EUROPEAN COMMISSION

DIGIT
Digital Europe Programme

Access Point

Domibus 5.0.8

Extension Cookbook

Version [2.5]

Status [final]

Date: 18/03/2024

Table of Contents

1. INTRODUCTION

1.1. Purpose

The purpose of this document is to describe the technical specifications of Domibus extension mechanism. In particular, this document lays out applicable guidelines to support the technical implementation of an extension.

1.2. Scope

The scope of this document is to define:

- The functional aspects of the extension mechanism,
- The technical and operational aspects of the extension mechanism.

1.3. Audience

This document is intended for the Directorate Generals and Services of the European Commission, Member States (MS) and companies of the private sector wanting to customize the incoming AS4 messages authorisation and signing certificate trust validation.

In particular:

- Business Architects will find it useful for understanding the possible integration towards external trust systems.
- Analysts and developers will find it useful to understand and implement how to customize message trust and authorisation.
- Testers can use this document to test the use cases described.

1.4. Structure of this document

The present document contains the following chapters:

- **Chapter 1: Introduction** - describes the scope and purpose of this document.
- **Chapter 2: Overview** - provides an overview of the extension mechanism and the links to the associated Domibus documentation.
- **Chapter 3: Extension functional information** - describes the functional specifications of the existing extension mechanism and gives an overview of the customisation possibilities.
- **Chapter 4: Extension technical information -** describes the technical specifications of the existing extension mechanism. This chapter describes the extension interfaces in depth.
- **Chapter 5: Extension operational information -** describes the operational aspects of building and registering a custom extension.

- **Chapter 6: Annex -** references configuration files that are useful for operational aspects.
- **Chapter 7: Contact information**.

1.5. References

Ref.	Title	Content outline
[REF1]	Domibus Software Architecture Document	This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict individual aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.
[REF2]	Domibus Administration Guide	This document is intended for Server Administrators in charge of installing, managing and troubleshooting an eDelivery Access Point.
[REF3]	OASIS ebXML Messaging Services	These specifications describe a communication-protocol neutral method for exchanging electronic 285 business messages.
[REF4]	[WSSX509] OASIS Web Services Security	Web Services Security X.509 Certificate Token Profile Version 1.1.1. OASIS Standard.
[REF5]	X509Certificate	X509Certificate specifications.
[REF6]	Maven	Maven portal.
[REF7]	Maven shade plugin	Maven shade plugin portal.
[REF8]	Certificate Trust	Personal Identity Verification Portal explaining the concept of certificate trust.
[REF9]	WS-SecurityPolicy	WS-Policy defines a framework for allowing web services to express their constraints and requirements.
[REF10]	Properties file format	The properties files must follow a format that is recognised by the java.util.Properties class.

2. OVERVIEW

The extension cookbook defines the technical and operational aspects of Domibus extension mechanism with links to the functional specifications. It also provides guidelines for the adequate implementation of the interfaces.

The extension mechanism allows the customisation of AS4 message signing certificate trust validation and AS4 message authorisation in a flexible way.

Extensions are dependent on the ***domibus-iam-spi*** module, which is released together with the main Domibus application. Any changes to previous API versions will be addressed in a migration guide.

It is assumed that the reader is aware of the operational, technical and functional context of eDelivery Domibus access point:

- Domibus Software Architecture Document [REF1]
- Domibus Administration Guide [REF2]
- OASIS ebXML Messaging Services [REF3]

3. EXTENSION FUNCTIONAL INFORMATION

3.1. Trust validation

3.1.1. Overview

Identity certificates are issued and digitally signed by a Certificate Authority. The Certificate Authority that signed your own certificates is called an Intermediate Certificate Authority (ICA), because it was issued by another Certificate Authority. This process of issuing and signing continues until there is one Certificate Authority that is called the Root Certificate Authority (CA).

The whole process of proving identity when issuing the certificates, auditing the certificate authorities and the cryptographic protections of the digital signatures establish the basis of Trust for certificates (see also [REF8]).

3.1.2. Default Domibus behaviour

Domibus uses WS-Policy [REF9] framework to express the access point MSH webservice constraints and requirements. Please consult the section *"Security Policies"* in the Domibus Administration guide [REF2] to have an overview of the different configuration provided with the Domibus Distribution.

Domibus uses an on disk (private) KeyStore to store the access point private certificate and a second on disk (public) KeyStore to store public certificates of counterpart access point. Please refer to section *"Certificates"* in the Domibus Administration Guide [REF2] for setup instructions.

Domibus can ensure the trust in two ways:

1. Direct trust: if the leaf certificate is present and valid in the public KeyStore,
2. Indirect trust: if the leaf certificate is not present in the public KeyStore, the leaf certificate issuer is checked until the certificate root is reached. Practically, Domibus trusts an incoming message if the certificate chain, excluding the leaf, is present and valid in the public KeyStore. This is typically the case for a dynamic receiver profile.

Incoming AS4 messages are signed with the sender certificate. The AS4 protocol uses WSS SOAP Message Security which references X509 certificate by one of the following means:

- Reference to a subject key identifier

To configure Domibus to reference certificate by subject key identifier, please use the provided eDeliveryAS4Policy policy.

Domibus will use the subject key identifier contained in the AS4 SOAP envelope to extract the signing certificate from the public KeyStore. If the certificate is found and valid, the AS4 message will be considered as trusted. This type of configuration only supports direct trust method.

- Reference to an issuer and a serial number

To configure Domibus to reference certificate by issuer and serial number, please use the provided eDeliveryAS4Policy_IS policy.

Domibus will use issuer and serial number contained in the AS4 SOAP envelope to extract the signing certificate from the public KeyStore. If the certificate is found and valid, the AS4 message will be considered as trusted. This type of configuration only supports direct trust method.

- Reference to a binary security token

To configure Domibus to extract the certificate out of the AS4 message SOAP envelope, please use the provided eDeliveryAS4Policy_BST policy.

Domibus will extract the signing certificate from the AS4 SOAP envelope and use its issuer and serial number to retrieve its equivalent from the public KeyStore.

