EUROPEAN COMMISSION

DIGIT
Digital Europe Programme

Access Point

Domibus 5.0.5

Validation Extension Cookbook

Version [1.7]

Status [Final]

Date: 26/05/2023

Document Approver(s):

Approver Name	Role
Bogdan DUMITRIU	Project Manager

Document Reviewers:

Reviewer Name	Role
Chaouki BERRAH	eDelivery Support

Summary of Changes:

Version	Date	Created by	Short Description of Changes
0.1	09/03/2022	Cosmin BACIU	Initial version
1.0	11/03/2022	Chaouki BERRAH	Review and update
1.1	03/06/2022	Caroline AEBY	Domibus 5.0 and Support FMB updated
1.2	26/09/2022	Caroline AEBY	Domibus 5.0.1
1.3	24/11/2022	Caroline AEBY	Domibus 5.0.2
1.4	29/11/2022	Caroline AEBY	Updates
1.5	17/01/2023	Caroline AEBY	Domibus 5.0.3
1.6	31/01/2023	Caroline AEBY	Domibus 5.0.4
1.7	25/05/2023	Caroline AEBY	Domibus 5.0.5

Table of Contents

1. INTRODUCTION

1.1. Purpose

The purpose of this document is to describe the technical specifications of Domibus Validation Extension mechanism. 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 validation options.
- Analysts and developers will find it useful to understand and customize the UserMessage validation implementation.
- Testers can use this document to test the use cases described.

1.4. References

Ref.	Title	Content outline
[REF1]	Domibus Software Architecture Document	This document provides a comprehensive architectural overview of the system, using several 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.

Ref.	Title	Content outline
[REF3]	Maven	Maven portal.
[REF4]	Maven shade plugin	Maven shade plugin portal.

2. OVERVIEW

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

The validation extension mechanism allows the customisation of the AS4 ***UserMessage*** validation prior to be saved in the database and delivered to the plugins. It also allows the validation of the UserMessage and its payloads if requested from a custom plugin.

A validation extension must implement the SPI defined in ***domibus-MSH-spi*** module, which is released together with the main Domibus application. Any changes to previous API versions will be documented in a migration guide.

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

- Domibus Software Architecture Document [REF1]
- Domibus Administration Guide [REF2]

3. EXTENSION FUNCTIONAL INFORMATION

3.1. AS4 UserMessage validation

3.1.1. Overview

In several scenarios, it is useful to validate the received AS4 ***UserMessage*** metadata and associated ***payloads*** before saving them in the database and delivering them to the plugins.

A common example is the scanning of received messages using an antivirus.

3.1.2. Default Domibus behaviour

After an AS4 UserMessage is received via the MSH endpoint, Domibus saves the AS4 metadata in the database and the associated payloads, either in the database or on the file system.

Domibus does not perform any sort of validation in this case and it delivers the received messages to the configured plugin.

3.1.3. Custom behaviour

By creating a custom Validation Extension, a custom validation for the received AS4 UserMessages can be implemented.

The validation of the UserMessage can be performed ***synchronously*** or ***asynchronously***.

Synchronous validation: the UserMessage is rejected (not valid) at the receiving Domibus, acting as C3, a ***Signal error message*** is sent back, as a response, to the sending C2. This is useful when C2 needs to be informed, synchronously, when the validation of the received UserMessage does not pass.

Asynchronous validation: This can be implemented if the validation of the UserMessage and its associated payloads is likely to take a long time. This might be the case when the UserMessage payloads are scanned using an antivirus. In such a scenario, it is no longer possible to synchronously return a Signal with an error to the sending C2 Access Point.

Choosing between synchronous validation and asynchronous validation depends on the business case and the expected processing time of the validation.

4. EXTENSION TECHNICAL INFORMATION

The Validation Extension SPI has in this version one interface:
eu.domibus.core.spi.validation.UserMessageValidatorSpi

4.1. General information

To create a validator, two conditions must be met:

1. The class implementing the interface must be a bean. One way to achieve this is by annotating the class with @Service or @Component annotations.

   @Component

   public class CustomMessageValidation implements UserMessageValidatorSpi { …

2. The class implementing the interface must have a package name starting with “eu.domibus”:

   package eu.domibus.my.custom.validator;

4.2. Implementing *eu.domibus.core.spi.validation.UserMessageValidatorSpi* interface

The interface has 2 methods:

- *validateUserMessage*
- *validatePayload*

4.2.1. *The validateUserMessage method*

The *validateUserMessage* method validates the incoming AS4 UserMessage and its associated payloads.

If the validation passes, Domibus continues its normal processing and saves the incoming UserMessage in the database and informs the configured plugin.

If the validation does not pass, the method must throw a *UserMessageValidatorSpiException* containing the details of the error. In this case, the exception is transformed by Domibus into an EBMS3 exception and a signal with an error is sent as a response to the sending C2.

4.2.1.1. Data dictionary

The following table describes the parameter of the *validateUserMessage* method:

Class	Field	Description

UserMessageDTO	userMessage	AS4 UserMessage metadata and its associated payloads received by C3.

The following table describes the CertificateTrustException class:

Class	Field	Data origin	Description
UserMessageValidatorSpiException	N/A	N/A	Runtime exception raised in case the validation does not pass.

4.2.2. The validatePayload method

The ***validatePayload*** method validates an AS4 UserMessage payload on demand, for instance, from a custom plugin. A typical use case for this method is to scan the payloads using an antivirus solution.

If the validatiàn does not pass, the method must throw a *UserMessageValidatorSpiException* that contains the details of the error.

4.2.2.1. Data dictionary

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

Class	Field	Description
InputStream	payload	AS4 UserMessage payload to be validated on demand.
String	mimeType	The mime type of the payload.

5. VALIDATION EXTENSION OPERATIONAL INFORMATION

5.1. Implementing an extension

The recommended way to implement an extension is to use Maven (see [REF3]) and the maven-shade-plugin (see **Error! Reference source not found.**). By setting the Domibus main POM as the p arent 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 whether 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 referenced in § 6.1.

5.1.1. Maven 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>5.0-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-MSH-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-MSH-spi` is the library containing the interfaces described above and their related model.

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

The use of other dependencies existing in Domibus dependencies management 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 (see [REF1]).

Example of use:

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

5.2. Registering an extension

Domibus has a main configuration folder referenced by *domibus.config.location* property. Please review the chapter named “Domibus Deployment’” in the Administration guide [REF2] to know how to configure the specified 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. Deployment

In order to install a custom validation extension in 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. 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/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)