EUROPEAN COMMISSION

DIRECTORATE-GENERAL FOR INFORMATICS

REST API Pilot: Building and Configuring

Published on 8 September 2022

Document Author(s)

Author Name	Role
Arun Venugopal	Software Developer

Document Reviewer(s)

Reviewer Name	Role
Bogdan Dumitriu	eDelivery Project Officer

Document Approver(s)

Reviewer Name	Role
Bogdan Dumitriu	eDelivery Project Officer

This document was created as a deliverable in the 2020 ISA^2 Innovative Public Services (IPS) action.

Table of Contents

1 Introduction

The main objective of the pilot is to create sample implementations of REST APIs conformant to the ISA² IPS REST API Profile both to provide a reference for future implementations and to validate the profile's fitness for purpose. As such, the aim was to pilot all three sections of the profile: the API Core Profile, the API Documentation and the Messaging API Specification.

As the actual business functionality is only relevant to the extent that it supports the main objective presented above, most such functionality is either mocked or provided in a simplified way, so as to focus the effort on the use of REST APIs.

1.1 Purpose of this document

This document is intended for developers, architects and analysts who would like to use the pilot as a reference for designing and implementing their own REST API(s) conformant to the ISA² IPS REST API Profile.

The document provides instructions for installing and testing the pilot artefacts and an overview of how the REST API Profile is applied in the pilot.

2 Build artefacts

In order to build and run the pilot components, the following tools/applications must be installed:

- **Java JDK 1.8** (tested with Oracle JDK 1.8 and Oracle JDK 11, not tested with OpenJDK versions)
- **Maven 3.6+**
- **docker 19.03+** (needed to build and run demo environment)
- **docker-compose 1.24+** (needed to build and run demo environment)
- **linux bash** (to build demo docker image using example build.sh script, a standard Linux shell/a command interpreter is needed)

2.1 Repository Details

The URL of the git repository containing the REST API Profile is https://github.com/isa2-api4ips/rest-api-profile.

The URL of the git repository containing the pilot implementation is https://github.com/isa2-api4ips/rest-api-oop-pilot.

2.2 How to build the artefacts

To build project execute maven command

```
# checkout pilot code
git clone https://github.com/isa2-api4ips/rest-api-oop-pilot

# enter repository pilot folder
cd rest-api-oop-pilot

# build maven artifacts
mvn clean install
```

Code Block 1 Build Artefacts

2.3 Artefacts

- **./dsd-mock/target/dsd-mock-${project.version}.war**: a mock implementation of the DSD registry.
- **./national-broker/target/national-broker-${project.version}.war**: an example implementation of the National Broker.
- **./ui-client/target/national-broker-ui-client-${project.version}.war**: an example implementation of the UI Client application which uses the services of the National Broker.
- **./isa2-messaging-profile-impl/target/isa2-messaging-profile-impl-${project.version}.jar**: the library implementing the Messaging API Specification of the ISA² IPS REST API Profile.
- **./oop-common/target/oop-common-${project.version}.jar**: project shared utility classes.
- **rest-api-demo:${project.version}**: docker image of the demo of the REST API Pilot.

3 Deployment

3.1 Deploy Artefacts to WildFly Application Server

Download and uncompress the WildFly application server bundle "Jakarta EE Full & Web Distribution":

