EUROPEAN COMMISSION

DIGIT
Digital Europe Programme

Domibus

Interface Control Document

Default JMS Plugin

Version [1.9]

Status [Final]

Date: 03/06/2022

Document Approver(s):

Approver Name	Role
Joao RODRIGUES	Program Manager
Adrien FERIAL	Project Manager

Document Reviewers:

Reviewer Name	Role
Daniels MAARTEN	Tester
Ioana DRAGUSANU	Developer
Yves ADAM	Quality Manager
Cosmin BACIU	Technical Leader
Tiago MIGUEL	Developer
Caroline AEBY and Chaouki BERRAH	Technical Writers

Summary of Changes:

Version	Date	Created by	Short Description of Changes
0.01	18/03/2016	Pedro TAVARES	Initial version
0.02	31/03/2016	Pedro TAVARES	Update with comments of eCodex + Update section 4.1 with simple function for submitting a message
0.03	06/04/2016	Pedro TAVARES	Update section 4.1 with simple function for submitting a message
1.00	15/05/2016	Pedro TAVARES	Update document with the new format
1.01	08/08/2016	Pedro TAVARES	Update document with Yves ADAM comments
1.02	31/08/2016	Yves ADAM	Upgrade from version 3.1.1 to 3.2
1.03	22/03/2017	Cosmin BACIU	Upgrade from version 3.2 to 3.2.3
1.04	10/04/2017	Cosmin BACIU	Use a Domibus generic version instead of a specific version
1.05	29/06/2017	Cosmin BACIU	Updated the JMS properties to use _ instead of -
1.06	07/09/2017	Tiago MIGUEL	Updated the JMS plugin properties to expose in the OUT queue the message payload's file location. Reviewed document contents and formatting
1.07	09/10/2017	CEF Support	List of reviewers updated.
1.08	04/12/2017	CEF Support	References updated for Domibus 3.3.1 release.
1.09	08/03/2018	CEF Support	References to Domibus 3.3.2 & links

			updated
1.10	20/03/2018	CEF Support	Reuse notice added; e-Sens AS4 profile replaced by eDelivery AS4 profile.
1.20	16/07/2018	Chaouki BERRAH	"payload_[NUM]_description: A description of the payload" removed
1.21	25/07/2018	Cosmin BACIU Ioana DRAGUSANU	Multi-tenancy JMS plugin configuration
1.22	30/08/2018	Cosmin BACIU	Multi-tenancy updated
1.23	07/09/2018	Cosmin BACIU	Additional information for multi-tenancy
1.24	17/09/2018	Caroline AEBY	multi-tenancy => Multitenancy
1.25	26/09/2018	Caroline AEBY	End of the standby service
1.26	25/05/2019	Ioana DRAGUSANU Caroline AEBY	Updates for Domibus 4.1-RC1
1.27	24/03/2020	Caroline AEBY Chaouki BERRAH	FInalrecipientType for Dynamic Discovery usage added (from Domibus 4.1.4)
1.28	16/04/2020	Cosmin BACIU	Added 3.3.6 for Domibus 4.2
1.29	21/09/2020	Caroline AEBY	Domibus 4.2 RC release
1.30	14/10/2020	Caroline AEBY Arun VENUGOPAL	Domibus 4.2. FR – changes in mandatory metadata fields on message send by C1
1.3.1	15/10/2020	Caroline AEBY Arun VENUGOPAL	Updates related to payload handling
1.3.2	22/10/2020	Caroline AEBY Arun VENUGOPAL	PartyIdType optional both for DynamicDiscovery & normal use
1.3.3	30/11/2020	Caroline AEBY Cosmin BACIU	Domibus 4.2 review
1.4	17/12/2020	Caroline AEBY	Domibus 4.2 final version
1.5	22/06/2021	Cosmin BACIU Caroline AEBY	Receive notifications
1.6	31/01/2022	Caroline AEBY	Message value corrected
1.7	15/03/2022	Cosmin BACIU Caroline AEBY	References updates + Referencing Payloads chapter added
1.8	29/04/2022	Caroline AEBY	No more CEF references + links updated
1.9	03/06/2022	Caroline AEBY	eDelivery support FMB udpated

Table of Contents

1. INTRODUCTION

1.1. Purpose

The purpose of this document is to outline the JMS Data Format Exchange to be used as part of the default JMS backend integration solution for the Domibus Access Point[1].
According to eDelivery, an Access Point is an implementation of the OpenPEPPOL AS2 Profile or the eDelivery AS4 Profile. The data exchange protocols of eDelivery are profiles, meaning that several options of the original technical specifications were narrowed down in order to increase consistency, interoperability and to simplify deployment. The profile of AS2 was developed by OpenPEPPOL[2], and the profile of AS4 was developed by e-SENS[3] in collaboration with several service providers while being implemented in the e-Justice domain by e-CODEX. An Access Point exposes two interfaces:

- An interface to connect the Backend system with the Access Point. Typically, this interface is customisable as communication between Access Points and Backend systems may use any messaging or transport protocol.
- A standard messaging interface between Access Points, this interface is configurable according to the options of the profiles supported by eDelivery. It is important to note that eDelivery standardises the communication only between the Access Points.

This document will univocally define the JMS plugin that acts as an interface to the Access Point (Corner Two and Corner Three in the four corner topology that will be explained later in this document) component of the eDelivery building block.

There is 1 interface described in this document:

Interface	Description	Version
JMS backend integration	The JMS plugin	4.x.y

Table 1 - Interface described

1.2. Scope

This document covers the service interface of the Access Point from the perspective of the JMS backend integration. It includes information regarding the description of the JMS-Queues, information model and the types of messages for the services provided. This specification addresses no more than the service interface of the Access Point. All other aspects of its implementation are not covered by this document (i.e. the service consumer). The ICD specification provides both the provider (i.e. the implementer) of the services and their consumers with a complete specification of the following aspects:

[1] https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/Domibus

[2] http://www.peppol.eu/

[3] http://www.esens.eu/

- Interface Functional Specification, this specifies the set of services and the operations provided by each service;
- Interface Behavioural Specification, this specifies the expected sequence of steps to be respected when calling a service or a set of services;
- Interface Message standards, this specifies the syntax and semantics of the data and metadata;
- Interface Policy Specification, this specifies constraints and policies regarding the operation of the service.

1.3. Audience

This document is intended to the Directorate Generals and Services of the European Commission, Member States (MS) and also companies of the private sector wanting to set up a connection between their backend systems and the Access Point.

In particular:

- Architects will find it useful for determining how to best exploit the Access Point to create a fully-fledged solution and as a starting point for connecting a Back-Office system to the Access Point.
- Analysts will find it useful to understand the Access Point that will enable them to have a holistic and detailed view of the operations and data involved in the use cases.
- Developers will find it essential as a basis of their development concerning the Access Point plugin services.
- Testers can use this document in order to test the interface by following the use cases described.

1.4. References

The table below provides the reader with the list of reference documents.

