interstore

D2.2 – LEGACY SYSTEMS PROTOCOL CONVERTER

WP2 - Open-source Interoperability Toolkit

T2.2 – Legacy systems protocol converter

Submission date: 31 Mar 2024

Project Acronym	INTERSTORE
Call	HORIZON-CL5-2022-D3-01
Grant Agreement N°	101096511
Project Start Date	01-01-2023
Project End Date	31-12-2025
Duration	36 months

INFORMATION

Written By	Sunesis (SUN) CyberGrid (CYG)	
Checked by		
Reviewed by	Marcantonio La Franca (ENG), Nithin Manuel(RWTH)	12/03/2024 , 21/03/2024
Approved by	Antonello Monti (RWTH) – Project Coordinator Francesco Guaraldi (ENX)	
Status	Final	

DISSEMINATION LEVEL

CO	Confidential	
CL	Classified	
PU	Public	X

VERSIONS

Date	Version	Comment
9. 1. 2024	0.1	Outline - draft
18. 1. 2024	0.2	Improved outline
24. 1. 2024	0.3	Added architecture
30. 1. 2024	0.4	Added descriptions
7. 2. 2024	0.5	Added configuration
15. 2. 2024	0.6	Added transformation engine description
21. 2. 2024	0.7	Improved transformation engine description
29. 2. 2024	0.8	Added repo description
5. 3. 2024	0.9	Improvements
6. 3. 2024	0.95	Added UML diagram
21. 3. 2024	1.00 final	Final version

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

Acknowledgement

InterSTORE is a EU-funded project that has received funding from the European Union's Horizon Research and Innovation Programme under Grant Agreement N. 101096511.

Disclaimer

The sole responsibility for the content of this report lies with the authors. It does not necessarily reflect the opinion of the European Union. The European Commission is not responsible for any use that may be made of the information contained therein.

While this publication has been prepared with care, the authors and their employers provide no warranty with regards to the content and shall not be liable for any direct, incidental or consequential damages that may result from the use of the information or the data contained therein.

ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
EMS	Energy Management System
GPL	General Public License
HESS	Hybrid Energy Storage Systems
JAR	Java ARchive
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON web token
LPC	Legacy Protocol Converter
LTS	Long Term Support
MIT	Massachusetts Institute of Technology
MQTT	Message Queuing Telemetry Transport
msg	Message
NATS	Neural Autonomic Transport System
QoS	Quality of Service
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
XML	Extensible Markup Language
YAML	Yet Another Markup Language

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

TABLE OF CONTENTS

1 Introduction ... 6
2 Software Architecture and Usage Scenarios ... 7
3 Languages, Technologies and Tools ... 9
4 Data Models and Transformations ... 10
4.1 Mapping definitions ... 11
4.1.1 Transforming from JSON to XML ... 12
5 Configuration ... 15
5.1 Connections ... 15
5.1.1 NATS ... 15
5.1.2 MQTT ... 16
5.1.3 Modbus ... 16
6 Deployment and usage ... 17
6.1.1 JAR ... 17
6.1.2 Docker ... 17
7 Open-source access on GitHub ... 18
8 Docker image available on Docker Hub ... 19
9 Conclusion ... 20
10 REFERENCES ... 21
11 LIST OF FIGURES ... 22

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

1 Introduction

The Legacy Protocol Converter, initially developed within the Horizon Europe Interstore project, acts as a middleware, allowing devices that use different communication protocols to exchange data with EMS systems that use the IEEE2030.5 standard. IEEE 2030.5 is a standard for communications between the smart grid and consumers.

Legacy Protocol Converter supports:

- IEEE2030.5 communication: This is the primary function of the Legacy Protocol Converter. It can handle IEEE2030.5 messages in both JSON and XML formats.
- Next-generation NATS messaging: This is a new messaging protocol that the converter uses to communicate with devices and EMS systems. It is designed to be more efficient and scalable than traditional protocols like REST over HTTP.
- MQTT and Modbus protocols: These are common protocols used by many devices. The Legacy Protocol Converter can translate messages from these protocols into the IEEE2030.5 format for use with EMS systems.

