domibusConnectorClient-4.0.0-RELEASE - Administration- and User-Guide

Table of contents

1. Introduction

1.1. Scope and Objective of this document

This document is a technical guide on how the domibusConnectorClient can give support in all its variants is to be integrated, installed and used to connect the domibusConnector web application and your national implementation.

The focus of this document is to give technical guidance on taking the decision on the variant of the domibusConnectorClient to be used to fit the needs of your national environment best. It also describes the functionalities of the domibusConnectorClient and gives an overview of preconditions to be met to integrate the domibusConnectorClient into your environment.

Therefore this guide addresses technical staffs that have knowledge of the national implementation in place and network infrastructure.

This guide document for the domibusConnectorClient only focuses on the client itself.

For more detailed information on the domibusConnector web application, please see the documentation shipped with the domibusConnector-4.0.0-RELEASE.

1.2. The domibusConnector as a web application

Starting with version 4.0-RELEASE, the domibusConnector is on the technical basis of a web application.
This means, that the domibusConnector itself is a "ready-to-use" software component that only needs to be configured, set-up and deployed in a web container.
Once installed and configured properly, the domibusConnector should run on its own, besides some maintenance.
The domibusConnectorClient otherwise is delivered in different variants to support the connection between the domibusConnector web application and your national environment.
It is advised though, that the domibusConnector web application is already properly installed and configured in your environment before starting with the integration of the domibusConnectorClient.

1.3. The domibusConnectorClient

The domibusConnector web application offers different interfaces that can be used to approach its functionalities. Those interfaces can be used directly, if intended. A detailed description of the interfaces offered by the domibusConnector web application can be found in the documentation shipped with the domibusConnector-4.0.0-RELEASE.

To close the missing link between your own implementation of the e-CODEX use cases and the services of the domibusConnector, also a domibusConnectoClient has been implemented to support the connection to the domibusConnector.

The domibusConnectorClient is distributed in the following different variants that are described in detail in this document:

- domibusConnectorClient-4.0.0-RELEASE-Libraries
- domibusConnectorClient-4.0.0-RELEASE-Standalone

1.4.   Architectural overview

The following graphic gives a more detailed insight on the different variants of the domibusConnectorClient:
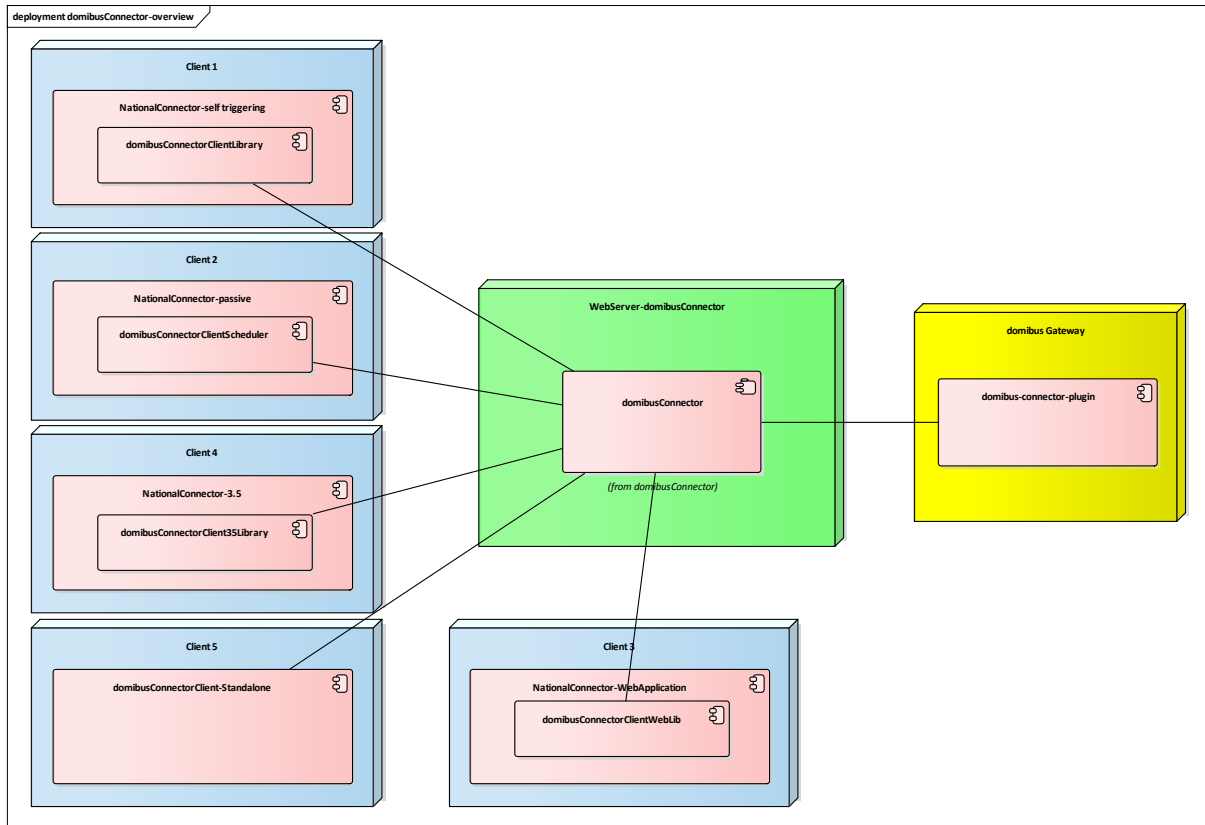
Depending on the specific environment the domibusConnectorClient is used, one of the distributed libraries or the domibusConnectorClient-Standalone can be chosen.

The domibusConnectorClient as a software package is built as a multi-module package. The dependencies the different modules have to each other also describe the functionalities of the libraries:
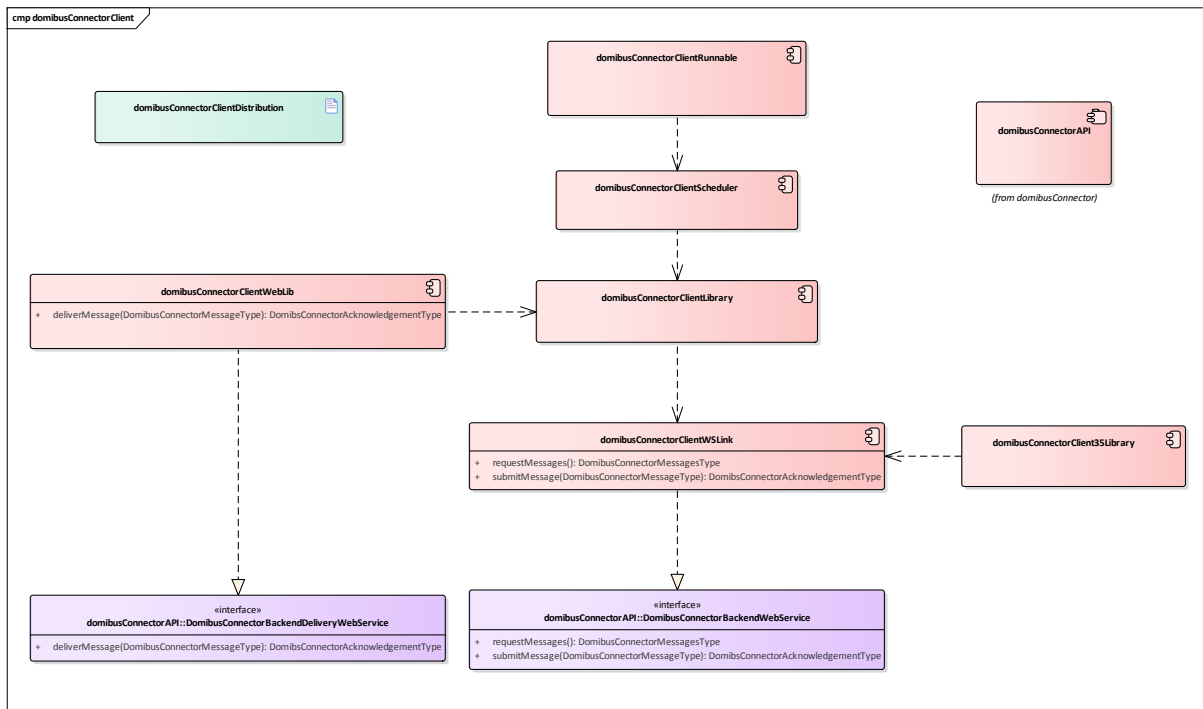
In this component model the "domibusConnectorClientRunnable" represents the domibusConnectorClient-Standalone. A detailed description of the domibusConnectorClient-Standalone can be found in chapter domibusConnectorClient-Standalone.