If the certificate is found and valid, the AS4 message will be considered as trusted. If the leaf certificate is not found within the public KeyStore, Domibus will try to trust the message against the trust chain as explain above. This type of configuration supports both direct and indirect trust method.

- Reference to a Binary Security token with PKI path

To configure Domibus to extract the certificate and its trust certificate chain directly out of the AS4 message SOAP envelope, please use the provided eDeliveryAS4Policy_BST_PKI policy.

Generally, the chain is composed of the leaf signing certificate, the ICA and the CA.

Domibus verifies the signing trust path of the certificate chain. To achieve this verification, Domibus expects to find the CA certificate of the chain within the public KeyStore.

If the CA is present and valid, and the signing path of the chain is valid, the AS4 message will be considered as trusted. This type of configuration only supports indirect trust method.

3.1.3. Custom behaviour

By creating a new Trust extension, one can bypass the default Domibus behaviour previously explained.

There is one certificate trust validation extension provided with the Domibus distribution: the domibus-authentication-dss-extension.

For more information about the DSS extension, please refer to the chapter "*DSS Extension Configuration"* in Domibus Administration guide [REF2].

3.1.4. Trust validation sequence

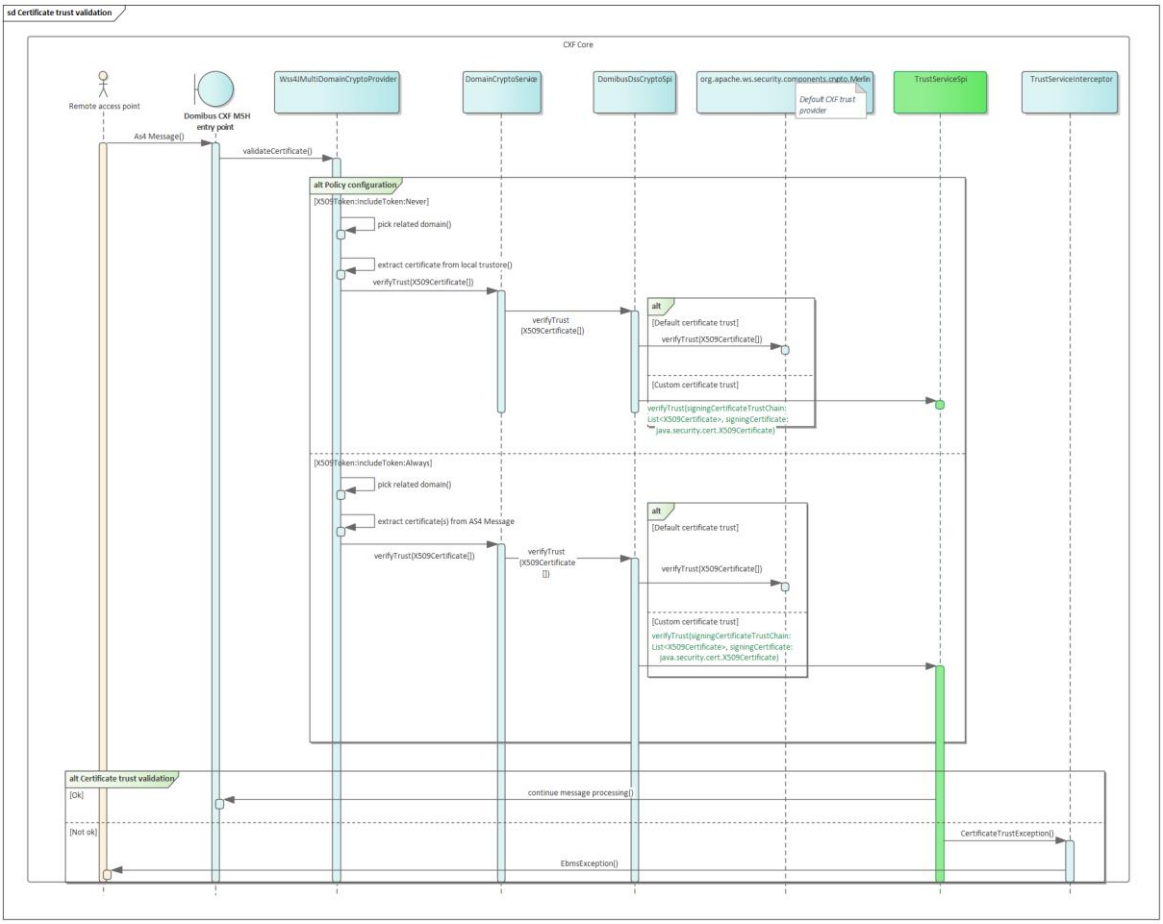

Figure 1- Incoming message certificate trust sequence

3.2. Authorisation

3.2.1. Overview

The authorisation extension goal is to provide the possibility to accept or refute messages based on an aggregation of data extracted from different parts of the Domibus system. Information related to certificate, PMode and AS4 message content is provided to the extension.

3.2.2. Default Domibus behaviour

The default Domibus trust mechanism is verifying AS4 message authorisation with the following configurable means:

- Subject expression validation

By setting a valid regular expression within the “domibus.sender.trust.validation.expression” property, Domibus will apply the regular expression to the signing certificate “subject distinguished name” (see also[REF5]).

If the subject distinguished name does not match with the regular expression, the authorisation process will end and reject the AS4 message.

If “domibus.sender.trust.validation.expression” is empty, no validation is performed on the certificate subject distinguished name.

- Public keystore alias validation

By setting “domibus.sender.trust.validation.truststore_alias” property to true, Domibus will extract the party name configured in the PMode that corresponds to the PartyId present within the AS4Message.

The process is as follows:

1. The process looks in the PMode for the value “configuration/businessProcesses/parties/identifier@partyId” that is equal to the value “/eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId” from the AS4 message.

2. Out of that entry, the process extracts the value “configuration/businessProcesses/parties/party@name” to retrieve the name attributed to that party within the PMode.