#	Document	Contents outline
[REF1]	Access Point Component Description Document	Overview of eDelivery Access Point
[REF2]	Using HTTP Methods for RESTful Services	Short description of HTTP Methods for RESTful Services
[REF3]	Business Document Metadata Service Location - Software Architecture Document	This document is the Software Architecture document of the CIPA eDelivery Business Document Metadata Service Location application (BDMSL) sample implementation. It intends to provide detailed information about the project: 1) An overview of the solution 2) The different layers 3) The principles governing its software architecture.
[REF4]	ebXML (Electronic Business using eXtensible Markup Language)	ebXML (Electronic Business using eXtensible Markup Language)
[REF5]	Web Services Description Language (WSDL) 1.1	Web Services Description Language (WSDL) 1.1 WS-I Basic Profile Version 1.1
[REF6]	XML Schema 1.1	XML Schema 1.1
[REF7]	Extensible Markup Language (XML) 1.0	Extensible Markup Language (XML) 1.0

#	Document	Contents outline
[REF8]	Hypertext Transfer Protocol 1.1	Hypertext Transfer Protocol 1.1
[REF9]	SOAP Messages with Attachments	SOAP Messages with Attachments
[REF10]	AS4 Profile of ebMS 3.0 Version 1.0	AS4 Profile of ebMS 3.0 Version 1.0
[REF11]	eDelivery - profile	https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eDelivery+AS4
[REF12]	eDelivery – Pmode Configuration	eDelivery – Pmode Configuration *(will be available at a later stage)*
[REF13]	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/	XSDs for ebms3
[REF14]	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cs02/ebms_core-3.0-spec-cs-02.pdf	ebXML (Electronic Business using eXtensible Markup Language)

1.5. Acronyms

Acronym	Definition
ebMS	ebXML Messaging Service Specification
MEP	Message Exchange Pattern A Message Exchange Pattern describes the pattern of messages required by a communications protocol to establish or use a communication channel.
ebXML	Electronic Business XML Project to use XML to standardise the secure exchange of business data.
P-Mode	Processing Mode
MSH	Message Service Handler The MSH is an entity that is able to generate or process messages that conform to the ebMS specification, and which act in at least one of the two ebMS roles: Sender and Receiver. In terms of SOAP processing, an MSH is either a SOAP processor or a chain of SOAP processors. In either case, an MSH has to be able to understand the eb:Messaging header (qualified with the ebMS namespace).

2. INTERFACE FUNCTIONAL SPECIFICATION

2.1. The four corner model

In order to understand the Use Cases that will be described below it is important to explain the topology; i.e. the four – corner model.

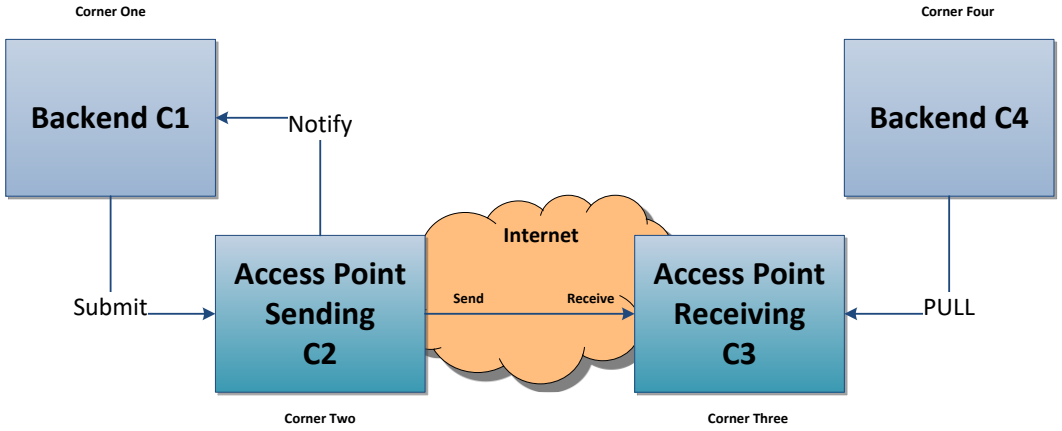

Figure 1 - The four corner model

In this model we have the following elements:

- Corner One (C1): Backend C1 is the system that will send messages to the sending AP (Access Point)
- Corner Two (C2): Sending Access Point C2
- Corner Three (C3): Receiving Access Point C3
- Corner Four (C4): Backend C4 is the system that will receive messages from the receiving AP (Access Point)

The JMS backend is described in this document. JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication. JMS is also known as the standard for Java asynchronous messaging service. Messaging is a technique to enabling inter-application communications.

There are two types of messaging domains in JMS.

- Point-to-Point Messaging Domain
- Publisher/Subscriber Messaging Domain

The present JMS backend integration uses **Publisher/Subscriber Messaging pattern** where senders of messages, called publishers, do not plan the messages to be sent directly to specific receivers (called subscribers) but, instead, characterize published messages into classes without knowledge of which subscribers will be. Similarly, subscribers express interest in one or more classes and only receive the messages that are of their interest, without knowledge of which publishers are sending those messages. The intent of interest is done by means of a subscription.

2.2. Introduction to Domibus - AS4

Using as reference DEP DIGITAL[4], Domibus is the Open Source project of the AS4 Access Point maintained by the European Commission. Third-party software vendors offer alternative implementations of the eDelivery AS4 Profile (commercial or open-source). Each software vendor also provides different added-value services from integration to the support of day-to-day operations. For safeguarding interoperability, eDelivery encourages implementers to consult the list of software products that have passed the conformance tests by the European Commission of the eDelivery AS4 profile[5].

The sample software, Domibus, may be used to test other implementations of the AS4 profile or as a working solution in a production environment. The users of the sample implementation remain fully responsible for its integration with backend systems, deployment and operation. The support and maintenance of the sample implementation, as well as any other auxiliary services, are provided by the European Commission according to the terms defined in the eDelivery Access Point Component Offering Description.

It is also important to comment on the PMode. A processing mode – or PMode – is a collection of parameters that determine how user messages are exchanged between a pair of Access Points with respect to Quality of Service, Transmission Mode and Error Handling. A PMode maps the recipient Access Point from the partyId, which represents the backend offices associated to this Access Point.

[4] https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/Domibus

[5] https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/eDelivery+AS4+conformant+solutions

3. INTERFACE BEHAVIOURAL SPECIFICATION

A JMS queue is a staging area that contains messages that have been sent and are waiting to be read. Contrary to what the name queue suggests, messages don't have to be received in the order in which they were sent. A JMS queue only guarantees that each message is processed only once.

Domibus queues are classified in 3 types:

- ***Internal queues***: are accessed only by the core of the system
- ***Notification queues***: are populated by the core of the system in order to be retrieved by the plugins deployed on the local access points
- ***Backend queues***: are accessed by the backend themselves to either insert into or retrieve message from it.

Role of the plugins: plugins are the intermediate components that will allow incoming messages from corner 1 to enter corner 2 and outgoing messages to exit corner 3 to reach corner 4. These plugin must be compliant to Domibus specifications, and are specific to the backend implementation.

The following will introduce the queues chronologically, i.e. following the flow of message processed from corner 1 to corner 4.