The different libraries offered are described in detail in chapter domibusConnectorClient-Libraries.

All public interfaces are using the transition model of the domibusConnector. This is a common model representing the messages transported. The transition model is placed in the "domibusConnectorAPI". Its structure is as follows:

class eu.domibus.connector.domain.transition

DomibusConnectorMessagesType *Serializable*
- messages: DomibusConnectorMessageType [1..*]

DomibsConnector AcknowledgementType *Serializable*
- messageId: string
- result: boolean
- resultMessage: string

DomibusConnectorMessageType *Serializable*
- messageAttachments: DomibusConnectorMessageAttachmentType [0..*]
- messageConfirmations: DomibusConnectorMessageConfirmationType [0..*]
- messageContent: DomibusConnectorMessageContentType [0..1]
- messageDetails: DomibusConnectorMessageDetailsType
- messageErrors: DomibusConnectorMessageErrorType [0..*]

DomibusConnectorMessageErrorType *Serializable*
- errorDetails: String
- errorMessage: String
- errorSource: String

DomibusConnectorMessageContentType *Serializable*
- document: DomibusConnectorMessageDocumentType [0..1]
- xmlContent: base64Binary

DomibusConnectorMessageAttachmentType *Serializable*
- attachment: base64Binary
- description: String [0..1]
- identifier: String
- mimeType: String
- name: String

DomibusConnectorMessageDocumentType *Serializable*
- detachedSignature: DomibusConnectorDetachedSignatureType [0..1]
- document: base64Binary
- documentName: String

DomibusConnectorMessageDetailsType *Serializable*
- action: DomibusConnectorActionType
- backendMessageId: String [0..1]
- conversationId: String [0..1]
- finalRecipient: String
- fromParty: DomibusConnectorPartyType
- originalSender: String
- refToMessageId: String [0..1]
- service: DomibusConnectorServiceType
- toParty: DomibusConnectorPartyType

DomibusConnectorMessageConfirmationType *Serializable*
- confirmation: base64Binary
- confirmationType: DomibusConnectorConfirmationType

DomibusConnectorDetachedSignatureType *Serializable*
- detachedSignature: base64Binary
- detachedSignatureName: string
- mimeType: DomibusConnectorDetachedSignatureMimeType

«enumeration» DomibusConnectorConfirmationType
- SUBMISSION_ACCEPTANCE
- SUBMISSION_REJECTION
- RELAY_REMMD_ACCEPTANCE
- RELAY_REMMD_REJECTION
- RELAY_REMMD_FAILURE
- DELIVERY
- NON_DELIVERY
- RETRIEVAL
- NON_RETRIEVAL

«enumeration» DomibusConnectorDetachedSignatureMimeType
- BINARY
- XML
- PKCS7

DomibusConnectorPartyType *Serializable*
- partyId: String
- partyIdType: String [0..1]
- role: String [0..1]

DomibusConnectorServiceType *Serializable*
- service: String
- serviceType: String [0..1]

DomibusConnectorActionType *Serializable*
- action: String

2. Preconditions and technical requirements

This chapter describes what requirements have to be fulfilled to use the functionalities of the domibusConnectorClient. It also lists some technical specifications of the domibusConnectorClient to give a more detailed insight.

2.1. Supported operating systems

The domibusConnectorClient is a software product, that was completely implemented using the JAVA programming language.

As JAVA is by definition a platform independent environment, every operating system with a proper JAVA installation should fit the needs of setting up/ integrate the domibusConnectorClient.

During implementation and testing phase of the domibusConnectorClient, it was tested and installed on the following environments:

- Microsoft Windows 7
- Linux
- IBM AIX

2.2. Java Runtime

As the domibusConnectorClient is a JAVA application, it also requires a proper installation of a Java Runtime to be able to run the software.

The recent version 4.0.0-RELEASE of the domibusConnectorClient was implemented and compiled with an Oracle JDK jdk-8u161. So at least this version or above should be in place to avoid incompatibilities.

2.3. Technical specifications

For your information the main frameworks and technologies the domibusConnectorClient was implemented with is listed here.

- Java 8 (Oracle jdk-8u161)
- Spring framework 4.3.12.RELEASE
- Spring-boot 1.5.8.RELEASE
- Apache CXF 3.2.1
- Apache Maven 3

2.4. The domibusConnectorClient distribution package

To get started, you first need to download and extract the distribution package.

The domibusConnectorClient distribution package is placed on the e-CODEX Nexus repository server at:

https://secure.e-codex.eu/nexus/content/groups/public/eu/domibus/connector/client/domibusConnectorClientDistribution/4.0.0-RELEASE /

To get access to the distribution packages, you need to identify via authentication at the Nexus server.

The following distribution packages can be found there:
- domibusConnectorClient-4.0.0-RELEASE-Libraries
- domibusConnectorClient-4.0.0-RELEASE-Standalone

2.4.1. domibusConnectorClient-4.0.0-RELEASE-Standalone

This client replaces the domibusConnector-Standalone prior to version 4.0-RELEASE. It is a completely self-running application that runs without having any other implementation in place.

The domibusConnectorClient-Standalone interoperates with the file system to receive and send messages from and to the domibusConnector.

It can also be started using a graphical user interface (GUI) to support reading and sending messages. This GUI also supports in setting the configuration.

The contents of the domibusConnectorClient-4.0.0-RELEASE-Standalone distribution package are the following:

File/directory	Description
/bin (directory)	This directory contains the application JAR file "domibusConnectorClientRunnable.jar".
/conf (directory)	This directory contains all the properties that need to be configured. For more details see chapter Configuration properties.
/documentation (directory)	Contains this guide.
/lib (directory)	All Java libraries that are needed to run the domibusConnectorClient-Standalone besides the "domibusConnectorClientRunnable.jar" can be found in this folder.
/messages	There are two subfolders underneath: ■ "outgoing" ■ "incoming" Both are empty folders where the default settings point to with the purpose to store received messages in "incoming" and to search for new messages in "outgoing". For more details see chapter domibusConnectorClient-Standalone
DomibusConnectorClient.bat	This is a startup script to initialize the domibusConnectorClient-Standalone. It is built to run in MS Windows environments to initialize the application properties, set the Java Runtime and build

File/directory	Description
	the classpath. Running this script starts the domibusConnectorClient-Standalone in console mode. No GUI is started.
DomibusConnectorClient.sh	This is a startup script to initialize the domibusConnectorClient-Standalone. It is built to run in Unix-compatible environments to initialize the application properties, set the Java Runtime and build the classpath. Running this script starts the domibusConnectorClient-Standalone in console mode. No GUI is started.
DomibusConnectorClientGUI.bat	The same as "DomibusConnectorClient.bat", but it additionally starts the GUI as well.
DomibusConnectorClientGUI.sh	The same as "DomibusConnectorClient.sh", but it additionally starts the GUI as well.

The functionalities of the domibusConnectorClient-Standalone and how to install and configure it is described in detail in chapter domibusConnectorClient-Standalone.