https://www.wildfly.org/downloads/

To deploy the dsd-mock, national-broker and national-broker-ui-client wars, follow the steps bellow:

```
# ----------------------------
# Init WildFly configuration
# ----------------------------
# Set DSD mock server configuration. (Replace ${isa2-ips-rest-api} with
the pilot project code folder and JBOSS_HOME with the wildfly home folder
or set the environment variables.)
${JBOSS_HOME}/bin/jboss-cli.sh --file=${isa2-ips-rest-api}/dsd-
mock/src/main/setup/config/wildfly/dsd-configuration-H2.cli

# Set National Broker server configuration. (Replace ${isa2-ips-rest-api}
with the pilot project code folder and JBOSS_HOME with the wildfly home
folder or set the environment variables.)
${JBOSS_HOME}/bin/jboss-cli.sh --file=${isa2-ips-rest-api}/national-
broker/src/main/setup/config/wildfly/national-broker-configuration-H2.cli

# ----------------------------
# Copy DSD mock and National Broker resources
# ----------------------------
# Copy DSD mock resources (truststore, keystore)
cp -r ${isa2-ips-rest-api}/dsd-mock/src/main/setup/config/dsd
${JBOSS_HOME}/standalone/data/

# Optionally copy properties to be able to change them if needed
# mkdir ${JBOSS_HOME}/standalone/data/dsd/config
# cp ${isa2-ips-rest-api}/dsd-mock/src/main/resources/dsd.properties
${JBOSS_HOME}/standalone/data/dsd/config/dsd.properties

# Copy National Broker resources (truststore, keystore)
cp -r ${isa2-ips-rest-api/national-broker/src/main/setup/config/national-
broker ${JBOSS_HOME}/standalone/data/

# Optionally copy properties to be able to change them if needed
# mkdir ${JBOSS_HOME}/standalone/data/national-broker/config
# cp ${isa2-ips-rest-api}/national-
broker/src/main/resources/national_broker.properties
${JBOSS_HOME}/standalone/data/dsd/config/national_broker.properties

# ----------------------------
# Copy artefacts
# ----------------------------
cp ${isa2-ips-rest-api}/dsd-mock/target/dsd-mock-1.0-SNAPSHOT.war
${JBOSS_HOME}/standalone/deployments/
cp ${isa2-ips-rest-api}/national-broker/target/national-broker-1.0-
SNAPSHOT.war ${JBOSS_HOME}/standalone/deployments/
cp ${isa2-ips-rest-api}/national-broker-ui-client/target/national-broker-
ui-client-1.0-SNAPSHOT.war ${JBOSS_HOME}/standalone/deployments/

# ----------------------------
# Set property to initialize database. (This step is needed just for the
first time.)
# ----------------------------
SERVER_OPTS="${SERVER_OPTS} -Ddsd.database.create=true -
Ddsd.database.script=/tmp/dsd-h2-data.sql -
Dnationalbroker.database.create=true -
Dnationalbroker.database.script=/tmp/national-broker-h2-data.sql"

# ----------------------------
# Startup WildFly
# ----------------------------
"${JBOSS_HOME}"/bin/standalone.sh --server-config=standalone-full.xml -
b=0.0.0.0 ${SERVER_OPTS} -DJAVA_OPTS="${JAVA_OPTS}"
```

Code Block 2 Deploy and startup demo application

3.2 REST API Pilot Docker Image

The REST API OOP Pilot demo project provides sample Docker build files to facilitate installation, configuration, and environment setup for users. The demo Docker image uses base image **jboss/wildfly:23.0.2.Final** for application server and the h2 database.

Note: the Docker image is not production ready and it should only be used for demonstration and development purposes.

This project offers a sample Dockerfile for the REST API Pilot project with the following artefacts deployed:

- DSD mock component (dsd-mock-${project.version}.war)
- National Broker component (national-broker-${project.version}.war)
- National Broker UI Client component (national-broker-ui-client-${project.version}.war)

3.2.1 How to build and run

To assist users who are not already familiar with Docker to get started, a simple utility shell script is provided: `build.sh`.

Expert users are welcome to directly call `docker build` with their preferred set of parameters.

3.2.1.1 Install Docker Image

To build the Docker image containing the REST API Pilot demo, first the project artefacts must be build. Detailed instructions for how to do that are provided above, in section **Build artefacts**.