The processing of a message, in short is processed as follows:

1. Corner 1 sends a message to an (input) plugin of corner 2.
2. The (input) plugin calls a set of API's exposed by the core to store the message into the database, generates a unique message ID and put that ID the internal dispatch queue referring to it.
3. The core of corner 2 discovers the message ID in the internal dispatch queue and the dispatcher sends it to the appropriate access point (corner 3).
4. The core of corner 3 stores it into the database, and creates a message into the internal notification queue referring to it
5. The notification listener of corner 3 discovers the message ID in the internal notification queue and makes it available into the notification dedicated queue of the appropriate (output) plugin of corner 3.
6. The (output) plug-in discovers the message ID into its dedicated queue and retrieves the message from the database
    - JMS (output) plugins will put it into the outQueue onto which its back-end (corner 4) is listening to.
    - Web service (output) plugin (Future implementation) will be send it directly to its back-end (retry will be done later in case of temporary unavailability of corner 4).

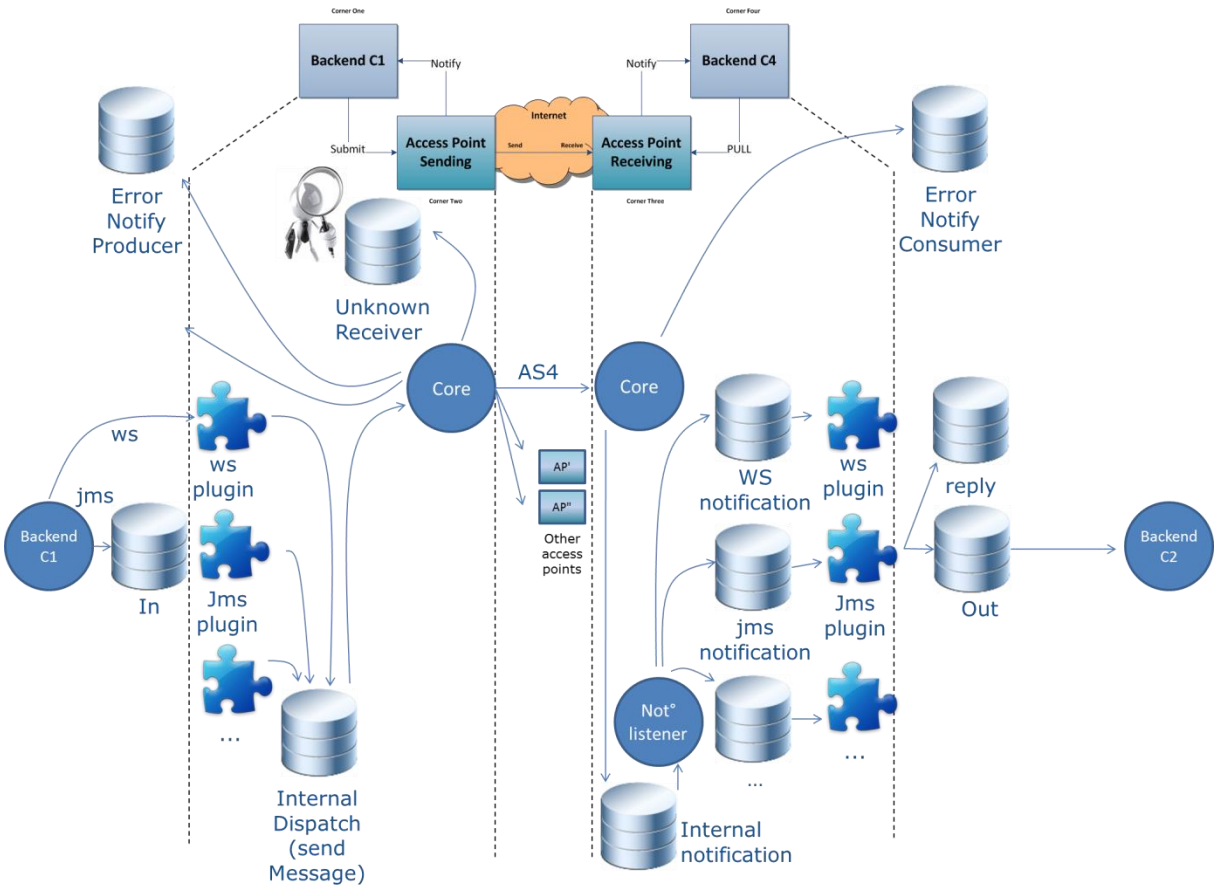

Figure 2 – Messages processing

The following section specifies the data format to be used to enable the following functions via JMS:

- Submit a message to the Access Point
- Push pending messages to a queue for retrieval

It uses the JMS MapMessage type in order to implement the request and response data formats for each of the functions mentioned above. The Meta data in each case will be set in the JMS message properties using name/value pairs and these will be outlined in each case.

3.1. JMS-Messages

Before going into the detail of the JMS queues it is important to describe the meaning of each tag included in the message that will be sent. It is Important to note that most values (parties, services, actions, etc…) are specified by the use case and multilateral agreements and thus not to be chosen by the caller when the message is submitted. They are underlined in the table below.

Name	Description
mpc	Message Partition Channels (MPCs) allow for partitioning the flow of messages from a Sending MSH to a Receiving MSH into several flows that can be controlled separately and consumed differently.
action	This element is a string identifying an operation or an activity within a Service. Its actual semantics is beyond the scope of this specification. Action SHALL be unique within the Service in which it is defined. The value of the Action element is specified by the designer of the service.
service	This element identifies the service that acts on the message. Its actual semantics is beyond the scope of this specification. The designer of the service may be a standards organization, or an individual or enterprise. In other words, service element denotes the service that processes the message at the destination. As an example of what might exist in the Service element, consider the text urn:Invoice, denoting a message that should be processed by the invoice service.
serviceType	The Service element MAY contain a single @type attribute, that indicates how the parties sending and receiving the message will interpret the value of the element. There is no restriction on the value of the type attribute. If the type attribute is not present, the content of the Service element MUST be a URI.
conversationId	The Party initiating a conversation determines the value of the ConversationId element that SHALL be reflected in all messages pertaining to that conversation. The actual semantics of this value is beyond the scope of this specification. Implementations SHOULD provide a facility for mapping between their identification scheme and a ConversationId generated by another implementation.
messageId	This element has a value representing – for each message - a globally unique identifier. Note: In the Message_Id and Content_Id MIME headers, values are always surrounded by angle brackets. However references in mid: or cid: scheme URI's and the MessageId and RefToMessageId elements MUST NOT include these delimiters.
refToMessageId	This element occurs at most once. When present, it MUST contain the MessageId value for which the message is related.
agreementRef	AgreementRef is a string value that identifies the agreement that governs the exchange. The value of an AgreementRef element MUST be unique within a namespace mutually agreed by the two parties. This could be a concatenation of the From and To PartyId's values, a URI containing the Internet domain name of one of the parties, or a namespace offered and managed by some other naming or registry service. It is RECOMMENDED that the AgreementRef is a URI.
agreementRefType	This attribute indicates how the parties sending and receiving the message will interpret the value of the reference. There is no restriction on the value of the type attribute. If the type attribute is not present, the content of the eb:AgreementRef element MUST be a URI.
fromRole	This element occurs once, and identifies the authorized role (fromAuthorizedRole) of the Party sending (present as a child of the From element) the message. The value of the fromRole element is a non- empty string, with a default value of http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultRole. Other possible values are subject to partner agreement.