2.4.2. domibusConnectorClient-4.0.0-RELEASE-Libraries

This is the distribution package holding all other variants of the domibusConnectorClient.

Its contents are the following:

File/directory	Description
/conf (directory)	This directory contains all the properties that need to be configured. For more details see chapter Configuration properties.
/documentation (directory)	Contains this guide.
/libraries/domibusConnectorClientLibrary-4.0.0-RELEASE.jar	This is an integrate able library that can be used to be embedded into an already implemented application. It can also be a basis for new developments as well.
/libraries/domibusConnectorClientScheduler-4.0.0-RELEASE.jar	This library is an extension of the domibusConnectorClientLibrary. It enhances the functionality of the library with time triggered jobs that can be configured to run the functionalities of the library automatically triggered.
/libraries/domibusConnectorClientWebLib-4.0.0-RELEASE.jar	If your national application is a web application that runs inside of a web service container, the domibusConnectorClientWebLib offers the opportunity, to start a web service itself for the delivery of messages from the domibusConnector. The advantage of this variant is, that no jobs need to

	be triggered, as the connection between the domibusConnector and the client work as a push/push web service.
/libraries/domibusConnectorClient35Library-4.0.0-RELEASE.jar	This library only addresses implementers that had previous versions of the domibusConnector framework prior to 4.0-RELEASE in place. It offers access to the functionalities of the new domibusConnectorClient by using the old interfaces that were in place up to version 3.5.1 of the domibusConnector framework. All of those interfaces are marked as deprecated though.
/libraries/domibusConnectorClientWSLink-4.0.0-RELEASE.jar	This is a library all previous libraries depend on. It initializes the web service client that connects to the domibusConnector backend web service.
/libraries/domibusConnectorAPI-4.0.0-RELEASE.jar	This is a library all previous libraries depend on. It holds the interfaces of the domibusConnector.

All the variants listed above as libraries, how they work and how they can be installed/integrated are described in separate chapters of this document.

3. Certificate and Key-Store

To ensure the highest reasonable level of security, the domibusConnector uses web service security on different levels. The connection between the domibusConnectorClient and the domibusConnector underlies the OASIS ws-security standard. A detailed description of the standard can be found here:

https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

Every backend client that connects to the domibusConnector needs to sign and encrypt messages.

For this purpose, the domibusConnectorClient needs a certificate to fulfil the following conditions:

- Authenticate the domibusConnectorClient at the backend of the domibusConnector.
- Sign the body and header of SOAP messages that are sent to the domibusConnector.
- Encrypt the body and header of SOAP messages that are sent to the domibusConnector.
- Decrypt the body and header of SOAP messages received from the domibusConnector.
- Verify the signature of the body and header of SOAP messages received from the domibusConnector.

The following graphic shows the environment specifications of the used certificates and stores:
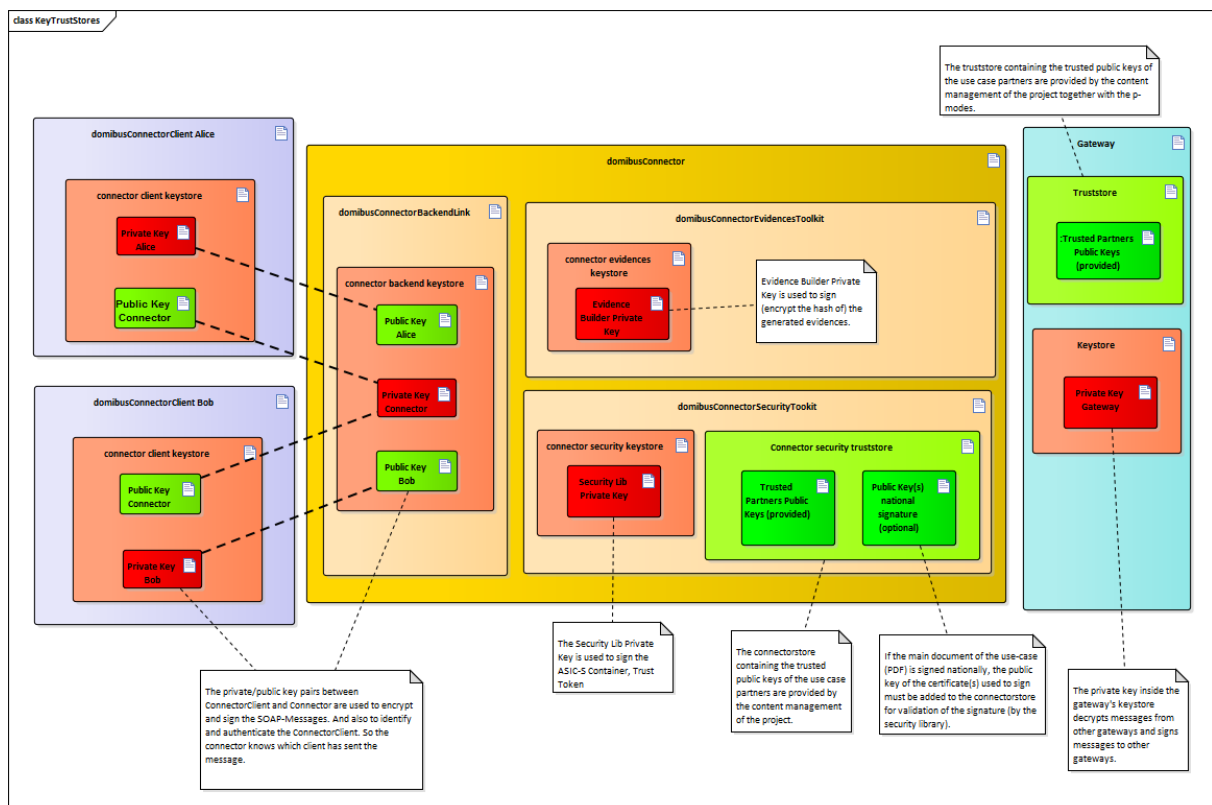
3.1.   Certificate for the domibusConnectorClient

Every client needs a certificate to sign and decrypt the messages sent/received by/to the connector.

The type of certificate used must be compatible to be loaded into a Java-Keystore (JKS). During testphase certificates with RSA algorithm were used. A minimum keysize of 2048 is recommended.

The common name (CN) of the client certificate must match the configured backend name at the domibusConnector.

3.2. Public Key of the domibusConnector

To be able to encrypt messages to the domibusConnector and to verify the signature of messages from the domibusConnector, the public key of the domibusConnector backend keystore is required. Details on the backend keystore and configuration of a backend client can be found in the "domiubsConnecto_InstallationGuide.pdf" distributed together with the domibusConnector installed.

3.3. domibusConnectorClient keystore

Both, the private key of the domibusConnectorClient certificate described above and the public key of the domibusConnector backend certificate configured on the domibusConnector the client should connect to, need to be added to a keystore.

The domibusConnectorClient therefore needs a Java-Keystore containing those keys.

This keystore needs to be configured in the "connector-client.properties" described in chapter Configuration properties.

4. Configuration properties

To give the domibusConnectorClient the missing links about your environment, some properties have to be set in a property file.

Usually this is called "connector-client.properties".

Also, the possibility is given to adopt the logging configuration. This gives the opportunity to control where logs are written at and what to log.

Example properties and an example for logging configuration can be found in the distribution package at "/conf".

The properties in those file are all well described on what is expected there.