Key features of Legacy Protocol Converter:

- Built-in transformation framework: This framework allows users to define how incoming messages should be transformed into the outgoing IEEE2030.5 format. This is important because different devices and systems may use different message formats.
- Configuration file: The converter uses a configuration file to specify connection details for NATS, MQTT, and Modbus devices. Users can also define transformations within the configuration file.
- Flexibility: The converter can support multiple transformations, each with different incoming and outgoing connections, message formats, and structures. This allows for a high degree of flexibility in how the converter is used.

The Legacy Protocol Converter can be deployed and run using:

- A Docker container.
    - Pre-built Docker images are available on Docker Hub,
    - A custom Docker image can be build.
- Using a Java JAR file and execute it on any computer with OpenJDK Java Runtime Environment.
- Compile and build the project out of source code.
- It is possible to integrate LPC in custom projects by including packages.

LPC can be executed on-premise or in the cloud. It supports a variety of environments, including Kubernetes, Docker and classical virtual machines, as well as bare-metal.

Overall, the Legacy Protocol Converter is a tool for enabling communication between devices and EMS systems that use different communication protocols. It supports the latest IEEE2030.5 standard and provides a flexible and efficient way to translate messages between different protocols.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

2 Software Architecture and Usage Scenarios

Legacy Protocol Converter consists of a transformation module in the package *si.sunesis.interoperability.lpc.transformation,* where classes for handling transformations are defined.