Name	Description
toRole	This element occurs once, and identifies the authorized role (toAuthorizedRole) of the Party receiving (present as a child of the To element) the message. The value of the toRole element is a non- empty string, with a default value of http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultRole. Other possible values are subject to partner agreement.
messageType	A string representing the type of the message.
JMScorrelationId	The JMSCorrelationID header field is used for linking one message with another. It typically links a reply message with its requesting message. JMSCorrelationID can hold a provider-specific message ID, an application-specific String object, or a provider-native byte[] value.
fromPartyId	Access Point C2.This element has a string value content that identifies a party, or that is one of the identifiers of this party who is sending the message.
fromPartyType	A string that identifies the type of the sender partyId. The type attribute indicates the domain of names to which the string in the content of the fromPartyId element belongs. It is RECOMMENDED that the value of the type attribute be a URI. It is further RECOMMENDED that these values be taken from the EDIRA , EDIFACT or ANSI ASC X12 registries. Technical specifications for the first two registries can be found at and [ISO6523] and [ISO9735], respectively.
toPartyId	Access Point C3. This element has a string value content that identifies a partyId, or that is one of the identifiers of this party. The one who is receiving the message.
toPartyType	A string that identifies the type of the receiver partyId. The type attribute indicates the domain of names to which the string in the content of the toPartyId element belongs. It is RECOMMENDED that the value of the type attribute be a URI. It is further RECOMMENDED that these values be taken from the EDIRA , EDIFACT or ANSI ASC X12 registries. Technical specifications for the first two registries can be found at and [ISO6523] and [ISO9735], respectively.
originalSender	Backend C1. This element has a string value content that identifies the message producer. Who is created the message.
finalRecipient	Backend C4. This element has a string value content that identifies the message consumer. Who is the final receiver of the message.
finalRecipientType	Backend C4. This element has a string value content that could identify the message consumer, along with finalRecipient element. In Dynamic mode, this message property is also required to send JMS message. This field is mandatory for SMP to find the 'ToParty'. Default:value is "iso6523-actorid-upis"
protocol	The description of the protocol used. For the scenario described in this document it MUST be AS4.
totalNumberOfPayloads	Defines the number of payloads available in the message.
P1InBody (true/false)	Boolean that indicates if the payload is in the body of the AS4 message or not. If the payload is not in the body of the AS4 message it will be sent as attachment in the SOAP message.
putAttachmentInQueue	If true, all the payloads from the User Message will be stored as bytes in the JMS message. If false and Domibus is configured to save the payloads on the file system(property **domibus.attachment.storage.location**), the payloads file locations will be stored in the JMS message This property should be disabled for large file transfers.
username	Mandatory in Multitenancy mode. The user that submits messages to Domibus. It is used to associate the current user with a specific domain

Name	Description
password	Mandatory in Multitenancy mode. The credentials of the user defined under the property **username**

Table 2 – JMS Message fields

The only mandatory rule is that only messageType=submitMessage messages may be put on the domibus.backend.jmsInQueue. All other queues (that go from the plugin to the backend) can be freely aggregated. I.e. if you only want one replyQueue you are free to send all success and errorMessages there.

3.2. Dynamic Discovery with JMS Plugin

*The fields in charge to identify the Party (C3) are empty in this scenario. (*toRole, toPartyId, toPartyType)

*The **finalRecipientType** is mandatory together with **finalRecipient** to enable messaging in Dynamic Discovery mode.*

The following example shows the relevant section of a function for submitting a message in Dynamic mode. The *complete function is shown in the **6 ANNEXE**.*

```
messageMap.setStringProperty("messageType", "submitMessage");

messageMap.setStringProperty("service", "urn:www.cenbii.eu:profile:bii04:ver1.0");

messageMap.setStringProperty("serviceType", "cenbii-procid-ubl");

messageMap.setStringProperty("action", "busdox-docid-
qns::urn:oasis:names:specification:ubl:schema:xsd:Invoice-
12::Invoice##urn:www.cenbii.eu:transaction:biicoretrdm010:ver1.0:#urn:www.peppol.eu:bis:peppol4a:
ver1.0::2.0");

messageMap.setStringProperty("fromPartyId", "senderalias");

messageMap.setStringProperty("fromPartyType", "urn:oasis:names:tc:ebcore:partyid-
type:unregistered");

messageMap.setStringProperty("fromRole", "http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/initiator");

messageMap.setStringProperty("originalSender", "urn:oasis:names:tc:ebcore:partyid-
type:unregistered:C1");

messageMap.setStringProperty("finalRecipient", "0777:tt001:oasis");

messageMap.setStringProperty("finalRecipientType", "iso6523-actorid-upis");

messageMap.setStringProperty("protocol", "AS4");
```

3.3. JMS-Queues

3.3.1. domibus.backend.jmsInQueue

Description:

Submit a message from a Backend to Domibus. If a property is set in the plugin properties (jms-plugin.properties) but not in the message itself, the value from the properties file will be used.

Message type: javax.jms.MapMessage

Property name	Optional	Available in plugin properties	Notes
messageType	No	No	Value = submitMessage
messageId	Yes	No	Must be a globally unique Id, max 255 characters long
action	No	Yes	
conversationId	Yes	No	
JMScorrelationId	Yes	No	Used by a backend to correlate between JMS messages submitted in jms.InQueue and response sent in jms.ReplyQueue.
fromPartyId	No	Yes	
fromRole	No	Yes	
fromPartyType	Yes	Yes	
toPartyId	Yes	Yes	Empty in **DYNAMIC DISCOVERY**
toRole	Yes	Yes	Empty in **DYNAMIC DISCOVERY**
toPartyType	Yes	Yes	
originalSender	Yes	No	
finalRecipient	Yes	No	**Mandatory in DYNAMIC DISCOVERY**
finalRecipientType	Yes	No	**Mandatory in DYNAMIC DISCOVERY**
service	No	Yes	
serviceType	Yes	Yes	Only optional if the service is untyped
protocol	Yes	No	Values other than AS4 or empty will raise an exception
refToMessageId	Yes	No	
agreementRef	Yes	Yes	
totalNumberOfPayloads	No	No	Outlining the total number of payloads, 0 payloads is valid

P1InBody (true/false)	Yes	No	If true, payload_1 will be sent in the body of the AS4 message. Only XML payloads may be sent in the AS4 message body.
putAttachmentInQueue (true/false)	Yes	Yes	If true, all the payloads from the User Message will be stored as bytes in the JMS message. If false and Domibus is configured to save the payloads on the filesystem (property **domibus.attachment.storage.location**), the payloads file locations will be stored in the JMS message This property should be disabled for large file transfers.
username	Yes	No	Mandatory in Multitenancy mode
password	Yes	No	Mandatory in Multitenancy mode

Table 3 - domibus.backend.jmsInQueue message fields

Payload handling:

The following properties should be set for each payload in the message. In the list below, the string "[NUM]" of each property name should be replaced with a numerical value representing each payload. The payload with the prefix payload_1 is transported inside the body of the AS4 message if the property p1InBody is set to true.

Each payload can either be sent in byte format or set in the MapMessage using the setBytes method of the MapMessage class, or an URL from where the payload can be downloaded by the Domibus Access Point. Each payload should be identified by the property payload_[NUM].