3. The party name is then used as an alias to extract a certificate from the public KeyStore.
   - If the certificate found is not equal to the one used to sign the message, the authorisation process will end and reject the AS4 message.
   - If no certificate is found for the given alias, Domibus logs a warning but the authorisation process continues.

By setting “domibus.sender.certificate.subject.check” property to true, and extra validation is performed on the alias. The authorisation process verifies that the certificate “subject distinguished name” contains the alias. If it does not, the authorisation process will end and reject the AS4 message.

3.2.3. Custom behaviour

By creating a new authorisation extension, one can bypass the default Domibus behaviour previously explained.

An authorisation extension can use information from the AS4 Message, the signing certificate and the PMode to perform a custom authorisation.

No default implementation is provided with Domibus.

3.2.4. Authorisation sequence

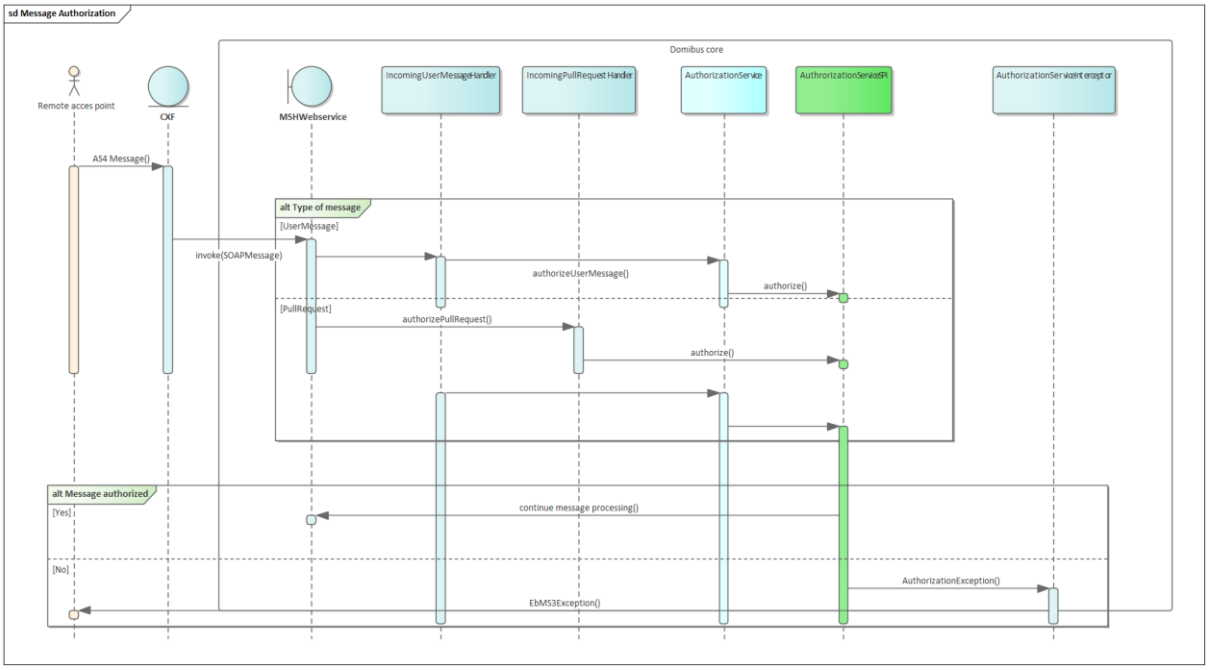

Figure 2- Incoming message authorisation sequence

3.3. Caching

Domibus has two types of caching mechanisms available: local and distributed.

The local cache is available when Domibus is deployed in a single instance and in a cluster. The local cache must be used when you do not want to replicate the cache amongst the cluster deployment.

The distributed cache is only available in a cluster deployment. It should be used when you want to replicate the cache amongst the cluster members.

3.3.1. Local cache

Domibus uses Ehcache implementation for local caching using @Cacheable annotations or programmatically.

At extension level it is possible to add two configuration files for Ehcache, to define their own cache names. However, please note that the caches will be merged to Domibus Ehcache manager.

3.3.1.1. Default Ehcache file (classpath file)

The expected format is *-extension-default-ehcache.xml e.g. ext1-extension-default-ehcache.xml

The expected location (in a Java project) is: /src/main/resources/config/ehcache/*-extension-default-ehcache.xml which will go to .jar file of the extension into e.g. /config/ehcache/ext1-extension-default-ehcache.xml location

The cache names defined in this file should be prefixed with the extension name to avoid collision with Domibus cache names. Otherwise, Domibus will throw an exception and will not deploy:

```
<cache alias="ext1.policyCache"

    maxBytesLocalHeap="5m"

    timeToLiveSeconds="3600"

    overflowToDisk="false">

</cache>
```

3.3.1.2. External cache

The expected format is *-extension-ehcache.xml e.g ext1-extension-ehcache.xml

The expected location is: ${domibus.config.location}/extensions/config

The cache names defined here could override the one defined in the classpath file (see above), they must not override/collide with Domibus cache names:

```
<cache alias="ext1.policyCache"

    maxBytesLocalHeap="5m"

    timeToLiveSeconds="360"

    overflowToDisk="false">

</cache>
```

In the above file, ext1.policycache overrides the one defined in the classpath file.

The following algorithms will be used:

A. Domibus parses all extension default files from classpath (.jar) - -default-ehcache.xml and adds all caches to an extension cache manager (in memory).

B. Domibus parses all extension non default files - from /extensions/config folders - the caches are added to the same extension cache manager, and they may override the caches defined at step A.

C. Domibus merges the extension cache manager (step A+B) to the Domibus cache manager. If there is an extension cache which is overriding a Domibus cache -> a DomibusCoreException is thrown and the deployment is stopped

3.3.2. Distributed cache

The distributed cache is only available in a cluster deployment. In a non-cluster deployment, the distributed cache defaults to the local cache.

In a cluster deployment, once you add an entry in a distributed cache, the change is replicated automatically amongst the cluster members.

The distributed cache is accessible via Java API and via REST.

A custom extension can have access to the distributed cache via the eu.domibus.ext.services. DistributedCacheExtService Java class.