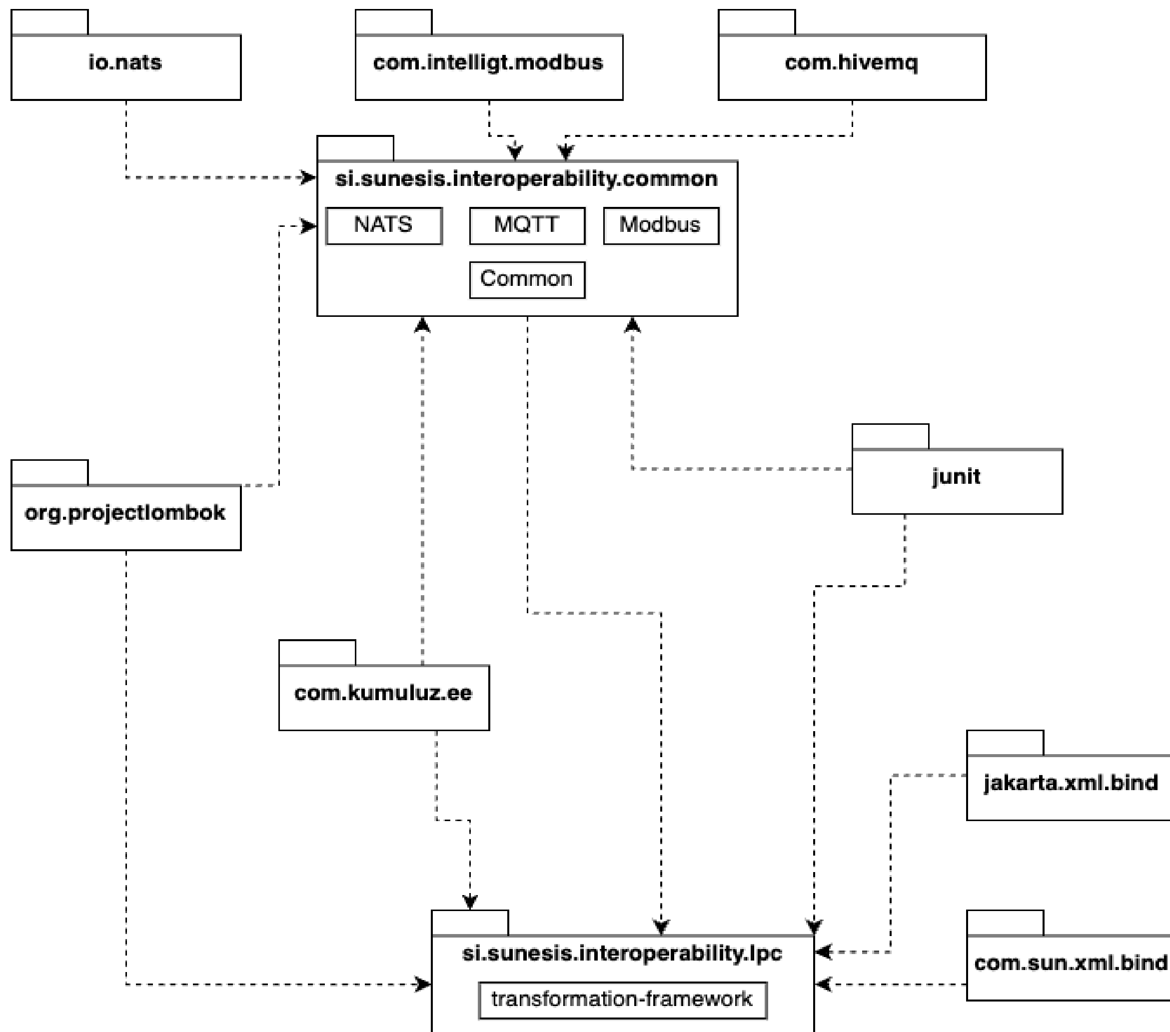

Figure 1: Legacy Protocol Converter package diagram

Figure 1: Legacy Protocol Converter package diagram shows the overall package structure. On the figure you can see the packages, which are used within the interoperability common and legacy protocol converter. For each of these two projects sub-modules are also shown.

For handling configuration files, configuration framework is used, where users can define parameters of connections and transformations. Configuration can be defined in:

- A configuration file using YAML syntax.
- Through environment variables.

The configuration file is handled in the package *si.sunesis.interoperability.lpc.models.* For handling whole configuration file, *YamlTransformationsModel* is used. There is a list of connections *YamlConnectionModel* and a list of transformations *YamlTransformationModel.*

Depending on the type of connections defined in the configuration file, appropriate client is initiated to communicate with devices/servers. Each client has different parameters

necessary to make a connection. For connection, clients that are used are from project interoperability-common with package for:

- Modbus *si.sunesis.interoperability.modbus*,
- MQTT *si.sunesis.interoperability.mqtt* and
- NATS *si.sunesis.interoperability.nats.*

Each client implements the same interface, so the user does not have to know which client is which as they each handle message according to their respective protocol.

For each defined transformation, Legacy Protocol Converter will subscribe using the connections listed in the incoming connections to the incoming topic. When the LPC receives the message, it transforms the message according to the defined structure and mapping in the outgoing message and vice-versa for receiving messages from outgoing connections.

For handling all of the logic behind transformations, class *TransformationHandler* in the package *si.sunesis.interoperability.lpc.transformations.transformation* is built with this purpose. It handles subscribing and publishing messages, requesting messages from Modbus and transforming between structures and formats.

In the package *si.sunesis.interoperability.lpc.transformations.mappers* are two different mapper classes, one for XML messages and one for JSON messages. That is because they each handle mappings depending on the format using respective libraries.

Examples can be seen in the chapter 4.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

3 Languages, Technologies and Tools

Legacy Protocol Converter is developed in Java. Java 17 or higher is required for Legacy Protocol Converter, with plans to upgrade to Java 21 in the future releases.

Legacy Protocol Converter is developed using microservice architecture. The source code is based on Java Microprofile specification. KumuluzEE framework is used for the project as it is open-source lightweight framework for microservices and offers support for cloud-native applications.