The variants on how to include the properties into your domibusConnectorClient is dependent on what variant you have in place.

```
# This property indicates the name of the client configured on the domibusConnector backend.
# It must match the alias and commonName (CN) of the certificate used in the client store.
connector.client.name=

# defines the endpoint address to the installed domibusConnector
connector.backend.ws.address=

# defines the name and role of the gateway used by the domibusConnector.
gateway.name=
gateway.role=

# Properties to define the connector client store to be used.
connector.client.keystore.type=jks
connector.client.keystore.password=
connector.client.keystore.file=
connector.client.key.alias=
connector.client.key.password=

# In case of push/pull connection to the domibusConnector and the domibusConnectorClientScheduler in place
# those properties define the time span between the job triggers.
connector.client.timer.check.incoming.messages.ms=10000
connector.client.timer.check.outgoing.messages.ms=20000


#HTTP and HTTPS proxy settings if necessary
http.proxy.enabled=false
http.proxy.host=
http.proxy.port=
http.proxy.user=
http.proxy.password=

# defines if content mapper module should be used.
connector.client.use.content.mapper=false

# If content mapper module is used, an implementation of the DomibusConnectorContentMapper interface has to be implemented with national content.
# Here the full qualified name of the implementation class which implements DomibusConnectorContentMapper must be given.
# If no content mapper is used (connector.use.content.mappe set false), property can be left empty:
# property connector.content.mapper.implementation.class.name=null
connector.client.content.mapper.implementation.class.name=

##################################################################################################################
############################### Standalone Connector specific properties ###########################################
##################################################################################################################


# full qualified path to the directory where the incoming messages should be stored.
# Needs write privileges and also must be able to create sub directories.
connector.client.messages.incoming.directory=messages/incoming
connector.client.messages.incoming.create-directory=true

# full qualified path to the directory where the outgoing messages will be triggered at.
connector.client.messages.outgoing.directory=messages/outgoing
connector.client.messages.outgoing.create-directory=true

# every message has message properties in a file. This file can be named here. If not set, the default value 'message.properties' is used.
#message-roperties-file-name=null
message-roperties-file-name=message.properties
```

5. domibusConnectorClient-Libraries

The domibusConnectorClient is shipped in different variants to support the needs of your national environment best.

In the domibusConnectorClient-Libraries package all necessary libraries are contained that can be integrated into any Java, or JAR compatible application.

The decision on what library to choose depend on different functionalities they bring with. An overview on the different variants and their placement can be found in chapter Architectural overview.

5.1.1. Using Apache Maven

Since all variants of the domibusConnectorClient are shared over the e-Codex Nexus repository server, we recommend using Apache Maven to import the domibusConnectorClient into your national implementation. The main advantage in that particular case is that all dependent libraries, no matter what variant you are using, are imported automatically without the need to administrate every single library on your own. Additionally, for future releases, it is much easier to upgrade the libraries.

Details on functionality, usage and configuration of Apache Maven repositories can be found at the official documentation page:

https://maven.apache.org/

The data for accessing the e-Codex Nexus repository are:

- Repository Link: https://secure.e-codex.eu/nexus/content/repositories/releases
- Username: ecodex
- Password: eCodeX2012

Those credentials should give readable access to all required artefacts released.

5.1.2. domibusConnectorClientLibrary

The domibusConnectorClientLibrary offers support for the new interfaces between the domibusConnector and the domibusConnectorClient.

5.1.2.1. Installation

Using Maven the dependencies can be resolved with

```
<dependency>
    <groupId>eu.domibus.connector-client</groupId>
    <artifactId>domibusConnectorClientLibrary</artifactId>
    <version>${current.version}</version>
</dependency>
```

Not using Maven, you will need the following distributed JAR files on your classpath:

- domibusConnectorAPI-4.0.0-RELEASE.jar
- domibusConnectorClientLibrary-4.0.0-RELEASE.jar
- domibusConnectorClientWSLink-4.0.0-RELEASE.jar

5.1.2.2. Interfaces

The domibusConnectorClientLibrary offers the following interfaces:
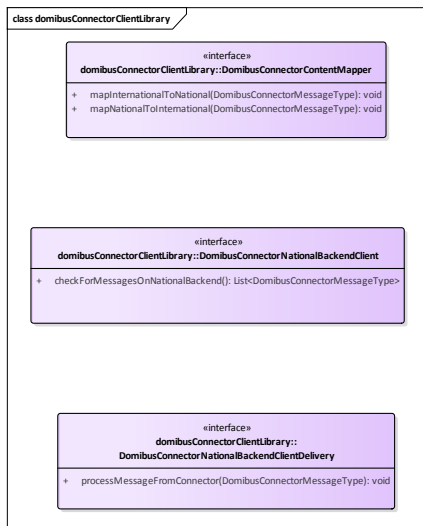
- **DomibusConnectorContentMapper:** This interface is called during processing of messages to map the structured content of a message between common format exchanged between partners and proprietary format only used within the own environment. The target of implementing this interface is to exchange the content of DomibusConnectorMessageType -> DomibusConnectorMessageContentType -> xmlContent with the mapped content. The two methods offered by the interface distinct whether to map from your national format to international format or the other way.
  Within the configuration the implementation of this interface should be set by setting those parameters:

```
# defines if content mapper module should be used.
connector.client.use.content.mapper=false

# If content mapper module is used, an implementation of the DomibusConnectorContentMapper interface has to be implemented with national content.
# Here the full qualified name of the implementation class which implements DomibusConnectorContentMapper must be given.
# If no content mapper is used (connector.use.content.mappe set false), property can be left empty:
# property connector.content.mapper.implementation.class.name=null
connector.client.content.mapper.implementation.class.name=
```

- **DomibusConnectorNationalBackendClient:** This interface must be implemented to tell the domibusConnectorClient where and how to get the messages on your backend to be processed.
- **DomibusConnectorNationalBackendClientDelivery:** This interface must be implemented to tell the domibusConnectorClient where and how to handle processed messages received from the domibusConnector.

The last two interfaces don't need to be configured additionally as they will automatically be recognized when added to the classpath. If no implementation of those interfaces can be found, an ImplementationMissingException is thrown.

5.1.3. domibusConnectorClientScheduler

The domibusConnectorClientScheduler is an extension to the domibusConnectorClientLibrary. Its extension is that there are configurable time-triggered jobs included. They trigger the sending and retrieving of messages. The interfaces which can/must be implemented are the same as with the domibusConnectorClientLibrary.

5.1.3.1. Installation

Using Maven the dependencies can be resolved with

```
<dependency>
    <groupId>eu.domibus.connector-client</groupId>
    <artifactId>domibusConnectorClientScheduler</artifactId>
    <version>${current.version}</version>
</dependency>
```

Not using Maven, you will need the following distributed JAR files on your classpath:

- domibusConnectorAPI-4.0.0-RELEASE.jar
- domibusConnectorClientScheduler-4.0.0-RELEASE.jar
- domibusConnectorClientLibrary-4.0.0-RELEASE.jar
- domibusConnectorClientWSLink-4.0.0-RELEASE.jar

5.1.3.2. Configuration

In addition to the configuration properties that must be set for the domibusConnectorClientLibrary, the following parameters need to be set:

```
# In case of push/pull connection to the domibusConnector and the domibusConnectorClientScheduler in place
# those properties define the time span between the job triggers.
connector.client.timer.check.incoming.messages.ms=10000
connector.client.timer.check.outgoing.messages.ms=20000
```

- connector.client.timer.check.incoming.messages.ms: This property set in milliseconds tells the domibusConnectorClientScheduler what time span to use between checking the domibusConnector for new incoming messages.
- connector.client.timer.check.incoming.messages.ms: This property set in milliseconds tells the domibusConnectorClientScheduler what time span to use between checking over the implemented "DomibusConnectorNationalBackendClient" interfaces if there are new messages to process to the domibusConnector.

5.1.4. domibusConnectorClientWebLib

This library enables the domibusConnector to directly push messages to the client. As the library implements a web service it requires to be run in a web container.

In particular this means that the application using this library must be a web application to be able to start and run the web service.

The outgoing messages have the same interfaces and are handled with the underlying domibusConnectorClientLibrary. The incoming messages though will be pushed from the domibusConnector directly to the domibusConnectorClientWebLib.