You can find below the methods available:

- *void createCache(String cacheName): creates or gets a distributed cache with the specified name. If the cache does not exist, it will be created with the default values and near cache configuration specified in domibus-default.properties*
- *void createCache(String cacheName, int cacheSize, int timeToLiveSeconds, int maxIdleSeconds): creates or gets a distributed cache with the specified name and configuration. If the cache does not exist, it will be created with the specified configuration and near cache configuration specified in domibus-default.properties*
- *void createCache(String cacheName, int cacheSize, int timeToLiveSeconds, int maxIdleSeconds, int nearCacheSize, int nearCacheTimeToLiveSeconds, int nearCacheMaxIdleSeconds): creates or gets a distributed cache with the specified name and configuration.*
- *void addEntryInCache(String cacheName, String key, Object value) throws CacheExtServiceException: adds an entry in the cache*
- *Object getEntryFromCache(String cacheName, String key) throws CacheExtServiceException: gets an entry from the cache*
- *void evictEntryFromCache(String cacheName, String key) throws CacheExtServiceException: evicts an entry from the cache*

For more details about the distributed caching, please check Domibus Software Architecture document and the Domibus Administration Guide.

4. EXTENSION TECHNICAL INFORMATION

4.1. Implementing TrustServiceSpi interface

The interface has 2 methods:

- verifyTrust
- getIdentifier

4.1.1. *The verifyTrust method*

The *verifytrust* method purpose is to validate the trust of the incoming AS4 message signing certificate.

As explain in section 3.1, the location from where the signing certificate is extracted depends on the security policy used.

The Domibus security policy configuration will determine from where to extract the certificates that will be passed as parameter to the *verifytrust* method.

If the method executes without exception, the signing certificate of the incoming AS4 Message will be considered as trusted and the processing of the message will resume.

```
void verifyTrust(List<X509Certificate> signingCertificateTrustChain,
        X509Certificate signingCertificate) throws CertificateTrustException;
```

If any runtime exception is triggered, the message is refused, and the exception is transformed into an EBMS exception by Domibus.

By throwing a CertificateTrustException exception, one can more precisely control the type of EBMS exception thrown by Domibus. A CertificateTrustException to EBMSException mapping table is documented in section 4.1.1.2.

The following table documents the location of certificates used as verifyTrust method parameters, based on the security policies provided with Domibus.

Policy name	Certificate token reference	Certificate location	Description
eDeliveryAS4Policy.xml	Key identifier Please review section 3.3.1 of OASIS Web Services Security document REF4.	Receiver trust store	With this policy, Domibus extracts the signing certificate from the configured public KeyStore.

Policy name	Certificate token reference	Certificate location	Description
eDeliveryAS4Policy_BST.xml	Binary security token Please review 3.3.2 of OASIS Web Services Security document [REF4].	AS4 Message	With this policy, Domibus extracts the signing certificate from the AS4 message.
eDeliveryAS4Policy_BST_PKIP.xml	Binary security token with PKIPATH Please review section 3.1.2 and 3.3.2 of OASIS Web Services Security document [REF4].	AS4 Message	With this policy, Domibus extracts the signing certificate and its truth path from the AS4 message.
eDeliveryAS4Policy_IS.xml	Issuer and serial number. Please review section 3.3.3 of OASIS Web Services Security document [REF4].	Receiver trust store	With this policy, Domibus extracts the signing certificate from the configured public KeyStore.

Only the eDeliveryAS4Policy_BST and eDeliveryAS4Policy_BST_PKIP policies extract the signing certificate from the AS4 Message. Therefore, if the trust validation custom extension requires not to use the public KeyStore for incoming messages trust validation, one of those policies should be used.

4.1.1.1. Data dictionary

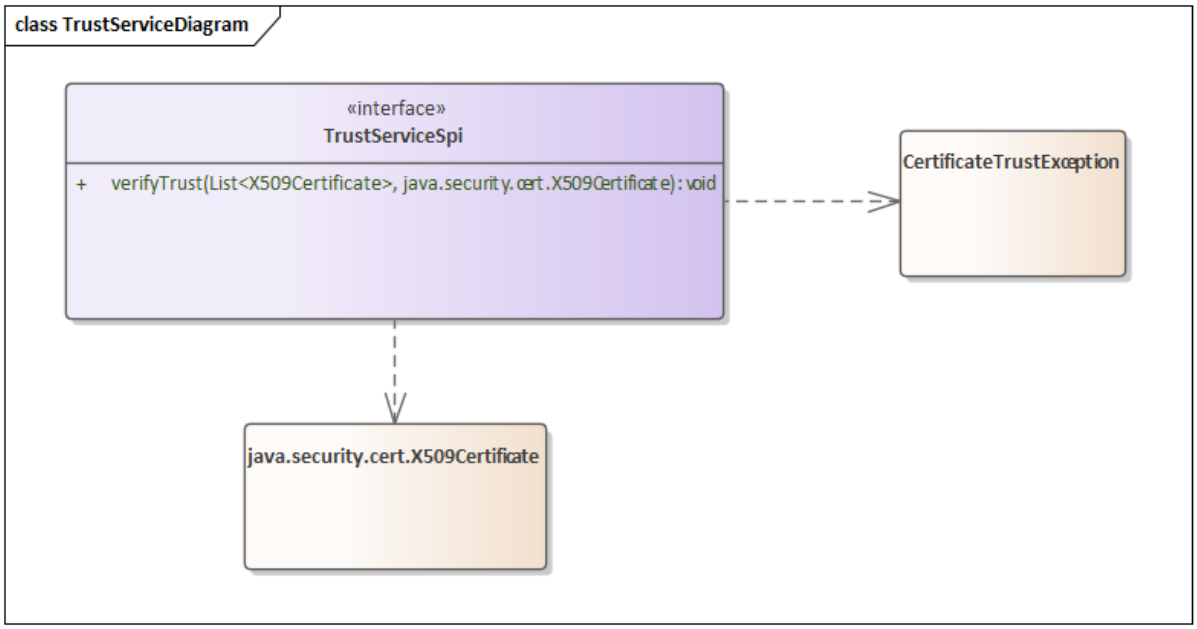

Figure 3- Trust Service Diagram

The following table describes the parameters of the *verifyTrust* method:

Class	Field	Data origin	Description
List<X509Certificate>	See X509Certificate specification referenced document [REF5].	AS4 Message or Domibus public KeyStore store depending on the policy configuration.	List of java.security.cert.X509Certificate containing the signing certificate chain of trust.
X509Certificate	See X509Certificate specification referenced document [REF5].	AS4 Message or Domibus public KeyStore store depending on the policy configuration.	The signing certificate.