KumuluzEE Maven dependencies that are used, are:

- KumuluzEE BOM, version 4.1.0
- KumuluzEE Core, version 4.1.0
- KumuluzEE Servlet Jetty, version 4.1.0
- KumuluzEE CDI Weld, version 4.1.0
- KumuluzEE Logs, version 1.4.6

The Legacy Protocol Converter is also using the common package from the *si.sunesis.interoperability.common* group for managing connections with NATS, MQTT and Modbus.

Maven dependencies in the common package are:

- KumuluzEE BOM, version 4.1.0
- KumuluzEE MicroProfile 3.3, version 4.1.0
- KumuluzEE Logs, version 1.4.6
- Lombok, version 1.8.30
- NATS – Java Client, version 2.17.3
- HiveMQ – MQTT Client, version 1.3.3
- JLibModbus, version

Other Maven dependencies are:

- Lombok, version 1.8.30
- Jakarta XML Bind API, version 4.0.0
- Jaxb-impl, version 4.0.3

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

4 Data Models and Transformations

Data Models are handled depending on format of the message. XML messages are handled using Java's built-in methods in the package java.xml. JSON messages are handled with the help of Jackson JSON parser.

Code snippet for generating JSON:

```
JsonNode jsonNode = new ObjectMapper().readTree(message);
```

Code snippet for generating XML:

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();

factory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
factory.setFeature("http://apache.org/xml/features/disallow-
doctype-decl", true);
factory.setXIncludeAware(false);
factory.setNamespaceAware(false);

DocumentBuilder builder = factory.newDocumentBuilder();
InputSource inputSource = new InputSource(new
java.io.StringReader(xmlString));
Document document = builder.parse(message);
```

Each transformation generates XML or JSON message depending on the specified format in the configuration of the transformation.

```
lpc:
  connections:
    -
    -
  transformations:
    - name: string
      description: string
      connections:
        incoming-connection:
          -
          -
        incoming-topic: string
        incoming-format: XML/JSON
        modbus-function-code: integer
        modbus-device-id: integer
        outgoing-connection:
          -
          -
        outgoing-topic: string
        outgoing-format: XML/JSON
      to-outgoing: string
      to-incoming: string
      or
      to-incoming:
        modbus-registers:
          - register-address: integer
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
            path: string
            type: int8/int16/int32/int64/float32/float64
            pattern: string
            values: array
```

Options:

- name: Short name of the transformation
- description: Description of the transformation
- connections.incoming-connection: List of connection names, this connections will be used for sending/receiving the data from clients. If using Modbus, only Modbus connections must be listed here. Must match the name in the connections.
- connections.incoming-topic: On which topic MQTT/NATS client will send the incoming messages or listen for messages from devices. If using Modbus, this must be omitted.
- connections.incoming-format: Format of the incoming messages.
- connections.modbus-function-code: If using Modbus, it tells the LPC which function code to use when requesting/writing data.
- connections.modbus-device-id: If using Modbus, it tells the LPC to which device it will send the data.
- connections.outgoing-connection: List of connection names, this connections will be used for sending/receiving the data from server. Must match the name in the connections.
- connections.outgoing-topic: On which topic MQTT/NATS client will send the outgoing messages or listen for message from server.
- connections.outgoing-format: Format of the outgoing messages.
- to-outgoing: Structure of the outgoing message with defined mappings.
- to-incoming: Structure of the incoming message with defined mappings. If this using Modbus, this must be omitted.
- to-incoming.modbus-registers: List of definitions of modbus registers used for writing/reading the data.

If to-outgoing message structure is present, then LPC will listen/request messages and send the transformed message to the outgoing-topic. When using to-outgoing in combination with Modbus as connections.incoming-connection.

If to-incoming message structure is present, then LPC will listen/request messages and send the transformed message to the incoming-topic, or it will request/write data to appropriate Modbus registers.

4.1 Mapping definitions

Transforming messages from one structure to another is done with the help of a mapper. Each mapper has the following variables that are configurable:

- Type
- Path
- Pattern
- Values

Type specifies the type to which value must be converted to. Possible values are: *integer*, *float*, *double*, *date*, *datetime* and *string*. When using Modbus, number of bits is required, so *integer8* (or *int8*), *int16*, *int32*, *int64* and also *float32* and *float64*; *int64* is converted to long

and *float64* is converted to double. *date* and *datetime* are converted to long representing the number of milliseconds since January 1, 1970, 00:00:00 GMT.

Path specifies path to the value that will be used in new message structure. For XML/JSON this is done using XPath or JSON Pointer (e.g. /OutgoingEvent/currentStatus). For Modbus messages, register address must be provided.

Pattern is needed only when type is *datetime* or *date* as it specifies the format of the provided temporal value at path.

Values is needed only when value must be converted to the index of an array or index to some value from the array.

For easier explanation of options, here are the examples.

4.1.1 Transforming from JSON to XML

This examples showcases transformation of IncomingEvent in JSON format to IEEE2030.5 Event in XML format.

JSON IncomingEvent