The following properties can be set for each payload using the setStringProperty method of the MapMessage class:

- payload_[NUM]_MimeContentID: For example the MimeContentID for the first payload will be identified by the property payload_1_MimeContentID. This is the payload contentId. Setting it is required if the pmode payload profiling is used. If unset Domibus generates an UUID for it.
- payload_[NUM]_MimeType: The mime type of the payload. If not provided the mime type application/octet-stream is assumed
- payload_[NUM]_FileName: The file location of the payload, if putAttachmentInQueue is set.

Property Handling

Message properties are handled in the following way:

- Properties named property_[NAME] are put into the outgoing message using [NAME] as key inside the AS4 message.
- For each property_[NAME] property there MAY be a corresponding propertyType_[NAME] property set. The corresponding value MAY be NULL, indicating an untyped property. Older AS4 implementations which do not have implemented the latest errata MIGHT REJECT messages where a property type is NOT NULL

3.3.2. domibus.backend.jms.replyQueue

Description: The result of the submit operation and contains either the messageId or an error. The messageId is (usually) generated by Domibus. If the submission is rejected, no messageId is generated. Additionally, there is no guarantee that the set MessageId of a rejected message can be read. This message has to be correlated using the JMSCorrelationID. Corner 2 reports back to corner 1 about the success/failure of an intended message submission.

Message type: javax.jms.Message

Property name	Optional	Notes
messageType	No	Value=submitResponse
messageId	Yes	null, if there is an errorDetail
errorDetail	Yes	null, if there is a messageId

Table 4 - domibus.backend.jms.replyQueue message fields

Description: A message has been successfully sent to another AS4 Access Point. The status changes to messageSent when the message has been sent from C2 to C3. The reason why this is a different logical queue is to allow better configuration options, i.e. you might want to send those messages to a monitoring system (or dev/null) and not to the back office application. As this is only a logical queue, nothing prevents it from using the same physical queue if all of those messages have the same recipient.

Property name	Optional	Notes
messageType	No	Value=messageSent
messageId	No	

Table 5 - domibus.backend.jms.replyQueue message fields

Payload handling: N/A

Property Handling: N/A

3.3.3. domibus.backend.jms.outQueue

Description: submit a message from Domibus (corner 3) to a backend (corner4)

Message type: javax.jms.MapMessage

Property name	Optional	Notes
messageType	No	Value = incomingMessage
messageId	No	Must be a globally unique Id
action	No	
conversationId	No	

Property name	Optional	Notes
fromPartyId	No	
fromRole	No	
fromPartyType	Yes	
toPartyId	No	
toRole	No	
toPartyType	Yes	
originalSender	Yes	
finalRecipient	Yes	
finalRecipienttype	Yes	**ONLY for DYNAMIC DISCOVERY**
service	No	
serviceType	Yes	Only optional if the service is untyped
protocol	No	Value = AS4
refToMessageId	Yes	
agreementRef	Yes	
totalNumberOfPayloads	No	outlining the total number of payloads, 0 payloads is valid

Table 6 - domibus.backend.jms.outQueue message fields

Payload handling:

The following properties are set for each payload in the message. In the list below, the string "[NUM]" of each property name is replaced with a numerical value representing each payload. If a payload has been transported in the message body of the corresponding AS4 message, this is always the payload with the prefix payload_1. Each payload is sent in byte format. Each payload is identified by the property payload_[NUM].

The following properties may be available for each payload:

- payload_[NUM]_MimeContentID: For example the MimeContentID for the first payload will be identified by the property payload_1_MimeContentID. This is the payload contentId. Setting it is required if the pmode payload profiling is used. If unset Domibus generates an UUID for it.
- payload_[NUM]_MimeType: The mime type of the payload
- payload_[NUM]_FileName: The file location of the payload, if putAttachmentInQueue is set.

Property Handling

Message properties are handled in the following way:

- Properties named [NAME] are put into the incoming message using property_[NAME] as key inside the JMS message
- For each property_[NAME] property there is a corresponding propertyType_[NAME] property set. The corresponding value MAY be NULL, indicating an untyped property. Older AS4 implementations which do not have implemented the latest errata will only ever send untyped properties

3.3.4. domibus.backend.jms.errorNotifyProducer

Description: A message that was accepted as submission could not be sent to the recipient.

Message type: javax.jms.Message

Property name	Optional	Notes
messageType	No	Value=messageSendFailure
messageId	No	
errorCode	No	The ebMS3 error code of the corresponding error
errorDetail	No	A textual description of the error

Table 7 - domibus.backend.jms.errorNotifyProducer message fields

Payload handling: N/A

3.3.5. domibus.backend.jms.errorNotifyConsumer

Description: An incoming message was rejected because of an error or agreement violation. To generate such a message, the Domibus Access Point must, at least, be able to determine the intended recipient for the original message. If this is not possible, no messageReceptionFailure will be generated.

Message type: javax.jms.Message

Property name	Optional	Notes
messageType	No	Value=messageReceptionFailure
messageId	No	
errorCode	No	The ebMS3 error code of the corresponding error
errorDetail	No	A textual description of the error
endPoint	Yes	The internet address of the access point that tried to send the message

Table 8 - domibus.backend.jms.errorNotifyConsumer message fields

Payload handling: N/A

Property Handling: N/A

3.3.6. Routing messages to specific queues

By default, the JMS Plugin dispatches UserMessages to the default configured queues. It does not have support to dispatch to different queues depending on the combination service/action.

This means that all UserMessages dispatched to the JMS Plugin will end up in the same queue, regardles of service/action values. This behaviour applies even if the UserMessage concerns different backend systems.

The possibility to dispatch UserMessages to specific JMS queues depending on different service/action value provides flexibility to address different backends.

This way flows belonging to different backend systems remain separated from each other. Overflooding the application by UserMessages having one combination of service/action will only have an impact on the latency of processing UserMessages for one backend but not the other ones.

For this purpose, the possibility of routing UserMessages depending on service/action combination has been implemented. This feature is available for the JMS queues configured using the following properties: jmsplugin.queue.out, jmsplugin.queue.reply, jmsplugin.queue.consumer.notification.error, jmsplugin.queue.producer.notification.error.

3.3.6.1. Defining routing rules for a specific queue

In order to define a routing rule for a specific queue the routing rule name must be defined first.

The convention is to define the routing rule name based on default queue property and the keyword *routing.* For instance, one can define a routing rule named *rule1* for the default JMS out queue defined under the property *jmsplugin.queue.out:*

jmsplugin.queue.out.routing.rule1=Routing rule description

Once the rule name is defined, other properties, like service, action and routing queue can be also defined using the rule name. For instance:

jmsplugin.queue.out.routing.rule1.service=ServiceValue

jmsplugin.queue.out.routing.rule1.action=ActionValue

jmsplugin.queue.out.routing.rule1.queue=jms/domibus.backend.jms.outQueue.queue1

Once a UserMessage having a service/action combination is matching a service/action combination configured for a rule than the UserMessage will be dispatched to the queue configured for the matching rule.

Note: service/action combinations configured for routing rules must be unique.

3.4. JMS Plugin Configuration

The Default JMS Plugin is configured using the **jms-plugin.properties** file. Below we describe the available properties from the configuration file:

3.4.1. Message properties

This set of properties contains default values for the business process. When a message is submitted to the JMS backend with missing business values, those values are defaulting to the business values configured in the **jms-plugin.properties** file.