The following table describes the parameters of the CertificateTrustException class:

Class	Field	Data origin	Description
CertificateTrustException	N/A	N/A	Runtime exception allowing to control the type of EBMSException triggered by Domibus.
	trustErrorCode	N/A	Enumeration containing distinct trust error code.
	message	N/A	Default exception message.

4.1.1.2. Exception mapping

Throwing a CertificateTrustException from the trust extension gives the flexibility to control the type of EBMS exception returned by Domibus.

EBMSException Error Code	EBMSException Message	TrustException code	Severity	Description
EBMS:0004	T0003: Technical exception	CONFIGURATION	failure	Please use to notify any configuration issue.
EBMS:0004	Unknown error occurred	TRUST_VALIDATION	failure	Please use to notify a certificate trust path validation issue.
EBMS:0001	T0001: Certification validation problem.	OTHER_VALIDATION	failure	Please use to notify any other certificate validation.
EBMS:0004	T0003: Technical exception	UNKNOWN	failure	Please use to notify any other exception.

Any other runtime exception thrown by the authorisation module will be transformed into an EBMS exception with code EBMS:0004 and “T0003:Technical issue” as message.

4.1.2. The getIdentifer method

The domibus.extension.iam.authentication.identifier property is a domain specific property, which provides a way to configure the trust extension to be used. Per default the property value is DEFAULT_AUTHENTICATION_SPI. With the default configuration, Domibus will behave as described in the section 3.1.2.

In order to configure Domibus to use a specific trust extension for the domain, the above property value should be set with the value returned from the getIndentifer() method of the registered custom trust extension. A description of how to register the extension is available in section 5.2.

The property being domain specific, a Domibus configured for multitenancy can choose different trust strategy per domain. For more information on the multitenancy feature of Domibus, please check section “Multitenancy” in Domibus Administration guide [REF2].

The extension developer is free to choose a meaningful name. However, the returned value of the getIdentifer() function should be compliant with the property file format as described in the document REF9.

```
String getIdentifier();
```

4.2. Implementing AuthorizationServiceSpi interface

The interface has 3 methods:

- 2 *authorize* methods
- *getIdentifier* method

4.2.1. The UserMessage and PullRequest authorize methods

Two *authorize* methods are used as trust authorisation for UserMessage and PullRequest respectively.

This method is used to authorize an incoming AS4 UserMessage received by Domibus:

```
void authorize(
        List<X509Certificate> signingCertificateTrustChain,
        X509Certificate signingCertificate,
        UserMessageDTO userMessageDTO,
        UserMessagePmodeData userMessagePmodeData) throws AuthorizationException;
```

This method is used to authorize an incoming AS4 PullRequest received by Domibus:

```
void authorize(List<X509Certificate> signingCertificateTrustChain,
               X509Certificate signingCertificate,
               PullRequestDTO pullRequestDTO,
               PullRequestPmodeData pullRequestPmodeData) throws
AuthorizationException;
```

If the previous methods are executed without exception, the incoming AS4 Message will be authorized and the processing of the message will continue.

If any runtime exception is triggered, the message will be refused and the exception will be transformed into an EBMS exception by Domibus. By throwing an AuthorizationException runtime exception, one can more precisely control the type of EBMS exception thrown by Domibus. An AuthorizationException to EBMSException mapping table is documented in section 4.2.1.2.

4.2.1.1. Data dictionary

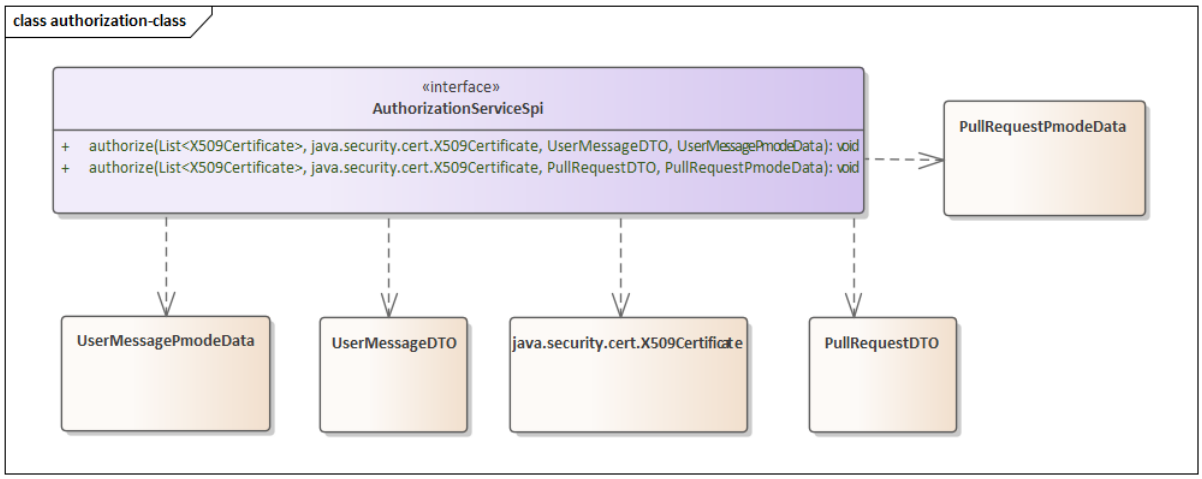

Figure 4 - The authorisation SPI interface and its dependencies

The following table describes the *authorize* method parameters:

Class	Field	Data origin	Description
List<X509Certificate>	Please refer to the X509Certificate specifications document (REF5).	AS4 Message or Domibus public KeyStore depending on the policy configuration.	List of java.security.cert.X509Certificate containing the signing certificate chain of trust.
X509Certificate	Please refer to the X509Certificate specifications document (REF5).	AS4 Message or Domibus public KeyStore depending on the policy configuration.	The signing certificate.
UserMessageDTO	N/A	AS4 UserMessage	The UserMessageDTO class is a model class mapping the AS4 UserMessage model. Please refer to the class model below for detailed information.
UserMessagePmodeData	N/A	PMode	The UserMessagePmodeData class is a model class grouping PMode information related to the received AS4 UserMessage.