```
{
  "datetime": "28-08-2023 12:00:35",
  "status": "active",
  "start": "28-08-2023",
  "duration": 900
}
```


XML IEEE2030.5 Event


```
<Event>
    <creationTime>1702909917932</creationTime>
    <EventStatus>
        <currentStatus>1</currentStatus>
        <dateTime>1693216835000</dateTime>
        <potentiallySuperseded>
            False
        </potentiallySuperseded>
    </EventStatus>
    <interval>
        <duration>900</duration>
        <start>1693216835000</start>
    </interval>
</Event>
```


Example of the configuration for this transformation:

```
lpc:
  connections:
    - name: NATS-connection
      type: NATS
      host: nats://localhost
      port: 4222
      reconnect: true
    - name: MQTT-connection
      type: MQTT
      ssl:
        default: true
      host: 123.mqtt-client.cloud
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
      port: 8883
      username: lpc-user
      password: lpc-password
  transformations:
    - name: JSON IncomingEvent to XML IEEE2030.5 Event
      description: Example showing transformation of messages from
JSON to XML
      connections:
        incoming-connection:
          - MQTT-connection
        incoming-topic: topic1
        incoming-format: JSON
        outgoing-connection:
          - NATS-connection
        outgoing-topic: event/myevent
        outgoing-format: XML
      to-outgoing:
        '<Event>
          <creationTime>$timestamp</creationTime>
          <EventStatus>
            <currentStatus>
              <lpc:mapping>
                <path type="integer">/status</path>
                <values>["scheduled", "active", "cancelled",
"cancelled_with_r", "superseded"]</values>
              </lpc:mapping>
            </currentStatus>
            <dateTime>
              <lpc:mapping>
                <path type="datetime">datetime</path>
                <pattern>dd-MM-yyyy HH:mm:ss</pattern>
              </lpc:mapping>
            </dateTime>
            <potentiallySuperseded>false</potentiallySuperseded>
          </EventStatus>
          <interval>
            <duration>
              <lpc:mapping>
                <path type="integer">duration</path>
              </lpc:mapping>
            </duration>
            <start>
              <lpc:mapping>
                <path type="date">/start</path>
                <pattern>dd-MM-yyyy</pattern>
              </lpc:mapping>
            </start>
          </interval>
        </Event>
        '
```

In this transformation it is specified that the MQTT-connection will subscribe to topic *topic1* and LPC will transform the message to XML structure, and it will send the transformed message using NATS-connection to topic *event/myevent*.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

Mapping is done with the help of XML tag *<lpc:mapping>*.

JSON IncomingEvent is the input to the transformation and the result is XML IEEE2030.5 Event.

For setting the value of *OutgoingEvent/currentStatus*, we are converting value *"active"* to integer *1*. Because the *IncomingEvent/status* is a string, and the option *values* is provided, this means it will map to the integer based on which index is the value of *IncomingEvent/status*. So *"active"* maps to *1*.

For setting the value of *OutgoingEvent/datetime*, conversion is done from *datetime* to *long*.

Because *type* is *datetime*, pattern of the incoming value is needed, so LPC know how to parse the value, hence *dd-MM-yyyy HH:mm:ss*. Also note that the leading / is not needed.

Same applies for setting the value of *OutgoingEvent/interval/start*, but here type is *date*.

For setting the value of *OutgoingEvent/interval/duration*, just the *type* needs to be provided, because mapping is done 1 on 1.

There is also reserved keyword *$timestamp* which is used to set the value of OutgoingEvent/creationTime to the current time in milliseconds.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

5 Configuration

General format of the configuration file is following:

```
lpc:
  connections:
    -
    -
  transformations:
    -
    -
```

Connections and transformation must be defined by the user.

5.1 Connections

Each incoming/outgoing connection must be configured in the list of connections, so the LPC knows how to connect accordingly.

Possible options for each connection are following:

```
lpc:
  connections:
    - name: string
      type: NATS/MQTT/Modbus
      host: string
      port: integer
      ssl:
        default: true/false
      username: string
      password: string
      reconnect: true/false
      device: string
      baud-rate: integer
      data-bits: integer
      parity: none/even/odd/space/mark
      stop-bits: integer
  transformations:
    -...
```

Name and type are required keys.

5.1.1 NATS

Currently supported parameters for connection with NATS are **host**, **port**, **username**, **password** and **reconnect.**

Example of configuration for NATS:

```
lpc:
  connections:
    - name: NATS-connection
      type: NATS
      host: nats://localhost
      port: 4222
      username: userTest
      password: testUser
      reconnect: true
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

```
  transformations:
    -...
```

5.1.2 MQTT

Currently supported parameters for connection with NATS are **host**, **port**, **ssl**, **username**, **password** and **reconnect.**

Example of configuration for MQTT:

```
lpc:
  connections:
    - name: MQTT-connection
      type: MQTT
      host: localhost
      port: 8883
      ssl:
        default: true
      username: userTest
      password: testUser
      reconnect: false
  transformations:
    -...
```

5.1.3 Modbus

Configuration for Modbus depends on connection type, LPC supports serial connection or TCP connection. For TCP connection parameters **host** and **port** are required. For serial connection parameter **device** is required, optional parameters are **baud-rate**, **data-bits**, **parity** and **stop-bits**. Modbus connection cannot be used as an outgoing connection, because it is a request-reply protocol.

Example of configuration for serial Modbus connection:

```
lpc:
  connections:
    - name: Modbus-connection
      type: Modbus
      device: /dev/ttymxc2
      baud-rate: 115200
      data-bits: 8
      parity: none
  transformations:
    -...
```

Example of configuration for TCP Modbus connection:

```
lpc:
  connections:
    - name: Modbus-connection
      type: Modbus
      host: localhost
      port: 502
  transformations:
    -...
```

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

6 Deployment and usage

For building the code, Maven is needed as this is a Maven project. There are two ways this project can be deployed, using JAR, or using Docker container.

When deploying the application, users must provide path to the configuration folder. Default configuration path is: *./conf*

6.1.1 JAR

JAR is generated by running Maven command:

mvn clean package

This will build all modules and once completed the build archives will be in the modules respected target folder.

Legacy Protocol Converter can be started from JAR using Uber-Jar:

java -jar -DCONFIGURATION=/path/to/config target/ ${project.build.finalName}.jar

6.1.2 Docker

Docker container is built using the command:

docker build -t lpc:latest .

and then started using the command:

docker run -v /path/to/config:/app/conf lpc:latest

When starting the application using Docker, users must mount the configuration folder, so that the Legacy Protocol Converter can use the configurations defined by the user.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

7 Open-source access on GitHub

The source code for the Client/Server, Legacy protocol converter and testing procedures will be available on GitHub. We have created a repository: https://github.com/Horizont-Europe-Interstore

Within the repository there are currently four projects:

- Interoperable client/server for distributed energy storage
- Legacy system protocol converter
- Testing procedures and software tools
- Interoperable data spaces framework

The LPC source code can be found here: https://github.com/Horizont-Europe-Interstore/Legacy-Protocol-Converter

Within the repository, detailed description of the project, source code and build and run instructions is given.