Then, from the demo folder start the `build.sh` script:

```
# first build the maven artifacts
# ...

# enter demo folder
cd demo

# build Docker image
./build.sh

# After the build we can see the image domibustest/rest-api-demo in the
local registry
docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
domibustest/rest-api-demo 1.0.0-SNAPSHOT 8b8f33550d75 10 seconds ago
1.23GB
```

Code Block 3 Build Docker image

3.2.1.2 Start Docker Image

Below is the command to start the Docker image:

```
docker run --name rest-api-demo -p 8080:8080  domibustest/rest-api-
demo:1.0.0-SNAPSHOT
```

Code Block 4 Start docker container

To allow changing properties outside the Docker container, the Docker image folders containing configuration files can be bound to persistent folders on localhost:

```
# create DSD mock configuration folder
mkdir dsd-conf

# copy configuration properties example
cp ${isa2-ips-rest-api}/dsd-mock/src/main/resources/dsd.properties ./dsd-
conf/

# create National Broker configuration folder
mkdir broker-conf

# copy configuration properties example
cp ${isa2-ips-rest-api}/national-
broker/src/main/resources/national_broker.properties ./broker-conf/

docker run --name rest-api-demo \
            -p 8080:8080  \
            -v ./dsd-conf:/opt/jboss/wildfly/standalone/data/dsd/config
            -v ./broker-conf:/opt/jboss/wildfly/standalone/data/national-
broker/config
             domibustest/rest-api-demo:1.0.0-SNAPSHOT
```

Code Block 5 Start docker container with external configuration

4 Parameters

The modules dsd-mock and national-broker have their default parameters set in the files:

```
dsd-mock/src/main/resources/dsd.properties
```

and

```
national-broker/src/main/resources/national_broker.properties
```

respectively.

The parameters can be overwritten by setting the java system parameter, for example:

```
     -Ddsd.database.create=true -Ddsd.database.script=/tmp/dsd-h2-
data.sql
```

or by setting the property file to:

```
     file:///${dsd.config.location}/dsd.properties
```

and:

```
     file:///${nationalbroker.config.location}/national_broker.propert
ies.
```

The default values for `*.location` properties are set as:

```
dsd.data.location=${JBOSS_HOME}/standalone/data/dsd/
dsd.config.location=${dsd.data.location}/config
```

and

```
nationalbroker.data.location=${JBOSS_HOME}/standalone/data/national-
broker/

nationalbroker.config.location=${nationalbroker.data.location}/config
```

4.1 DDS Mock Configuration

property	example	description
dsd.data.location	${JBOSS_HOME}/standalone/data/dsd/	Location of DSD mock
dsd.config.location	${dsd.data.location}/config	Location of DSD configuration folder
dsd.storage.location	${dsd.data.location}/storage	Location where DSD responses are stored for pull requests
Database configuration		
The datasource must be defined on the application server. The JDBC datasource is obtained using JNDI for name: `jdbc/DsdMockDS`		
Example (WifdFly configuration):		

```
<datasources>
    <datasource jndi-name="java:/jdbc/DsdMockDS" pool-
name="DsdMockDS" enabled="true" use-java-context="true" use-
ccm="true" statistics-enabled="true">
        <connection-
url>jdbc:h2:${jboss.home.dir}/standalone/data/DsdMockDB;AUTO_SERVER=
TRUE;AUTO_SERVER_PORT=13012;DB_CLOSE_DELAY=-
```

property	example	description

```
1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
        <driver>h2</driver>
        <pool>
            <min-pool-size>5</min-pool-size>
            <initial-pool-size>1</initial-pool-size>
            <max-pool-size>100</max-pool-size>
        </pool>
        <security>
            <user-name>DsdMock</user-name>
            <password>DsdMock</password>
        </security>
        <validation>
            <background-validation>true</background-validation>
        </validation>
    </datasource>
    <drivers>
        <driver name="h2" module="com.h2database.h2">
            <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
        </driver>
    </drivers>
</datasources>
```