Class	Field	Data origin	Description
	serviceName	PMode	PMode service name associated with the received AS4 Message.
	actionName	PMode	PMode action name associated with the received AS4 Message.
	partyName	PMode	PMode sender party name associated with the received AS4 Message.

The UserMessageDTO parameter has the following class structure:

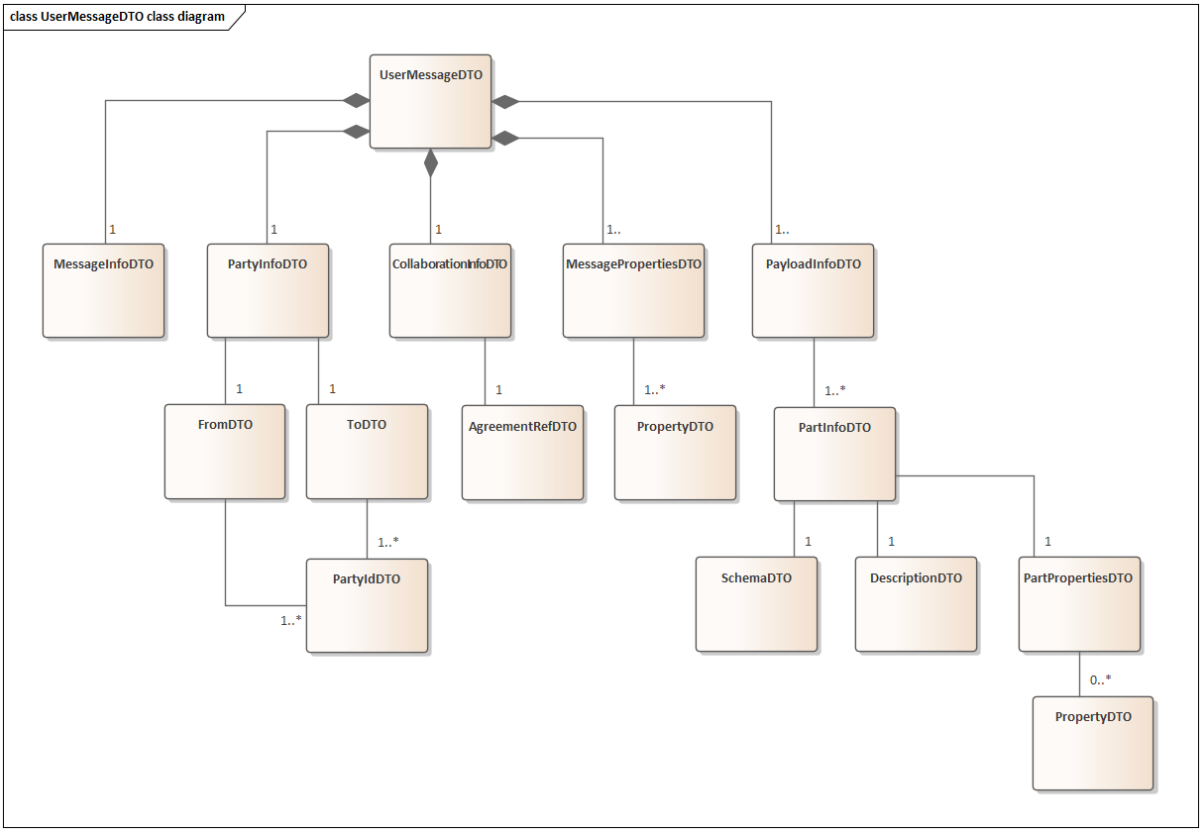

Figure 5 - The UserMessageDTO model

The following table describes the mapping between the class model and the AS4 UserMessage fields. Please review ebMS3 specification referenced document [REF3] for detailed AS4 UserMessage content.

Class	Field	Data origin	Description
UserMessage DTO	N/A	eb:Messaging/eb:UserMessage	ebMS3 specifications.
	mpc	eb:Messaging/eb:UserMessage/@mpc	ebMS3 specifications.
MessageInfoDTO		eb:Messaging/eb:UserMessage/eb:MessageInfo	ebMS3 specifications.
	timestamp	eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:Timestamp:	ebMS3 specifications.
	messageId	eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:MessageId:	ebMS3 specifications.
	refToMessageId	eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:RefToMessageId	ebMS3 specifications.
PartyInfoDTO		eb:Messaging/eb:UserMessage/eb:PartyInfo	ebMS3 specifications.
FromDTO		eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From	ebMS3 specifications.
	role	eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role	ebMS3 specifications.
PartyIdDTO			ebMS3 specifications.
	value	eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId	ebMS3 specifications.
	type	eb:Messaging/eb:UserMessage/eb:PartyInfo	ebMS3

Class	Field	Data origin	Description
		/eb:From/eb:PartyId@type	specifications.
ToDTO		eb:Messaging/eb:UserMessage/eb:PartyInfo /eb:To	ebMS3 specifications.
	role	eb:Messaging/eb:UserMessage/eb:PartyInfo /eb:To /eb:Role	ebMS3 specifications.
PartyIdDTO			ebMS3 specifications.
	value	eb:Messaging/eb:UserMessage/eb:PartyInfo /eb:To/eb:PartyId	ebMS3 specifications.
	type	eb:Messaging/eb:UserMessage/eb:PartyInfo /eb:To/eb:PartyId@type	ebMS3 specifications.
CollaborationInfoDTO		eb:Messaging/eb:UserMessage/eb:CollaborationInfo	ebMS3 specifications.
	conversationId	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:ConversationId	ebMS3 specifications.
	action	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action	ebMS3 specifications.
AgreementRefDTO			
	value	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef	ebMS3 specifications
	type	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef@type	ebMS3 specifications