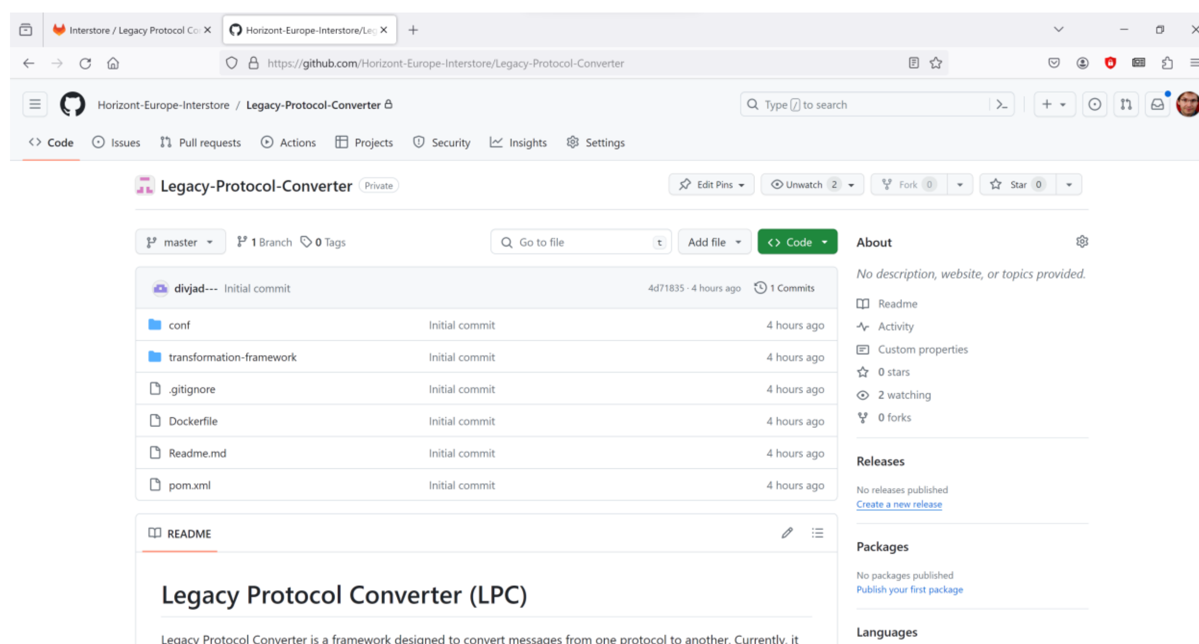

Figure 2: Open-source repository on GitHub.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

8 Docker image available on Docker Hub

A pre-built version of Docker image is publicly available on Docker Hub. This enables easy access to Docker images and their updates.

Docker image is available here: https://hub.docker.com/r/interstore/legacy-protocol-converter

Within the Docker Hub, detailed description and instructions for using Docker image are posted.