property	example	description
dsd.hibernate.connection.driver_class	org.h2.jdbcx.JdbcDataSource	Set hibernate driver class
dsd.hibernate.dialect	org.hibernate.dialect.H2Dialect	Set hibernate SQL dialect
dsd.hibernate.transaction.factory_class	org.hibernate.engine.transaction.internal.jta.CMTTransactionFactory	Set hibernate transaction factory
dsd.hibernate.transaction.jta.platform	org.hibernate.engine.transaction.jta.platform.internal.JBossAppServerJtaPlatform	Set hibernate transaction platform
dsd.hibernate.format_sql	false	Format SQL queries before logging
dsd.hibernate.show_sql	false	Log SQL queries
dsd.database.create	false	(Re)create database at application server startup. Database is recreated from the Hibernate DAO annotations
dsd.database.script		Path to init data SQL script
Messaging REST API		
dsd.messaging.api.type	MESSAGING_API_OBJECT	Instruction how to generate REST API schemas,

property	example	description
		headers and parameters. Allowed options are: • INLINE - definitions are generated inline where they are used • DOCUMENT_COMPONENTS - definitions are generated in the document components #/components/(schema|parameters|headers). • MESSAGING_API_COMPONENTS - definitions are referenced to the published Messaging API document (Property 'dsd.messaging.api.definition.url' is mandatory). • MESSAGING_API_OBJECT - definitions are referenced to the published JSON file for each object (Property 'dsd.messaging.api.definition.url' is mandatory).
dsd.messaging.api.definition.url	https://raw.githubusercontent.com/isa2-api4ips/rest-api-profile/main/messaging-api-specification/components	URL where the components of the Messaging API Specification are located
Keystore/Truststore		
dsd.messaging.security.keystore.location	${dsd.data.location}/keystores/dsd-keystore.p12	The location of the keystore
dsd.messaging.security.keystore.type	pkcs12	The type of the used keystore
dsd.messaging.security.keystore.password	test123	The password used to load the keystore
dsd.messaging.security.signature.key.alias	dsd-mock	The alias from the keystore of the message signing key

property	example	description
dsd.messaging.security.signature.key.password	test123	The private key password
dsd.messaging.security.truststore.location	${dsd.data.location}/keystores/dsd-truststore.p12	The location of the truststore
dsd.messaging.security.truststore.type	pkcs12	Type of the used truststore
dsd.messaging.security.truststore.password	test123	The password used to load the truststore
dsd.messaging.payload.digest.algorithm	SHA256	SHA1,SHA224,SHA256,SHA384,SHA512,SHA3-224,SHA3-256,SHA3-384,SHA3-512,SHAKE-128,SHAKE-256,SHAKE256-512,RIPEMD160,MD2,MD5,WHIRLPOOL
dsd.messaging.security.validation.enable	false	# if parameter is set to false, then security validations are just logged, else error is thrown! # Note: option false enables use of swagger-ui utils which does not calculate digest or sign the messages
OAuth 2		
dsd.oauth2.spring.security.enabled	false	If false, then OAuth authorization is disabled
dsd.oauth2.spring.security.token.url	https://dev-24443841.okta.com/oauth2/aus1096gcr9r537Ut5d7/v1/token	If OAuth is enabled, DSD uses OAuth **client credentials** authorization flow with scope - **SCOPE_ROLE_DSD** required. The authorization server for DSD for the pilot phase is OKTA with URL to obtain access token.
dsd.oauth2.spring.security.jwk.keyset.url	https://dev-24443841.okta.com/oauth2/aus1096gcr9r537Ut5d7/v1/keys	If OAuth is enabled, validation of access tokens submitted is done online using the keyset URL of OKTA

4.2 National Broker Configuration

property	example	description
nationalbroker.data.location	${JBOSS_HOME}/standalone/data/national-broker/	Location of National Broker
nationalbroker.config.location	${nationalbroker.data.location}/config	Location of National Broker configuration folder
nationalbroker.storage.location	${nationalbroker.data.location}/storage	Location where National Broker requests and responses are logged. Requests can be retrieved via the REST API interface.