5.1.4.1. Installation

Using Maven the dependencies can be resolved with

```
<dependency>
    <groupId>eu.domibus.connector-client</groupId>
    <artifactId>domibusConnectorClientWebLib</artifactId>
    <version>${current.version}</version>
</dependency>
```

Not using Maven, you will need the following distributed JAR files on your classpath:

- domibusConnectorAPI-4.0.0-RELEASE.jar
- domibusConnectorClientWebLib-4.0.0-RELEASE.jar
- domibusConnectorClientLibrary-4.0.0-RELEASE.jar
- domibusConnectorClientWSLink-4.0.0-RELEASE.jar

5.1.4.2. Configuration

On the client side no further configuration parameters have to be set in addition to those that must be set for the domibusConnectorClientLibrary.

Be aware though that there is additional configuration required on the domibusConnector backend config.

Further documentation on backend configuration on the domibusConnector can be found in the "domibusConnector_InstallationGuide" distributed with the domibusConnector.

5.1.5. domibusConnectorClient35Library

The domibusConnector35Library is only for implementers that already have an application in place that used the domibusConnector-framework up to version 3.5.1.

It offers the old interfaces that mostly were replaced.

All those interfaces are marked as deprecated in the code so that a clear distinction between old and new interfaces can be made besides the names.

This library should only be used as a workaround that gives implementers of such applications some time to change their implementation by using the new interfaces.

The domibusConnectorClient35Library will only be offered and distributed for versions 4.0.0-RELEASE and 4.1.0-RELEASE. Future developments from that point onwards will NOT include that library anymore!

5.1.5.1. Installation

Using Maven the dependencies can be resolved with

```
<dependency>
    <groupId>eu.domibus.connector-client</groupId>
    <artifactId>domibusConnectorClient35Library</artifactId>
    <version>${current.version}</version>
</dependency>
```

Not using Maven, you will need the following distributed JAR files on your classpath:

- domibusConnectorAPI-4.0.0-RELEASE.jar
- domibusConnectorClient35Library-4.0.0-RELEASE.jar
- domibusConnectorClientWSLink-4.0.0-RELEASE.jar

6. domibusConnectorClient-Standalone

The domibusConnectorClient-Standalone is a replacement for the domibusConnector-Standalone distributed prior to version 4.0.0-RELEASE.

Its intention is, to give participants, who do not have their own backend applications in place the opportunity to send and receive messages.

Also, if a participant plans to implement a new backend application, the standalone client can be a workaround and example of implementation.

6.1.   Using the domibusConnectorClient-Standalone

This chapter describes the usage of the standalone client without graphical user interface (GUI). It focuses on the interactions with the file system to fetch messages to be sent or store received messages.

The examples and screenshot here were taken from a MS Windows 7 file system. But there can also be other operating systems in place instead.

6.1.1. Sending a message with the domibusConnectorClient

In the "connector-client.properties" described in chapter Configuration properties the folder for outgoing messages can be configured. It is the property "connector.client.messages.outgoing.directory". When no folder is configured the connector client listens at a default folder, which is the „messages/outgoing"- folder within the installation path. To send a message, first a message folder has to be prepared. The message folder needs to contain the following files:
- ◼ The "message.properties" file -> see Structure of the message.properties
- ◼ A PDF file, the main document transported within the message (e.g. a FormA of EPO)
- ◼ An XML file, containing the structured data representing the main document (e.g. the structured data of a FormA of EPO)

The following files can be added optionally:
- ◼ A file containing the detached signature with which the main document has been signed.
- ◼ Additional files to be attached to the message.

Please find here a screenshot with example files:
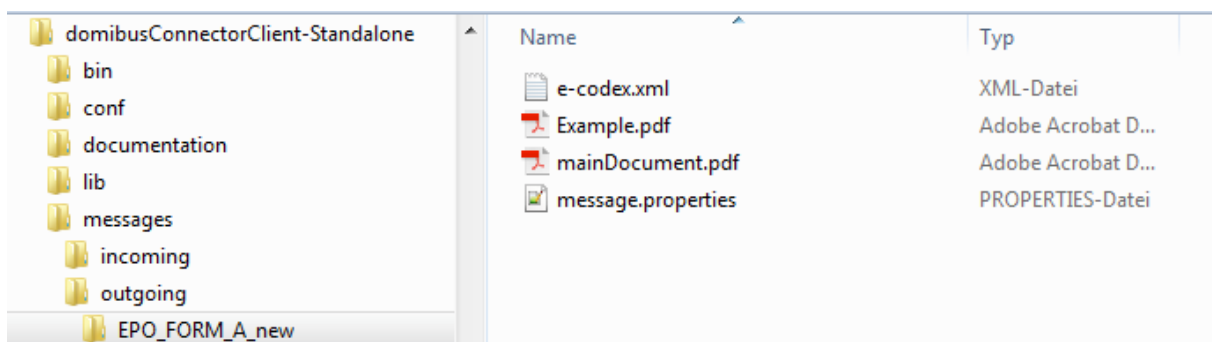
In this example a new folder "EPO_FORM_A_new" in "messages/outgoing" has been created with the following files in it:

- e-codex.xml: The structured data representing the main document
- mainDocument.pdf: The main document of the message
- ExamplePdf: an additional file to be attached to the message
- message.properties: Holding the basic information for the message

In this example the "message.properties" file could be like this:

```
content.pdf.file.name=mainDocument.pdf
content.xml.file.name=e-codex.xml

from.party.id=test_gw_a
from.party.role=GW

to.party.id=test_gw_b
to.party.role=GW

original.sender=test_sender
final.recipient=test_recipient

service=EPO
action=Form_A

conversation.id=
national.message.id=
ebms.message.id=
```

Here the message goes from „test_sender" over the gateway "test_gw_a" (your gateway) to the „test_recipient" at the gateway "test_gw_b" using service "EPO" and action "Form_A".
A detailed description of the possible properties in the "message.properties" file is given in chapter Structure of the message.properties.

The domibusConnectorClient is triggered to process messages when there are folders inside the configured folder for outgoing messages ending with „_message". The folders name before the ending „_message" is not relevant as it will be replaced by a generated folder name. Only when it ends with „_message" the connector will recognize it as a folder containing a new message which has to be processed. In this example the folder "EPO_FORM_A_new" could be renamed into "EPO_FORM_A_message".

When the domibusConnectorClient processed the message, the folder is renamed to the national message id value plus „_sent" postfix. In our example and for the case no national message id is given in the "message.properties", the domibusConnectorClient creates one. This is done by generating a date/time value with the pattern „yyyyMMddhhmmssSSSS" plus the original sender value. In our example this could be „20180427091850133_test_sender". This national message id will then be stored to the message by the domibusConnectorClient. The folder which contains the message will then be renamed with the national message id plus the „_sent" postfix. So in our example the folder will then be „20180427091850133_test_sender_sent".

When evidences to this message are produced or received, the domibusConnectorClient identifies the corresponding message with the national message id and if the folder still exists, it automatically stores the evidences into the message folder. This looks like that:
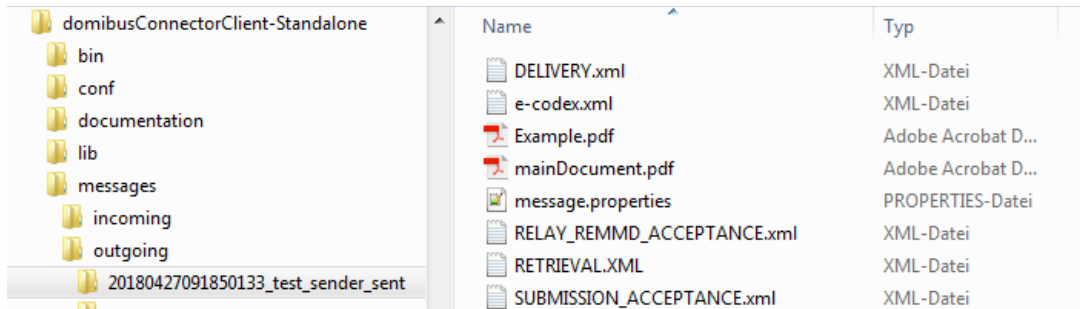
As seen in the screenshot, the evidences are all stored in the message folder. If the folder does not exist anymore, the domibusConnectorClient re-creates it.