Class	Field	Data origin	Description
	pmode	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef@pmode	ebMS3 specifications
ServiceDTO			
	value	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service	ebMS3 specifications
	type	eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service@type	ebMS3 specifications
MessagePropertiesDTO		eb:Messaging/eb:UserMessage/eb:MessageProperties	ebMS3 specifications.
PropertyDTO			ebMS3 specifications.
	value	eb:Messaging/eb:UserMessage/eb:MessageProperties/eb:Property	ebMS3 specifications.
	name	eb:Messaging/eb:UserMessage/eb:MessageProperties/eb:Property@name	ebMS3 specifications.
	type	eb:Messaging/eb:UserMessage/eb:MessageProperties/eb:Property@type	ebMS3 specifications
PayloadInfoDTO		eb:Messaging/eb:UserMessage/eb:PayloadInfo	ebMS3 specifications.
PartInfoDTO		eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo	ebMS3 specifications.

Class	Field	Data origin	Description
PartInfoDTO			ebMS3 specifications.
	href	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/@href	ebMS3 specifications.
	inBody	Domibus	Can be used to submit a message on the backend interface in order to send the payload of the AS4 UserMessage within the SOAP envelope body. This way of sending messages is not supported in the AS4 profile and is therefore not recommended.
	mime	Domibus	Contains the payload content type.
SchemaDTO		eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema	ebMS3 specifications.
	location	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema@location	ebMS3 specifications.
	version	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema@version	ebMS3 specifications.
	namespace	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema@namespace	ebMS3 specifications.

Class	Field	Data origin	Description
DescriptionDTO		Domibus	Deprecated, please use PartPropertiesDTO
PartPropertiesDTO		eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties	List containing the properties.
PropertyDTO			ebMS3 specifications.
	value	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties	ebMS3 specifications.
	type	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties@type	ebMS3 specifications.
	name	eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties@name	ebMS3 specifications.

The following table describes the parameters of the AS4 PullRequest *authorize* method:

Class	Field	Data origin	Description
List<X509Certificate>	Check X509Certificate specifications document ([REF5]).	AS4 Message or Domibus trust store depending on the policy configuration.	List of java.security.cert.X509Certificate containing the chain of trust of the signing certificate.
X509Certificate	Check X509Certificate specifications document ([REF5]).	AS4 Message or Domibus trust store depending on the policy configuration.	The signing certificate.
PullRequestDTO		eb:Messaging/eb:SignalMessage/eb:PullRequest	`The PullRequestDTO` class is a model class mapping the AS4 PullRequest model.
	mpc	eb:Messaging/eb:SignalMessage/eb:PullRequest@mpc	
PullRequestPmodeData	mpcName	PMode	PMode MPC name for the received AS4 PullRequest.

The following table describes the parameters of the AuthorizationException method:

Class	Field	Data origin	Description
AuthorizationException	N/A	N/A	Runtime exception allowing to control the type of EBMSException triggered by Domibus.
	authorizationError	N/A	Enumeration containing distinct error code.
	messageId	N/A	Field containing the message id or the refused message.

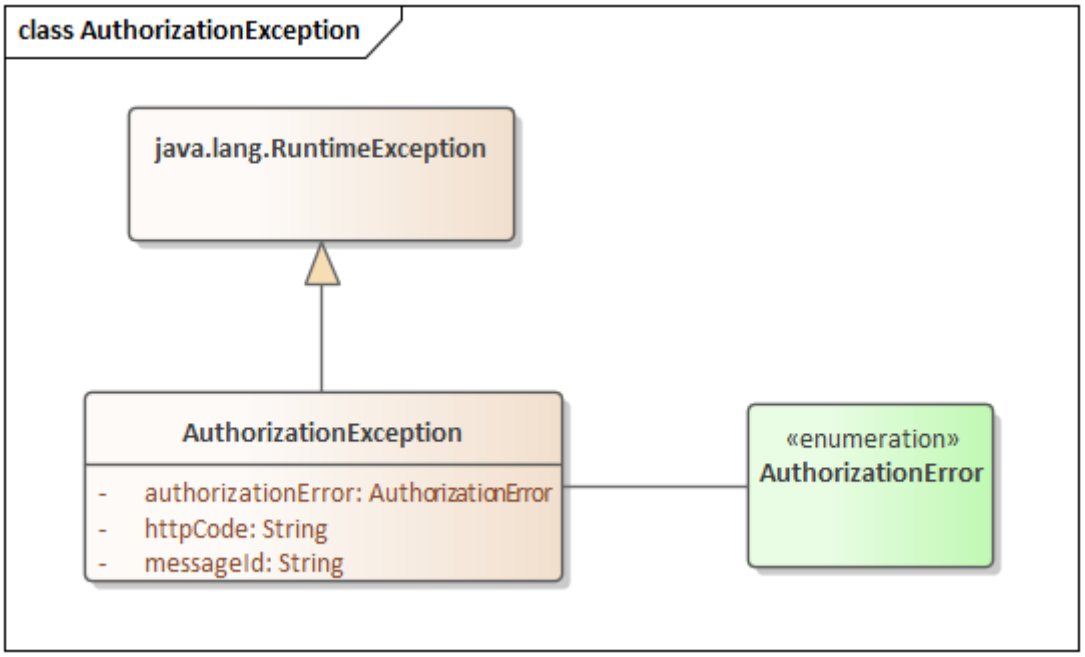

Figure 6 - The AuthorizationException method

4.2.1.2. Exception mapping

Throwing an AuthorizationException from the authorisation extension gives the flexibility to control the type of EBMS exception returned by Domibus. It also ensures that the message id of the failed message is copied within the EBMS exception.