Database configuration

The datasource must be defined on the application server. The JDBC datasource is obtained using JNDI for name: `jdbc/NationalBrokerDS`

Example (WifdFly configuration):

```
<datasources>
    <datasource jndi-name="java:/jdbc/NationalBrokerDS" pool-name="NationalBrokerDS" enabled="true" use-java-context="true" use-ccm="true" statistics-enabled="true">
        <connection-url>jdbc:h2:${jboss.home.dir}/standalone/data/NationalBrokerDB;AUTO_SERVER=TRUE;AUTO_SERVER_PORT=13011;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
        <driver>h2</driver>
        <pool>
            <min-pool-size>5</min-pool-size>
            <initial-pool-size>1</initial-pool-size>
            <max-pool-size>100</max-pool-size>
        </pool>
        <security>
            <user-name>NatBro</user-name>
            <password>NatBro</password>
        </security>
        <validation>
            <background-validation>true</background-validation>
        </validation>
    </datasource>
    <drivers>
        <driver name="h2" module="com.h2database.h2">
            <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
        </driver>
    </drivers>
</datasources>
```

property	example	description
2 3		
nationalbroker.hibernate.connection.driver_class	org.h2.jdbcx.JdbcDataSource	Set hibernate driver class
nationalbroker.hibernate.dialect	org.hibernate.dialect.H2Dialect	Set hibernate SQL dialect
nationalbroker.hibernate.transaction.factory_class	org.hibernate.engine.transaction.internal.jta.CMTTransactionFactory	Set hibernate transaction factory
nationalbroker.hibernate.transaction.jta.platform	org.hibernate.engine.transaction.jta.platform.internal.JBossAppServerJtaPlatform	Set hibernate transaction platform
nationalbroker.hibernate.format_sql	false	Format SQL queries before logging
nationalbroker.hibernate.show_sql	false	Log SQL queries
nationalbroker.database.create	false	(Re)create database at application server startup. Database is recreated from the Hibernate DAO annotations
nationalbroker.database.script		Path to init data SQL script
Messaging REST API		
nationalbroker.messaging.api.type	MESSAGING_API_OBJECT	Instruction how to generate REST API schemas, headers and parameters. Allowed options are: • INLINE - definitions are generated inline where they are used • DOCUMENT_COMPONENTS - definitions are generated in the document components #/components/(schema|parameters|headers). • MESSAGING_API_COMPONENTS - definitions are referenced to the published

property	example	description
		Messaging API document (Property 'nationalbroker.messaging.api.definition.url' is mandatory). • MESSAGING_API_OBJECT - definitions are referenced to the published JSON file for each object (Property 'nationalbroker.messaging.api.definition.url' is mandatory).
nationalbroker.messaging.api.definition.url	https://github.com/isa2-api4ips/rest-api-profile/tree/main/messaging-api-specification/components	URL where the components of the Messaging API Specification are located
Keystore/Truststore		
nationalbroker.messaging.security.keystore.location	${nationalbroker.data.location}/keystores/nb-keystore.p12	The location of the keystore
nationalbroker.messaging.security.keystore.type	pkcs12	The type of the used keystore
nationalbroker.messaging.security.keystore.password	test123	The password used to load the keystore
nationalbroker.messaging.security.signature.key.alias	national-broker	The alias from the keystore of the message signing key
nationalbroker.messaging.security.signature.key.password	test123	The private key password
nationalbroker.messaging.security.truststore.location	${nationalbroker.data.location}/keystores/nb-truststore.p12	The location of the truststore
nationalbroker.messaging.security.truststore.type	pkcs12	Type of the used truststore
nationalbroker.messaging.security.truststore.password	test123	The password used to load the truststore
nationalbroker.messaging.payload.digest.algorithm	SHA256	SHA1,SHA224,SHA256, SHA384,SHA512,SHA3-224,SHA3-256,SHA3-384,SHA3-512,SHAKE-128,SHAKE-256,SHAKE256-