Default values are defined for properties identifying the sending and the receiving parties, the business agreement and process. The complete list is available in section 3.3.1.

3.4.2. General properties

Property name	Default value	Description	Domain specific
jmsplugin.queue.notification	jms/domibus.notification.jms	This queue is used by Domibus to notify the JMS Plugin about message events.	No
jmsplugin.queue.in	jms/domibus.backend.jms.inQueue	This queue is the entry point for messages to be sent to Domibus via the JMS plugin	No
jmsplugin.queue.in.concurrency	5-20	Concurrency setting for the in queue Concurrency limits via a "lower-upper" String, e.g. "5-10", or a simple upper limit String, e.g. "10" (the lower limit will be 1 in this case)	No
jmsplugin.queue.out	jms/domibus.backend.jms.outQueue	This queue contains the received messages, the backend listens to this queue to consume the received messages	Yes
jmsplugin.queue.reply	jms/domibus.backend.jms.replyQueue	This queue is used to inform the backend about the message status after sending a message to Domibus	Yes
jmsplugin.queue.consumer.notification.error	jms/domibus.backend.jms.errorNotifyConsumer	This queue is used to inform the backend that an error occurred during the processing of receiving a message	Yes
jmsplugin.queue.producer.notification.error	jms/domibus.backend.jms.errorNotifyProducer	This queue is used to inform the backend that an error occurred during the processing of sending a message	Yes
jmsplugin.messages.notifications	MESSAGE_RECEIVED,MESSAGE_SEND_FAILURE,MESSAGE_RECEIVED_FAILURE,MESSAGE_SEND_SUCCESS,MESSAGE_STATUS_CHANGE	The notifications sent by Domibus to the plugin. The following values are possible: MESSAGE_RECEIVED,MESSAGE_FRAGMENT_RECEIVED,MESSAGE_SEND_FAILURE,MESSAGE_FRAGMENT_SEND_FAILURE,MESSAGE_RECEIVED_FAILURE,MESSAGE_FRAGMENT_RECEIVED_FAILURE,MESSAGE_SEND_SUCCESS,MESSAGE_FRAGMENT_SEND_SUCCESS,MESSAGE_STATUS_CHANGE,MESSAGE_FRAGMENT_STATUS_CHANGE	

Table 9 - General properties

3.5. Referencing Payloads

When using the JMS plugin exchange messages with Domibus, the payloads are embedded in the JMS message binary format. In order to send payloads embedded in the JMS messages, the following property of the JMS Plugin must be configured as below:

jmsplugin.putAttachmentInQueue = true

This is perfectly fine when sending small payloads, but it might become problematic when sending large files as JMS brokers are not designed to support large payloads.

When exchanging large files via the JMS plugin, it is preferred to store the payloads outside of the JMS messages and reference the location of the payloads in the JMS message as string properties.

There are two ways to reference the payloads in the JMS messages:

1. File reference
2. REST endpoint reference

3.5.1. File reference

In order to use payload file reference, Domibus must be configured to store the payloads on the file system. Please check the **Domibus Administration Guide** for more information on how to do this.

Once Domibus has been configured to store the payloads on the file system, the following properties of the JMS Plugin must be configured as below:

jmsplugin.putAttachmentInQueue = false

jmsplugin.attachment.reference.type = FILE

The payload file reference is set as a JMS string property having the following name:

payload_[NUM]_fileName where *NUM* is the payload number

3.5.2. REST endpoint reference

In order to use payload REST endpoint reference, the following properties of the JMS Plugin must be configured as below:

jmsplugin.putAttachmentInQueue = false

jmsplugin.attachment.reference.type = URL

Once activated, the payload URL will be included as a string property in the JMS message: e.g.:

***http://localhost:8080**/domibus/ext/messages/ids/MESSAGE_ENTITY_ID/payloads/PAYLOAD_CID*

where:

- *MESSAGE_ENTITY_ID* is the internal primary key id of the UserMessage
- *PAYLOAD_CID* is the cid of the UserMessage payload

All Domibus REST endpoints, included the ones above, are protected using basic authentication. Please use a plugin user in order to authenticate, for more information please check the **Domibus Administration Guide.**

The base context ***http://localhost:8080*** can be configured using the following property of the JMS Plugin as below:

jmsplugin.attachment.reference.context=http://localhost:8080/domibus

The payload REST endpoint reference is set a as a JMS string property having the following name:

payload_[NUM] _fileURL where *NUM* is the payload number

4. PLUGIN NOTIFICATIONS

Domibus core notifies the JMS Plugin on the following events: MESSAGE_RECEIVED, MESSAGE_SEND_FAILURE, MESSAGE_RECEIVED_FAILURE, MESSAGE_SEND_SUCCESS, MESSAGE_STATUS_CHANGE.

The type of events received can be configured using the JMS Plugin property: *jmsplugin.messages.notifications.* The property can be found in "jms-plugin.properties" file in the domibus-distribution-xxx-default-jms-plugin.zip. More details can be found in the Plugin Cookbook.

5. MULTITENANCY

The Default JMS Plugin can be used when Domibus is configured in Multitenancy mode.

In Multitenancy mode the plugins security is activated by default, no matter if value configured in **domibus.properties** for the **domibus.auth.unsecureLoginAllowed** property.

As a result, every request sent to the **domibus.backend.jmsInQueue** queue via the **Default JMS Plugin** needs to be authenticated via the **user** and **password** JMS properties. More information on how to create plugin users used for authentication can be found in the **Domibus Administration Guide**, in the section **Plugin Users.**

Please note that the default domain is already configured to use the Default JMS Plugin in Multitenancy mode and the below steps must be followed only for additional domains.

Each configured domain that is using the **Default JMS Plugin** to send messages to Domibus has to create the following JMS queues that will be used exclusively by the domain:

- **DOMAIN**.domibus.backend.jms.outQueue
- **DOMAIN**.domibus.backend.jms.replyQueue
- **DOMAIN**.domibus.backend.jms.errorNotifyConsumer
- **DOMAIN**.domibus.backend.jms.errorNotifyProducer

where **DOMAIN** is the domain name.

The backend C1 linked to a specific domain must subscribe to the associated JMS domain queues in order to receive notifications linked to that domain

More details on the above queues and the structure of the sent and received messages via the **Default JMS Plugin** can be found in the previous chapter.

The above mentioned queues have to be configured in the JMS broker specific to the chosen server: **activemq.xml** for Tomcat and in the application server configuration for WebLogic and WildFly. The details on how to configure JMS queues specific to a server can be found in the **Domibus Administration Guide.**

Once created the domain queues have to be configured in the **jms-plugin.properties** configuration file.

```
#Domain configuration
#The following queues need to be created per domain. Please replace the "DOMAIN" value with the domain code.
#It is recommended to secure the queues so that only users belonging to "DOMAIN" can read.
DOMAIN.jmsplugin.queue.out=DOMAIN.domibus.backend.jms.outQueue
DOMAIN.jmsplugin.queue.reply=DOMAIN.domibus.backend.jms.replyQueue
DOMAIN.jmsplugin.queue.consumer.notification.error=DOMAIN.domibus.backend.jms.errorNotifyConsumer
DOMAIN.jmsplugin.queue.producer.notification.error=DOMAIN.domibus.backend.jms.errorNotifyProducer
```

5.1. Domain specific properties

The JMS Plugin configuration allows configuring specific properties per domain. The entire properties specific to a domain must be prefixed by the domain name.

