups whose status you want reset. Valid values are:

- ACTIVE—One or more fragments of the group was received and there are still more fragments to come.
- COMPLETE—All fragments in the group were received.
- EXPIRED—A group fragment was received after the time set for the *joinInterval* parameter elapsed.
- REJECTED—There was a problem during processing of one of the fragments in the group.

groupIds **String**. Optional. Specifies the array of group IDs that have to be reset. When a value is specified, the value of the *status* parameter is ignored.

Output Parameters

None.

Summary of Elements for the WmAS4AU Package

wm.estd.as4.au.handlers:process

The following services are provided with the WmAS4AU package:

Element	Description
"wm.estd.as4.au.handlers:process" on page 111	Processes the incoming agreement update requests.
"wm.estd.as4.au.handlers:validate" on page 113	Validates the inbound and outbound agreement update messages.
"wm.estd.as4.au.message:generateTerminationException" on page 114	Generates the AgreementTerminationException payload in xml string format.
"wm.estd.as4.au.message:generateTerminationRequest" on page 114	Generates the AgreementTerminationRequest payload in xml string format.
"wm.estd.as4.au.message:generateTerminationResponse" on page 115	Generates the AgreementTerminationResponse payload in xml string format.
"wm.estd.as4.au.message:generateUpdateException" on page 115	Generates the AgreementUpdateException payload in xml string format.
"wm.estd.as4.au.message:generateUpdateRequest" on page 115	Generates the AgreementUpdateRequest payload in xml string format.
"wm.estd.as4.au.message:generateUpdateResponse" on page 116	Generates the AgreementUpdateResponse payload in xml string format.

Processes the incoming agreement update requests and executes the following steps:

1. Verifies the certificate update request, if present, as substitute for abstract UpdateRequest element and adds the new certificates into the initiator trading network partner profile.
2. Creates a new agreement with the new requested values by copying the existing agreements between the sender party and the receiver party who has a match with the current agreement identifier passed in the request and adds the new Certificate ID to the signing or encryption configuration in the newly created TPA.

Input Parameters

bizdoc	**Document Reference** bizdoc (wm.tn.rec:BizDocEnvelope) for which an agreement update request is initiated.

Output Parameters

tpaList	**Document List**. List of TPAs that are newly generated as per the UpdateAgreementIdentifier for all the identified CurrentAgreementIdentifier.
auError	**Document Reference List**. Optional. List of errors that occurred during the processing of the request. (wm.estd.as4.au.rec:AuError)

error **Document**. Optional. Any error that was encountered while processing the request.

Parameter	Description
description	**String**. Description of the error.

wm.estd.as4.au.handlers:validate

Validates the message types in agreement update messages.

Input Parameters

isInbound **String**. Optional. Whether a message is inbound or outbound. The default is false.

bizdoc **Document Reference**. Optional. BizDocEnvelope (wm.tn.rec:BizDocEnvelope) associated with the most recent document update.

message **String**. Optional. XML payload associated with the document.

fromParty **String**. Optional. Internal ID of the party as defined in the trading partner profile that sends the message.

toParty **String**. Optional. Internal ID of the party as defined in the trading partner profile that receives the message.

pmodeId **String**. Optional. TPA agreement ID to use for this message.

Note:

- Either the bizdoc or message is required.
- If the message string is passed, then *fromParty*, *toParty*, and *pmodeId* is mandatory.

Output Parameters

isValid **String**. Whether the inbound or outbound AgreementUpdate message is valid or not.

auError **Document Reference List**. Optional. List of errors that occurred during the validation of the message.

error **Document**. Optional. Any error that was encountered while validating the request.

Parameter	Description
description	**String**. Description of the error.

wm.estd.as4.au.message:generateTerminationException

Generates the AgreementTerminationException payload in xml string format.

Input Parameters

AgreementTerminationException **Document Reference**. Document reference object that specifies the required values for AgreementTerminationException message.

Note:
For information about required fields for document inputs and datatypes, see *OASIS ebCore Agreement Update Specification Version 1.0* and *ebcore-au-v1.0.xsd* schema.

Output Parameters

message **String**. Optional. Payload in xml string format.

error **String**. Optional. Any error that was encountered during the generation of the message.

Parameter	Description
description	**String**. Description of the error.

wm.estd.as4.au.message:generateTerminationRequest

Generates the AgreementTerminationRequest payload in xml string format.

Input Parameters

AgreementTerminationRequest **Document Reference**. Document reference object that specifies the required values for AgreementTerminationRequest message.

Note:
For information about required fields for document inputs and datatypes, see *OASIS ebCore Agreement Update Specification Version 1.0* and *ebcore-au-v1.0.xsd* schema.