property	example	description
		512,RIPEMD160,MD2,MD5,WHIRLPOOL
nationalbroker.messaging.security.validation.enable	false	# if parameter is set to false, then security validations are just logged, else error is thrown! # Note: option false enables use of swagger-ui utils which does not calculate digest or sign the messages
DSD integration		
nationalbroker.dsd.url	http://localhost:8080/dsd-mock	DSD URL address
nationalbroker.dsd.webhook.url	http://localhost:8080/national-broker	Webhook URL sent in messaging headers for webhook message exchange patterns
nationalbroker.dsd.finalRecipient	0088:123456789:test	National Broker finalRecipient
nationalbroker.application.originalSender	0088:123456789:national-broker	National Broker originalSender. The value is used when the National Broker is not sending on behalf of the end user (for example when synchronizing request statuses).
OAuth 2		
nationalbroker.oauth2.spring.security.enabled	false	If false, then OAuth authorization is disabled. If enabled, then OAuth **Authorization code with PKCE** flow is expected between UI Client and National Broker.
nationalbroker.oauth2.uiclient.idp.issuer	https://national-broker-poc.eu.auth0.com/	For the pilot phase, the authorization server between UI Client and National Broker is Auth0
nationalbroker.oauth2.uiclient.idp.certificate.alias	national-broker-poc.eu.auth0.com	Alias name of Auth0 public certificate inside truststore of National Broker
nationalbroker.oauth2.accesstoken.expiry.seconds	86400	Expiry time of Opaque access token issued by National Broker

property	example	description
nationalbroker.oauth2.idp.token.expected.audience	http://national-broker-server:8080/national-broker/oauth/token	Audience URL expected in the ID Token issued by authorization server
nationalbroker.oauth2.path.exceptions	/national-broker/messaging-webhook/**	Comma-separated authentication exceptions for webhook
nationalbroker.dsd.oauth.client.token.url	https://dev-24443841.okta.com/oauth2/aus1096gcr9r537Ut5d7/v1/token	If OAuth authorization is enabled, then National Broker to DSD follows OAuth - Client Credentials. The authorization for the pilot is OKTA.
nationalbroker.dsd.oauth.client.jwk.keyset.url	https://dev-24443841.okta.com/oauth2/aus1096gcr9r537Ut5d7/v1/keys	OKTA keyset URL for spring security framework to validate the keys received
nationalbroker.dsd.oauth.client.clientid		
nationalbroker.dsd.oauth.client.clientsecret		
nationalbroker.dsd.oauth.client.scopes	ROLE_DSD	

5 Application URLs

5.1 DSD Mock

- Swagger page: http://localhost:8080/dsd-mock/index.html - Swagger page for testing and exploring the DSD mock REST APIs.
- DSD Organization: http://localhost:8080/dsd-mock/v3/api-docs/organization - DSD Organization JSON REST API document.
- DSD Dataset: http://localhost:8080/dsd-mock/v3/api-docs/dataset - DSD dataset JSON REST API document.
- Messaging API: http://localhost:8080/dsd-mock/v3/api-docs/api - A generic messaging REST API document.
- Database console for inspecting the DSD database (For username/password see the standalone-full.xml database configuration): http://localhost:8080/dsd-mock/h2_console
  To access the database enter the following parameters:
  - Driver Class: `org.h2.Driver`
  - JDBC URL: `jdbc:h2:/opt/jboss/wildfly/standalone/data/DsdMockDB`
  - User Name: `DsdMock`
  - Password: `DsdMock`

5.2 National Broker

- Swagger page: http://localhost:8080/national-broker/swagger-ui.html - Swagger page for testing and exploring the National Broker REST APIs.
- National Broker REST API: http://localhost:8080/national-broker/v3/api-docs - National Broker REST API document.
- Database console for inspecting the DSD database (For username/password see the standalone-full.xml database configuration): http://localhost:13003/national-broker/h2_console
  To access database enter the following parameters:
  - Driver Class: `org.h2.Driver`
  - JDBC URL: `jdbc:h2:/opt/jboss/wildfly/standalone/data/NationalBrokerDB`
  - User Name: `NatBro`
  - Password: `NatBro`

5.3 National Broker UI Client

- National Broker UI Client: http://localhost:8080/ui-client/

6 Contact information

For any questions regarding this document please contact EC-EDELIVERY-SUPPORT@ec.europa.eu.