Your message has now been sent successfully!

6.1.2. Sending a message with detached Signature

The only difference when sending a message with a detached signature is that the name of the file holding the detached signature has to be set in the "message.properties".
So if we got a message folder like this…
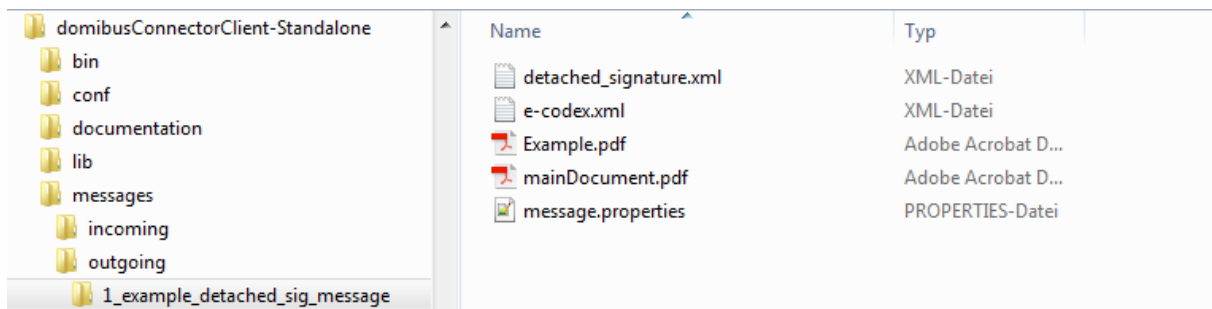
…the "message.properties" must also contain…

```
content.pdf.file.name=mainDocument.pdf
content.xml.file.name=e-codex.xml

detached.signature.file.name=detached_signature.xml

from.party.id=test_gw_a
from.party.role=GW

to.party.id=test_gw_b
to.party.role=GW

original.sender=test_sender
final.recipient=test_recipient

service=EPO
action=Form_A

conversation.id=
national.message.id=
ebms.message.id=
```

The domibusConnectorClient then recognizes the file as a detached signature file and it will be put to the message.

6.1.3. Receiving a message with the domibusConnectorClient

Let's assume that the message of the example above is making its way:

- from the sending domibusConnectorClient
- to the sending domibusConnector
- passing "test_gw_a"
- passing "test_gw_b"
- received by a domibusConnector at the receiving side
- At last be received by a domibusConnectorClient-Standalone.

Then this domibusConnectorClient will receive the message. The message will then be stored into the configured incoming message folder. If no incoming message folder is configured, it will be, by default, the „messages/incoming" sub folder of the installed domibusConnectorClient.

Inside this incoming messages folder the connector creates a new folder for the received message. This message will be named with a generated date/time pattern „yyyyMMddhhmmssSSSS" plus the gateway Id it has been received from. In our example this would be like „20180427092000133_test_gw_a".
In this folder all the message files are stored.
For our above example this would be…
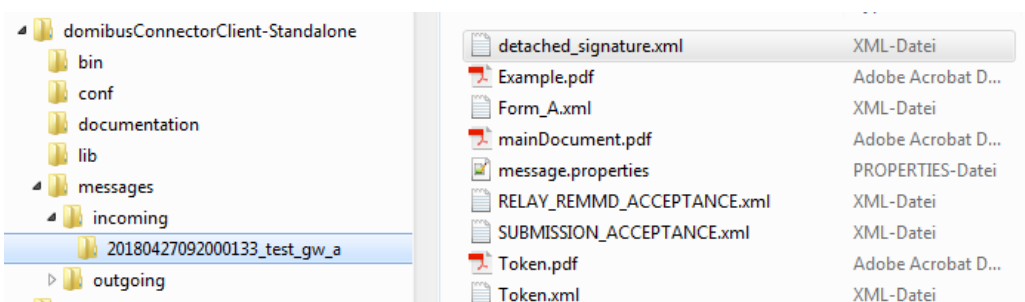
Besides already known files from the example above, those files are new:

- Form_A.xml: This is the structured data file. As no name for this file is transported with the message, the domibusConnectorClient names it after the action attribute the message has been sent along with
- RELAY_REMMD_ACCEPTANCE.xml and SUBMISSION_ACCEPTANCE.xml: these are the two evidences already created for that message at that time.
- Token.pdf: When the sending domibusConnector validates the signature of the main document this Token document is generated. It is the human readable representation of the validation result.
- Token.xml: The structured representation of the validation result.

As the domibusConnectorClient treats received messages which are successfully stored as delivered, it automatically generates and sends the evidences DELIVERY and RETRIEVAL to the sending party.

6.1.4. Structure of the message.properties

Sending a message needs some data not included in the message files themselves. Those are mostly routing information and information on files contained in the message. Therefore a file must be created for every message to be sent containing that data.

This file is built as a properties file with key/value pairs.

It is necessary to provide such a properties file for every message. The name of the message properties file can be configured within the "connector-client.properties". By default it is „message.properties".

Here is an example of an empty "message.properties" file with comments that describe what is expected:

```
# the national message id. This is optional,
# if no national messag id is given, the connector creates one
national.message.id=
# the finalRecipient and originalSender of the message
final.recipient=the final recipients name
original.sender=the original senders name
# The sending gateway party
from.party.id=
from.party.role=GW
# The receiving gateway party
to.party.id=
to.party.role=GW
# The service used
service=EPO
# The action used
action=Form_A
# The name of the PDF file representing the messages main document
content.pdf.file.name=
# The name of the XML file containing the structured data
# representing the message
content.xml.file.name=
# optional - if the main document PDF was signed with a
# detached signature this is provided by this file
detached.signature.file.name=
```

## 6.2.	The domibusConnectorClient GUI

The graphical user interface (GUI) of the domibusConnectorClient is designed to give a visualization and support for the functionalities of the domibusConnectorClient-Standalone.

To start the domibusConnectorClient together with the GUI, the startup-script "DomibusConnectorClientGUI.xxx" should be used, where the "xxx" ending depends on your operating system.

After the startup the domibusConnectorClient starts the GUI automatically by showing the contents of received messages:
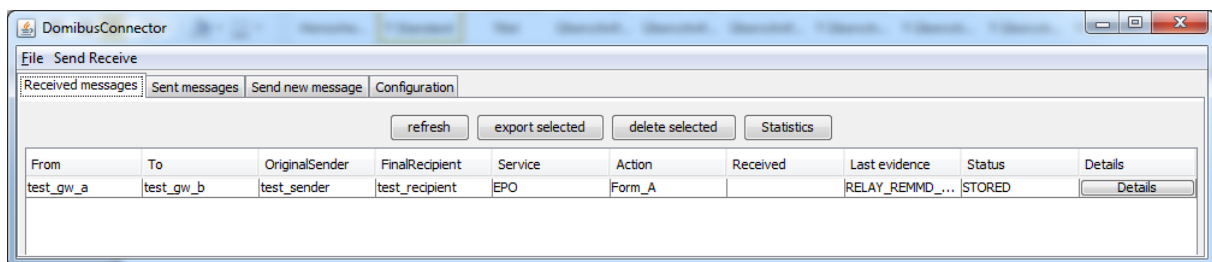
The menu shows a „File" item that only gives the option to shutdown the domibusConnectorClient. Be aware that this not only closes the GUI, but also shuts down the entire domibusConnectorClient as well.

It also has the option "Send Receive" where you can trigger the jobs to whether send or receive messages without having to wait for the timer job to be triggered.