Domain configuration Property	Description
DOMAIN.jmsplugin.fromPartyId	Sender party ID
DOMAIN.jmsplugin.fromPartyType	Sender party type
DOMAIN.jmsplugin.fromRole	Sender party role
DOMAIN.jmsplugin.toPartyId	Receiver party ID
DOMAIN.jmsplugin.toPartyType	Receiver party type
DOMAIN.jmsplugin.toRole	Receiver party role
DOMAIN.jmsplugin.agreementRef	Agreement reference
DOMAIN.jmsplugin.service	Service value
DOMAIN.jmsplugin.serviceType	Service type
DOMAIN.jmsplugin.action	Action value
DOMAIN.jmsplugin.putAttachmentInQueue	Default value is true. If configured to true, all the payloads from the User Message will be stored as bytes in the JMS message. If configured to false and Domibus is configured to save the payloads on the file system (property **domibus.attachment.storage.location** is configured), the payloads file locations will be stored in the JMS message. This property should be disabled for large file transfers.

6. ANNEX 1 – INTERFACE POLICY SPECIFICATION

The Party initiating a conversation MUST determine the value of the ConversationId element that is reflected in all messages pertaining to that conversation. The actual semantics of this value is beyond the scope of this specification. Implementations SHOULD provide a facility for mapping between their identification scheme and a ConversationId generated by another implementation.

The following details 2 simple functions, one for **Static Discovery mode**, the other for **Dynamic Discovery mode**, for submitting a message in the correct format to a queue where it will be picked up by a MessageListener on the Access Point.

STATIC DISCOVERY MODE:

```
package eu.domibus.plugin.jms;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.junit.Ignore;
import org.junit.Test;
import javax.jms.*;
import javax.naming.NamingException;
public class MessageSender {
    @Test
    @Ignore //This is just an example the used PMode does not actually exist
    public void sendMessage() throws NamingException, JMSException {
        ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://localhost:61616");//default port of activeMQ
        Connection connection = null;
        MessageProducer producer = null;
        connection = connectionFactory.createConnection("domibus", "changeit"); //username and password of
the default JMS broker
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Destination destination = session.createQueue("domibus.backend.jms.inQueue");
        producer = session.createProducer(destination);
        producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        MapMessage messageMap = session.createMapMessage();
        // Declare message as submit
        messageMap.setStringProperty("messageType", "submitMessage");
        // Set up the Communication properties for the message
        messageMap.setStringProperty("service", "demoService");
        messageMap.setStringProperty("action", "demoAction");
        messageMap.setStringProperty("conversationId", "");
        messageMap.setStringProperty("fromPartyId", "GW1");
        messageMap.setStringProperty("fromPartyIdType", "urn:oasis:names:tc:ebcore:partyid-type:iso3166-
1");
        messageMap.setStringProperty("fromRole", "buyer");
        messageMap.setStringProperty("toPartyId", "GW1");
        messageMap.setStringProperty("toPartyIdType", "urn:oasis:names:tc:ebcore:partyid-type:iso3166-1");
        messageMap.setStringProperty("toRole", "seller");
        messageMap.setStringProperty("originalSender", "sending_buyer_id");
        messageMap.setStringProperty("finalRecipient", "receiving_seller_id");
        messageMap.setStringProperty("serviceType", "");
        messageMap.setStringProperty("protocol", "AS4");
        messageMap.setStringProperty("refToMessageId", "");
        messageMap.setStringProperty("agreementRef", "");
        messageMap.setJMSCorrelationID("MESS1");
        //Set up the payload properties
        messageMap.setStringProperty("totalNumberOfPayloads", "3");
        messageMap.setStringProperty("payload_1_mimeContentId", "cid:cid_of_payload_1");
        messageMap.setStringProperty("payload_2_mimeContentId", "cid:cid_of_payload_2");
        messageMap.setStringProperty("payload_3_mimeContentId", "cid:cid_of_payload_3");
        messageMap.setStringProperty("payload_1_mimeType", "application/xml");
        messageMap.setStringProperty("payload_2_mimeType", "application/xml");
        messageMap.setStringProperty("payload_3_mimeType", "application/xml");
        messageMap.setStringProperty("payload_1_fileName", "filenameLocation1");
        messageMap.setStringProperty("payload_2_fileName", "filenameLocation2");
        messageMap.setStringProperty("payload_3_fileName", "filenameLocation3");
        String pay1 = "<XML><test></test></XML>";
        byte[] payload = pay1.getBytes();
        messageMap.setBytes("payload_1", payload);
        messageMap.setBytes("payload_2", payload);
        messageMap.setBytes("payload_3", payload);
        producer.send(messageMap);
        connection.close();
    }
}
```

DYNAMIC DISCOVERY MODE:

```
package eu.domibus.plugin.jms;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.junit.Ignore;
import org.junit.Test;
import javax.jms.*;
import javax.naming.NamingException;
public class MessageSender {
    @Test
    @Ignore //This is just an example the used PMode does not actually exist
    public void sendMessage() throws NamingException, JMSException {
        ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://localhost:61616");//default port of activeMQ
        Connection connection = null;
        MessageProducer producer = null;
        connection = connectionFactory.createConnection("domibus", "changeit"); //username and password of
the default JMS broker
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Destination destination = session.createQueue("domibus.backend.jms.inQueue");
        producer = session.createProducer(destination);
        producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        MapMessage messageMap = session.createMapMessage();
        // Declare message as submit
        messageMap.setStringProperty("messageType", "submitMessage");
        // Set up the Communication properties for the message
        messageMap.setStringProperty("service", "demoService");
        messageMap.setStringProperty("action", "demoAction");
        messageMap.setStringProperty("conversationId", "");
        messageMap.setStringProperty("fromPartyId", "GW1");
        messageMap.setStringProperty("fromPartyIdType", "urn:oasis:names:tc:ebcore:partyid-type:iso3166-
1");
        messageMap.setStringProperty("fromRole", "buyer");
        messageMap.setStringProperty("originalSender", "sending_buyer_id");
        messageMap.setStringProperty("finalRecipient", "receiving_seller_id");
        messageMap.setStringProperty("finalRecipienttype", " receiving_seller_id_type");
        messageMap.setStringProperty("serviceType", "");
        messageMap.setStringProperty("protocol", "AS4");
        messageMap.setStringProperty("refToMessageId", "");
        messageMap.setStringProperty("agreementRef", "");
        messageMap.setJMSCorrelationID("MESS1");
        //Set up the payload properties
        messageMap.setStringProperty("totalNumberOfPayloads", "3");
        messageMap.setStringProperty("payload_1_mimeContentId", "cid:cid_of_payload_1");
        messageMap.setStringProperty("payload_2_mimeContentId", "cid:cid_of_payload_2");
        messageMap.setStringProperty("payload_3_mimeContentId", "cid:cid_of_payload_3");
        messageMap.setStringProperty("payload_1_mimeType", "application/xml");
        messageMap.setStringProperty("payload_2_mimeType", "application/xml");
        messageMap.setStringProperty("payload_3_mimeType", "application/xml");
        messageMap.setStringProperty("payload_1_fileName", "filenameLocation1");
        messageMap.setStringProperty("payload_2_fileName", "filenameLocation2");
        messageMap.setStringProperty("payload_3_fileName", "filenameLocation3");
        String pay1 = "<XML><test></test></XML>";
        byte[] payload = pay1.getBytes();
        messageMap.setBytes("payload_1", payload);
        messageMap.setBytes("payload_2", payload);
        messageMap.setBytes("payload_3", payload);
        producer.send(messageMap);
        connection.close();
    }
}
```