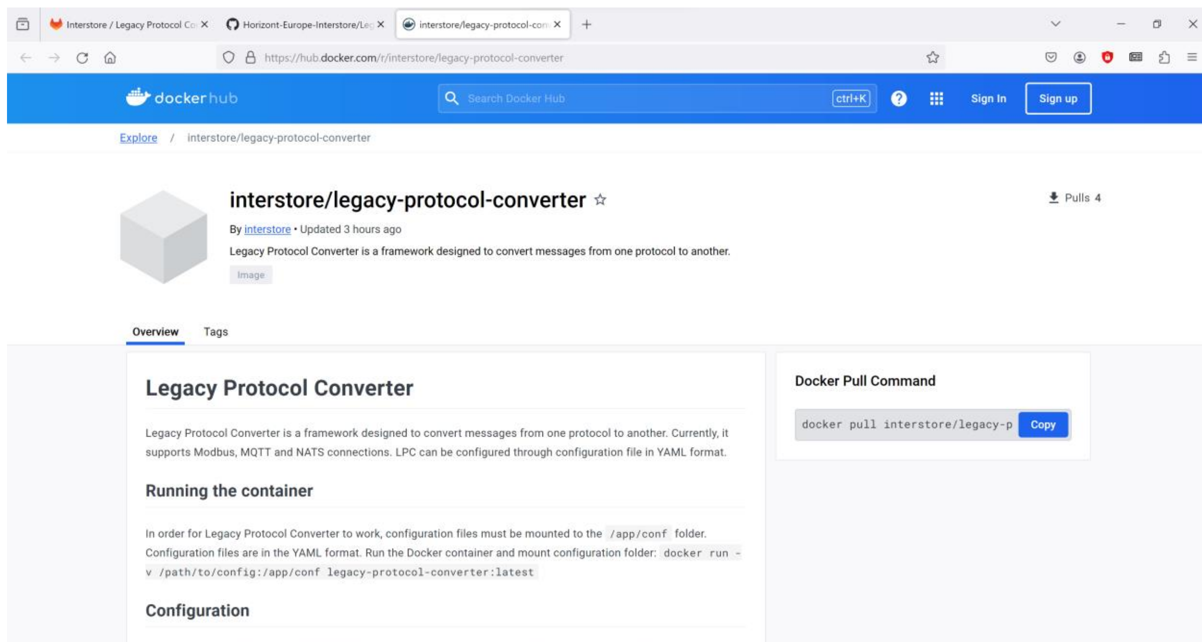

Figure 3: Doker image registry on Docker Hub.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

9 Conclusion

In this specification, we have provided overview of architecture and configuration for the Legacy Protocol Converter (initial version). The document provides a description of the software architecture, its components, packages and classes. It also provides a list of technologies used.

Furthermore, this document provides a detailed description of the configuration, parameters, and transformation engine to enable seamless protocol transformation between ModBus, MQTT and IEEE2030.5 using NATS in either XML or JSON representation. Legacy protocol converter provides support for ModBus and MQTT and provides a transformation and configuration framework, which allows simple and fully configurable transformation of messages between ModBus, MQTT, NATS, IEEE2030.5 XML and JSON messages.

LPC provides full support for next generation NATS messaging as a communication protocol between devices and EMS systems. This enables message-driven, loosely coupled and scalable communication platform, which provides out-of-the-box support for computer cloud environments.

Legacy protocol converter follows the microservice architecture. Is has been implemented in Java using open source OpenJDK using Java Microprofile and KumuluzEE framework. It is provided as Docker container, pre-built Java JAR archive, or custom-built configuration for specific use cases. All software artifacts are available on GitHub and Docker Hub.

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

10 REFERENCES

IEEE Standard for Smart Energy Profile Application Protocol. (2018). *IEEE Std 2030.5-2018 (Revision of IEEE Std 2030.5-2013)*, 1-361. doi: 10.1109/IEEESTD.2018.8608044.

Modbus. (2023, September 25). Retrieved from Mobus: https://modbus.org/

MQTT. (2023, September 25). Retrieved from MQTT: https://mqtt.org/mqtt-specification/

GNU. (2025, September 25). Retrieved from GPL: https://www.gnu.org/licenses/gpl-3.0.html

Apache. (2023, September 25). Retrieved from Apache: https://www.apache.org/licenses/LICENSE-2.0

NATS. (2023, September 25). Retrieved from Documentation: https://docs.nats.io/

Java Microprofile, https://microprofile.io/

KumuluzEE, https://ee.kumuluz.com/

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.

11 LIST OF FIGURES

Figure 1: Legacy Protocol Converter package diagram....7
Figure 2: Open-source repository on GitHub....18
Figure 3: Doker image registry on Docker Hub. 19

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096511. Disclaimer: The sole responsibility for any error or omissions lies with the editor. The content does not necessarily reflect the opinion of the European Commission. The European Commission is also not responsible for any use that may be made of the information contained herein.