In the main section there are 4 tabs:

- Received messages: It shows a listing of all the messages that can be resolved from the configured "connector.client.messages.incoming.directory" from the "connector-client.properties".
- Sent messages: Shows a listing of all the messages that can be resolved from the configured "connector.client.messages.outgoing.directory" from the "connector-client.properties".
- Send new message: Gives the possibility to create a new message for the domibusConnectorClient to send.
- Configuration: This gives the opportunity to edit all properties set in the "connector-client.properties" via the GUI.

## 6.2.1.	Received messages

If the property "connector.client.messages.incoming.directory" is set properly, the GUI displays every message in that folder. The representation of the messages must stick to the description above in this Guide in chapter Receiving a message with the domibusConnectorClient.
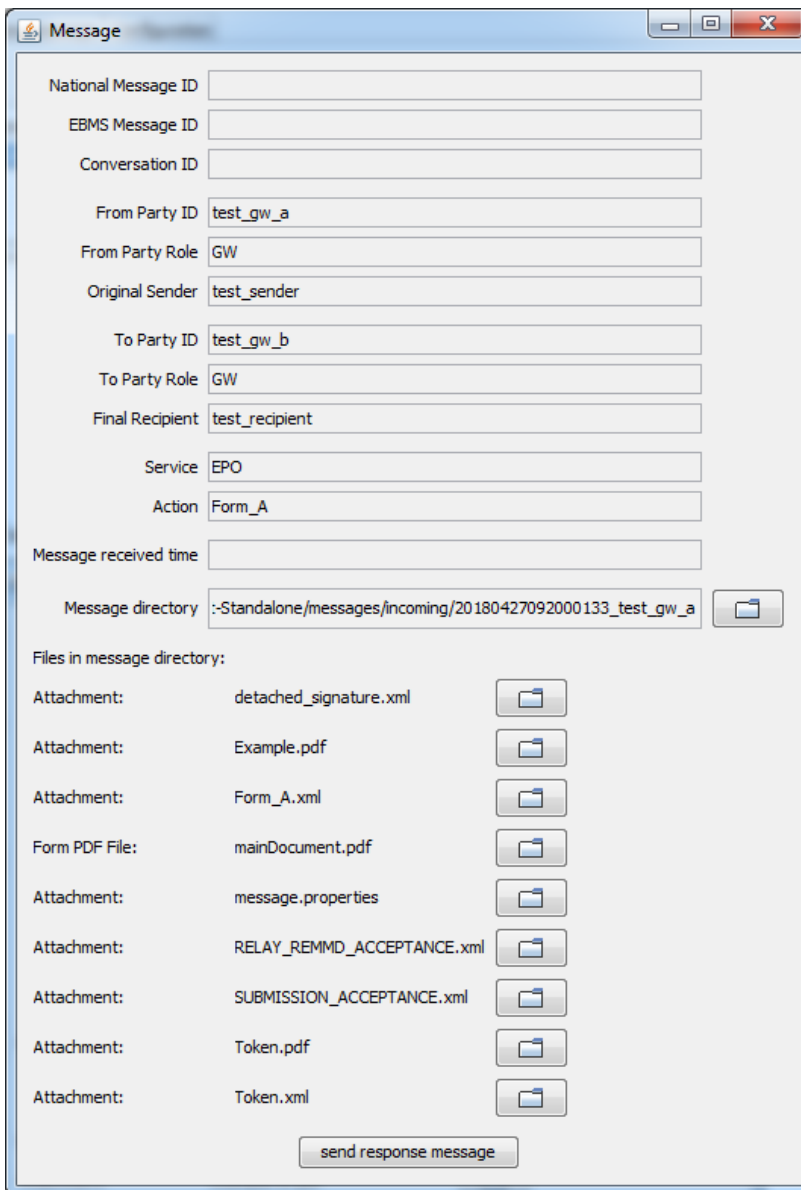
The information displayed in the list comes from the files inside the message directory. If there is no „message.properties" file, or if it is incomplete, no information can be displayed.

The buttons above of the listing give the following options:

- Refresh: simply refreshes the listing if there were any changes on the file system (for example: a new message has been received by the domibusConnectorClient).

- **Export selected:** To use this functionality, one or more messages in the listing must be selected. It gives the possibility to export those selected messages to a place of choice. There is also a possibility to export the messages as zipped archives.
- **Delete selected:** To use this functionality one or more messages in the listing must be selected. It gives the possibility to delete the underlying message folders.
- **Statistics:** This only opens an information message, that statistical information can only be retrieved by the domibusConnector user interface.

Every message can be opened for details with the „Details" button:

It shows more information from the „message.properties" file, gives the option to open the message-directory, lists all files contained in the message folder and opens the files if selected.

There is an additional functionality „send response message". It creates a new message to be sent and uses information from the message details of this message for pre-filling of some fields.

6.2.2. Sent messages

If the property "connector.client.messages.outgoing.directory" is set properly the GUI displays every message in that folder. The representation of the messages must stick to the description above in this Guide in chapter Sending a message with the domibusConnectorClient.

The information displayed in the list comes from the files inside the message directory. If there is no "message.properties" file, or if it is incomplete, no information can be displayed.
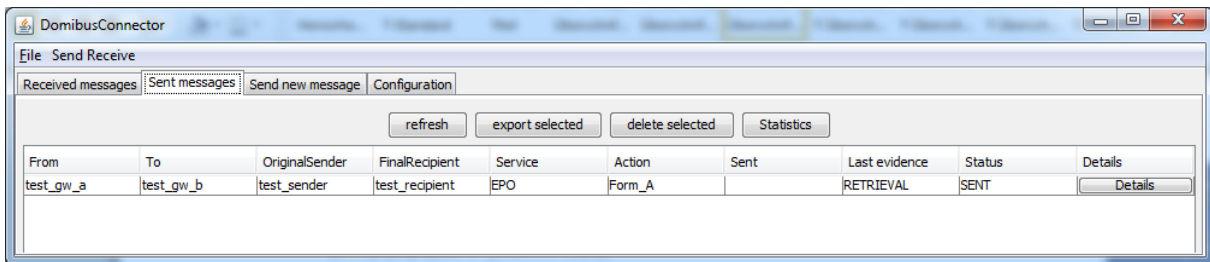
A description of the buttons above of this listing can be found in the previous section of this document.