EBMSException Error Code	EBMSException Message	AuthorizationException code	Severity	Description
EBMS:0001	Message contained in the TrustException.	INVALID_FORMAT	failure	Please use to notify any format issue.
EBMS:0004	A0001:Authorization to access the targeted application refused to sender.	AUTHORIZATION_REJECTED	failure	Please use to notify any authorisation rejection.
EBMS:0004	A0003:Technical issue.	AUTHORIZATION_MODULE_CONFIGURATION_ISSUE AUTHORIZATION_	failure	Please use to notify any issue with the authorisation system.

		SYSTEM_DOWN		
EBMS:0004	A0002:Technical issue.	AUTHORIZATION_CONNECTION_REJECTED	failure	Please use to notify any connection issue with the authorisation system.

Any other runtime exception thrown by the authorisation module will be transformed into an EBMS exception with code EBMS:0004 and “A0003:Technical issue” as a message.

4.2.2. The getIdentifer function

The domibus.extension.iam.authorization.identifier property is a domain specific property providing a way to configure the authorisation extension to be used. Per default the property value is DEFAULT_AUTHORIZATION_SPI. With the default configuration Domibus will behave as described in the section 3.2.2.

In order to configure Domibus to use a specific authorisation extension for a domain, the above property value should be set with the value returned from the getIndentifer() method of the extension. A description of how to register the extension is available in section 5.2.

The property being domain specific, a Domibus configured for multitenancy can choose different trust strategy per domain. For more information on the multitenancy feature of Domibus, please check section “Multitenancy” in Domibus Administration guide [REF2].

The extension developer is free to choose a meaningful name. However, the returned value of the getIdentifer() function should be compliant with the property file format as described in the document REF9.

```
String getIdentifier();
```

5. EXTENSION OPERATIONAL INFORMATION

5.1. Building an extension

The recommended way to build an extension is to use Maven (see [REF6]) and the maven-shade-plugin (see [REF8]). By setting the Domibus main POM as the parent POM, the extension benefits from the dependency management of Domibus. The following rule should be respected:

- Before using a library within your custom extension, please verify if the library exists within the Domibus dependencies. If it does, use the same version as the one existing in the dependency management.

- If the needed library exists, set its scope as provided.

The complete pom.xml is detailed in the section 6.1.

5.1.1. Dependency management

The following xml samples highlight the specific aspects of an extension pom configuration:

- Use the main Domibus pom as parent pom. The <**modelVersion** > must reflect the Domibus version that the extension is built for:

```
 <?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>domibus</artifactId>
        <groupId>eu.domibus</groupId>
        <version>4.2-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>your_artifact_id</artifactId>
    <name> your_artifact_name</name>
```

- The minimum set of dependencies to implement an extension is the following:
  => the `${project.version}` corresponds to the version defined in the **<parent>.<version>** tag;
  => any library provided by Domibus dependency management should have a provided **<scope>**

```
<dependencies>
    <!-- Domibus dependencies -->
    <dependency>
        <groupId>eu.domibus</groupId>
        <artifactId>domibus-ext-model</artifactId>
        <version>${project.version}</version>
        <scope>provided</scope>
    </dependency>
```

```
    <dependency>
        <groupId>eu.domibus</groupId>
        <artifactId>domibus-iam-spi</artifactId>
        <version>${project.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>eu.domibus</groupId>
        <artifactId>domibus-logging</artifactId>
        <version>${project.version}</version>
        <scope>provided</scope>
    </dependency>

</dependencies>
```

`domibus-ext-model` contains cross module classes. The DTO objects described above are defined in this library.

`domibus-iam-spi` is the core extension library containing the interfaces described above and their related model.

`domibus-logging` is not mandatory but its usage is recommended because it keeps a uniform logging style with the Domibus core.

The use of other dependencies existing in Domibus dependencies should be configured as follows: => note that the `<scope>` is set as provided and there is no version definition as it is inherited from the Domibus dependency management:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <scope>provided</scope>
</dependency>
```

5.1.2. Logging

The logging service is provided in the domibus-logging module, which is released together with the main Domibus application. More information about domibus-logging module can be found in the Domibus Software Architecture Document (the document can be downloaded at https://ec.europa.eu/digital-building-blocks/wikis/display/DIGITAL/Domibus in the documentation section).

Example of use:

private static final DomibusLogger LOG =
DomibusLoggerFactory.getLogger(BackendWebServiceImpl.class);

5.2. Registering an extension

Domibus has a main configuration folder. Please review the chapter named “Domibus Deployement’” in the Administration guide [REF2] to know how to configure that folder for the various application servers supported by Domibus.

In the following sections, we will refer to that folder as ${domibus.config.location}.

5.2.1. Properties configuration

In order to configure Domibus to use a custom trust extension, please adapt the ${domibus.config.location}/domibus.properties file. Uncomment the property domibus.extension.iam.authentication.identifier and set its value to the value returned by the getIdentifier() method of the TrustServiceSpi interface implementation.

In order to configure Domibus to use a custom authorisation extension, please adapt the ${domibus.config.location}/domibus.properties file. Uncomment the property domibus.extension.iam.authorization.identifier and set its value to the to the value returned by the getIdentifier() method of the AuthorizationServiceSpi interface implementation.

5.2.2. Deployment

In order to install a custom extension for Domibus, please follow the steps below**:**

1. Stop the application server;
2. Copy the custom plugin jar file into the plugins folder: ${domibus.config.location}**/extensions/lib**
3. Make sure that the steps in section 5.2.1 are completed;
4. Start the application server.

6. ANNEX

6.1. POM samples

The following link references the latest production parent pom.xml of Domibus. It contains the dependency management of Domibus:

https://ec.europa.eu/digital-building-blocks/code/projects/EDELIVERY/repos/domibus/browse/Domibus-authentication-dss-extension/pom.xml

The following link references the latest production pom.xml used in the Domibus-authentication-dss-extension module of Domibus:

https://ec.europa.eu/digital-building-blocks/code/projects/EDELIVERY/repos/domibus/browse/Domibus-authentication-dss-extension/pom.xml

7. CONTACT INFORMATION

eDelivery Support Team

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

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