7. ANNEX 2 - ERRORS CODES TABLE

The following tables summarize all possible errors returned by the Access Point services:

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS_0001	ValueNotRecognized	failure	Content	Although the message document is well formed and schema valid, some element/attribute contains a value that could not be recognized and therefore could not be used by the MSH.
EBMS_0002	FeatureNotSupported	warning	Content	Although the message document is well formed and schema valid, some element/attribute value cannot be processed as expected because the related feature is not supported by the MSH.
EBMS_0003	ValueInconsistent	failure	Content	Although the message document is well formed and schema valid, some element/attribute value is inconsistent either with the content of other element/attribute, or with the processing mode of the MSH, or with the normative requirements of the ebMS specification.
EBMS_0004	Other	failure	Content	
EBMS_0005	ConnectionFailure	failure	Communication	The MSH is experiencing temporary or permanent failure in trying to open a transport connection with a remote MSH.
EBMS_0006	EmptyMessagePartitionChannel	warning	Communication	There is no message available for pulling from this MPC at this moment.
EBMS_0007	MimeInconsistency	failure	Unpackaging	The use of MIME is not consistent with the required usage in this specification.
EBMS_0008	FeatureNotSupported	failure	Unpackaging	Although the message document is well formed and schema valid, the presence or absence of some element/ attribute is not consistent with the capability of the MSH, with respect to supported features.
EBMS_0009	InvalidHeader	failure	Unpackaging	The ebMS header is either not well formed as an XML document, or does not conform to the ebMS packaging rules.
EBMS_0010	ProcessingModeMismatch	failure	Processing	The ebMS header or another header (e.g. reliability, security)

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
				expected by the MSH is not compatible with the expected content, based on the associated P-Mode.
EBMS_0011	ExternalPayloadError	failure	Content	The MSH is unable to resolve an external payload reference (i.e. a Part that is not contained within the ebMS Message, as identified by a PartInfo/href URI).
EBMS_0101	FailedAuthentication	failure	Processing	The signature in the Security header intended for the "ebms" SOAP actor could not be validated by the Security module.
EBMS_0102	FailedDecryption	failure	Processing	The encrypted data reference the Security header intended for the "ebms" SOAP actor could not be decrypted by the Security Module.
EBMS_0103	PolicyNoncompliance	failure	Processing	The processor determined that the message's security methods, parameters, scope or other security policy-level requirements or agreements were not satisfied.
EBMS_0201	DysfunctionalReliability	failure	Processing	Some reliability function as implemented by the Reliability module is not operational, or the reliability state associated with this message sequence is not valid.
EBMS_0202	DeliveryFailure	failure	Communication	Although the message was sent under Guaranteed delivery requirement, the Reliability module could not get assurance that the message was properly delivered, in spite of resending efforts.
EBMS_0301	MissingReceipt	failure	Communication	A Receipt has not been received for a message that was previously sent by the MSH generating this error
EBMS_0302	InvalidReceipt	failure	Communication	A Receipt has been received for a message that was previously sent by the MSH generating this error, but the content does not match the message content (e.g. some part has not been acknowledged, or the digest associated does not match the signature digest, for NRR).
EBMS_0303	DecompressionFailure	failure	Communication	An error occurred during the decompression
EBMS_0020	RoutingFailure	failure	Processing	An Intermediary MSH was

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
				unable to route an ebMS message and stopped processing the message.
EBMS_0021	MPCCapacityExceeded	failure	Processing	An entry in the routing function is matched that assigns the message to an MPC for pulling, but the intermediary MSH is unable to store the message with this MPC
EBMS_0022	MessagePersistenceTimeout	failure	Processing	An intermediary MSH has assigned the message to an MPC for pulling and has successfully stored it. However, the intermediary set a limit on the time it was prepared to wait for the message to be pulled, and that limit has been reached.
EBMS_0023	MessageExpired	warning	Processing	An MSH has determined that the message is expired and will not attempt to forward or deliver it.
EBMS_0030	BundlingError	failure	Content	The structure of a received bundle is not in accordance with the bundling rules.
EBMS_0031	RelatedMessageFailed	failure	Processing	A message unit in a bundle was not processed because a related message unit in the bundle caused an error.
EBMS_0040	BadFragmentGroup	failure	Content	A fragment is received that relates to a group that was previously rejected.
EBMS_0041	DuplicateMessageSize	failure	Content	A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS_0042	DuplicateFragmentCount	failure	Content	A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS_0043	DuplicateMessageHeader	failure	Content	A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS_0044	DuplicateAction	failure	Content	A fragment is received but more than one fragment message in a group of fragments specifies a value for this element.
EBMS_0045	DuplicateCompressionInfo	failure	Content	A fragment is received but more than one fragment message in a group of fragments specifies a value for a compression element.
EBMS_0046	DuplicateFragment	failure	Content	A fragment is received but a previously received fragment message had the same values

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
				for GroupId and FragmentNum
EBMS_0047	BadFragmentStructure	failure	Unpackaging	The href attribute does not reference a valid MIME data part, MIME parts other than the fragment header and a data part are in the message, or the SOAP Body is not empty.
EBMS_0048	BadFragmentNum	failure	Content	An incoming message fragment has a value greater than the known FragmentCount.
EBMS_0049	BadFragmentCount	failure	Content	A value is set for FragmentCount, but a previously received fragment had a greature value.
EBMS_0050	FragmentSizeExceeded	warning	Unpackaging	The size of the data part in a fragment message is greater than Pmode[].Splitting.FragmentSize
EBMS_0051	ReceiveIntervalExceeded	failure	Unpackaging	More time than Pmode[].Splitting.JoinInterval has passed since the first fragment was received but not all other fragments are received.
EBMS_0052	BadProperties	warning	Unpackaging	Message properties were present in the fragment SOAP header that were not specified in Pmode[].Splitting.RoutingProperties
EBMS_0053	HeaderMismatch	failure	Unpackaging	The eb3:Message header copied to the fragment header does not match the eb3:Message header in the reassembled source message.
EBMS_0054	OutOfStorageSpace	failure	Unpackaging	Not enough disk space available to store all (expected) fragments of the group.
EBMS_0055	DecompressionError	failure	Processing	An error occurred while decompressing the reassembled message.
EBMS_0060	ResponseUsingAlternateMEP	Warning	Processing	A responding MSH indicates that it applies the alternate MEP binding to the response message.

Table 10 - Annex 2 - Errors codes table

8. ANNEXE 3 – DOCUMENT PARTS

(eDelivery)(Domibus)
(ICD)(Default JMS Plu

9. List of Figures

10. List of Tables

11. CONTACT INFORMATION

eDelivery Support Team

By email: EC-EDELIVERY-SUPPORT@ec.europa.eu

Support Service: 8am to 6pm (Normal EC working Days)