Every message in the list can be opened with the button „Details":
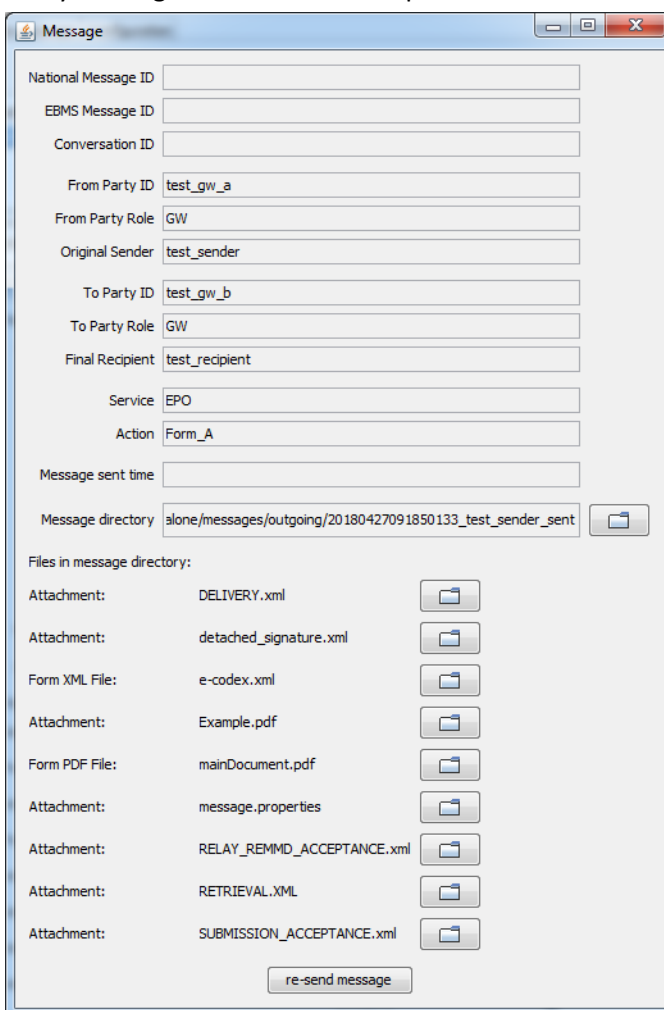
This window is almost the same as that of the details of a received message. But instead of sending a response message it gives the possibility to „re-send message".
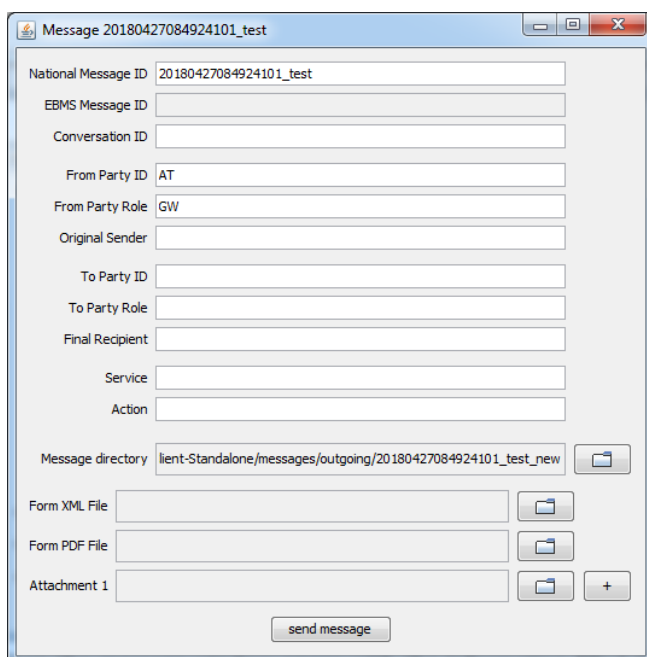
If selected, a new message will be created with all the information and attachments of the previous message.

6.2.3. Send new message

To be able to send a new message, a message folder must be created.
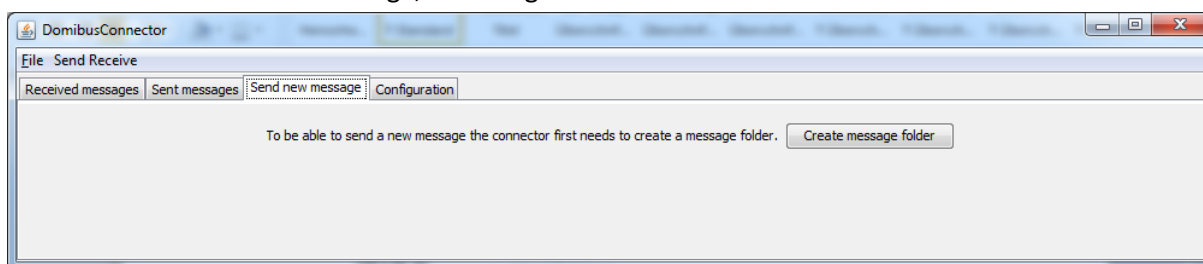
Then the message window opens with the information filled that the domibusConnectorClient can fill. That is a generically generated National Message ID, the From-Party information from the "connector-client.properties" and the Message directory that is already created.

All the other fields must be filled manually.

Also the necessary files for the message must be selected.
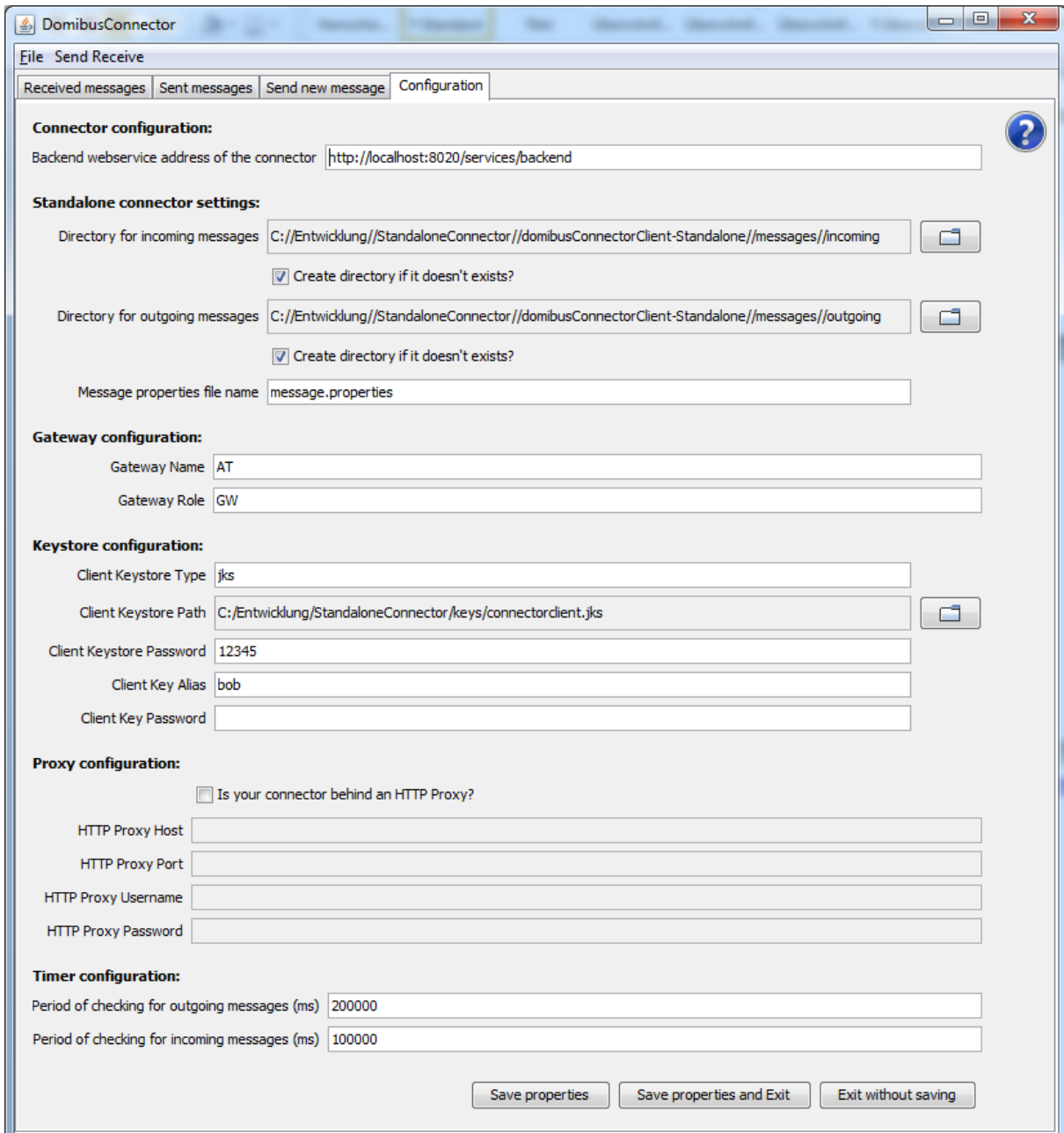
The Form XML File and the Form PDF File must be present and syntactically correct. They cannot be created or edited by the domibusConnectorClient.

After the „send message" button is clicked the message folder is prepared for the domibusConnectorClient to be picked up for sending.

6.2.4. Configuration

The GUI of the domibusConnectorClient-Standalone also has a functionality for editing the "connector-client.properties" described in chapter Configuration properties.

This can be done in section "Configuration":