Output Parameters

message **String**. Optional. Payload in xml string format.

error **String**. Optional. Any error that was encountered during the generation of the message.

wm.estd.as4.au.message:generateTerminationResponse

Generates the AgreementTerminationResponse payload in xml string format.

Input Parameters

AgreementTerminationResponse **Document Reference**. Document reference object that specifies the required values for AgreementTerminationResponse message.

Note:
For information about required fields for document inputs and datatypes, see *OASIS ebCore Agreement Update Specification Version 1.0* and *ebcore-au-v1.0.xsd* schema.

Output Parameters

message **String**. Optional. Payload in xml string format.

error **String**. Optional. Any error that was encountered during the generation of the message.

wm.estd.as4.au.message:generateUpdateException

Generates the AgreementUpdateException payload in xml string format.

Input Parameters

AgreementUpdateException **Document Reference**. Document reference object that specifies the required values for AgreementUpdateException message.

Note:
For information about required fields for document inputs and datatypes, see *OASIS ebCore Agreement Update Specification Version 1.0* and *ebcore-au-v1.0.xsd* schema.

Output Parameters

message **String**. Optional. Payload in xml string format.

error **String**. Optional. Any error that was encountered during the generation of the message.

wm.estd.as4.au.message:generateUpdateRequest

Generates the AgreementUpdateRequest payload in xml string format.

Input Parameters

AgreementUpdateRequest **Document Reference**. Document reference object that specifies the required values for AgreementUpdateRequest message.

Note:
For information about required fields for document inputs and datatypes, see *OASIS ebCore Agreement Update Specification Version 1.0* and *ebcore-au-v1.0.xsd* schema.

Output Parameters

message **String**. Optional. Payload in xml string format.

error **String**. Optional. Any error that was encountered during the generation of the message.

wm.estd.as4.au.message:generateUpdateResponse

Generates the AgreementUpdateResponse payload in xml string format.

Input Parameters

AgreementUpdateResponse **Document Reference**. Document reference object that specifies the required values for AgreementUpdateResponse message.

Note:
For information about required fields for document inputs and datatypes, see *OASIS ebCore Agreement Update Specification Version 1.0* and *ebcore-au-v1.0.xsd* schema.

Output Parameters

message **String**. Optional. Payload in xml string format.

error **String**. Optional. Any error that was encountered during the generation of the message.

B Supported P-Mode Parameters

Overview

This appendix names the P-Mode parameters that Module for AS4 supports.

Supported P-Mode Parameters

Module for AS4 supports the following P-Mode parameters:

- PMode.ID
- PMode.Agreement
- PMode.MEP
- PMode.MEPbinding
- PMode.Initiator.Party
- PMode.Initiator.Role
- PMode.Initiator.Authorization.username
- PMode.Initiator.Authorization.password
- PMode.Responder.Party
- PMode.Responder.Role
- PMode.Responder.Authorization.username
- PMode.Responder.Authorization.password
- PMode.Protocol.Address
- PMode.BusinessInfo.Service
- PMode.BusinessInfo.Action
- PMode.BusinessInfo.Properties
- PMode.BusinessInfo.PayloadProfile
- PMode.BusinessInfo.PayloadProfile.maxSize
- PMode.ErrorHandling.Report.SenderErrorsTo
- PMode.ErrorHandling.Report.ReceiverErrorsTo
- PMode.ErrorHandling.Report.AsResponse
- PMode.ErrorHandling.Report.ProcessErrorNotifyConsumer
- PMode.ErrorHandling.Report.ProcessErrorNotifyProducer
- PMode.ErrorHandling.Report.DeliveryFailuresNotifyProducer

PMode.ErrorHandling.Report.MissingReceiptNotifyProducer

PMode.Security.Include.TimeStamp

PMode.Security.X509.Sign

PMode.Security.X509.Signature.Certificate

PMode.Security.X509.Signature.HashFunction

PMode.Security.X509.Signature.Algorithm

PMode.Security.X509.Signature.SignReceiptBody

PMode.Security.X509.Encryption.Encrypt

PMode.Security.X509.Encryption.Certificate

PMode.Security.X509.Encryption.Algorithm

PMode.Security.UsernameToken.username

PMode.Security.UsernameToken.password

PMode.Security.PModeAuthorize

PMode.Security.SendReceipt

PMode.Security.SendReceipt.ReplyPattern

PMode.ReceptionAwareness

PMode.ReceptionAwareness.Retry

PMode.ReceptionAwareness.Retry.Parameters

PMode.ReceptionAwareness.DuplicateDetection

PMode.PayloadService.Compression

PMode.Splitting.FragmentSize

PMode.Splitting.JoinInterval

PMode.Protocol.AddActorOrRoleAttribute