# D2.1 - INTEROPERABLE CLIENT/SERVER FOR DISTRIBUTED ENERGY STORAGE

WP2 - Open-source Interoperability Toolkit

T2.1 – Interoperable client/server for distributed energy storage

Submission date: 31 Mar 2024 (Initial version)

| | |
|---|---|
| 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)<br>CyberGrid (CYG) | 2023-02-29 |
| Checked by | Matjaz Juric (SUN) | 2023-03-12 |
| Reviewed by | Ferdinando Bosco (ENG)<br>David Trafela (SUN) | 2024-03-14 |
| Approved by | Antonello Monti (RWTH) – Project Coordinator<br>Francesco Guaraldi (ENX) | |
| Status | Final | |

# DISSEMINATION LEVEL

| | | |
|---|---|---|
| CO | Confidential | |
| CL | Classified | |
| PU | Public | X |

# VERSIONS

| Date | Version | Comment |
|---|---|---|
| 14. 12. 2023 | 0.1 | Outline - draft |
| 8. 1. 2024 | 0.2 | Improved outline |
| 22. 1. 2024 | 0.6 | Software architecture description |
| 1. 2. 2024 | 0.7 | Languages and Tools |
| 21. 2. 2024 | 0.8 | Data models, deployments |
| 4. 3. 2024 | 0.9 | Open-source access, improvements |
| 6. 3. 2024 | 0.95 | UML diagram added |
| 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

# ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| API | Application Programming Interface |
| ASF | Apache Software Foundation |
| BSD | Berkeley Source Distribution |
| EMS | Energy Management System |
| FSF | Free Software Foundation |
| GPL | General Public License |
| HESS | Hybrid Energy Storage Systems |
| JAR | Java ARchive |
| JAXB | Java API for XML Bindings |
| 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 |
| XSD | XML Schema Definition |
| 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



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 purpose of this document is to describe the interoperable client/server for handling IEEE2030.5 messages over NATS. NATS is a high-performance, secure messaging system designed for distributed systems, microservices, IoT devices, and cloud-native applications.

Interoperable client/server is designed to provide an open source, out-of-the-box support for IEEE2030.5 communication between devices and EMS systems.

The objectives of the interoperable client/server are to:

- Provide an open source, out-of-the-box support for IEEE2030.5 communication between devices and EMS systems. The objective is to provide support for IEEE2030.5 messages in original XML format, as well as IEEE2030.5 in JSON format.
- Provide support for next generation NATS messaging as a communication protocol between devices and EMS systems, superseding other communication mechanisms (such as REST over HTTP) and enabling message-driven, loosely coupled and scalable communication platform.
- Provide a reference implementation as an open-source project, available on GitHub.

In this specification we have presented the architecture for the interoperable client/server (we will use the wording *client/server* in the rest of the document). It provides support for IEEE2030.5 messages in original XML format, as well as IEEE2030.5 in JSON format. It supports next generation NATS messaging as a communication protocol between devices and EMS systems, superseding other communication mechanisms (such as REST over HTTP) and enabling message-driven, loosely coupled and scalable communication platform.

The document describes the software architecture of the interoperable client/server library, usage scenarios and interfaces it provides. Client/server is not a standalone software product, but a software library to be used within devise and/or EMS systems to implement IEEE2030.5 over NATS.

The implementation is open-sourced and available 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.

# 2 Software Architecture and Usage Scenarios

Client/server consists of three modules:

- Client module,
- Server module and
- Client-server-common.

Client-server-common module is used for generating Java classes from XSD and for serializing/deserializing IEEE2030.5's types.

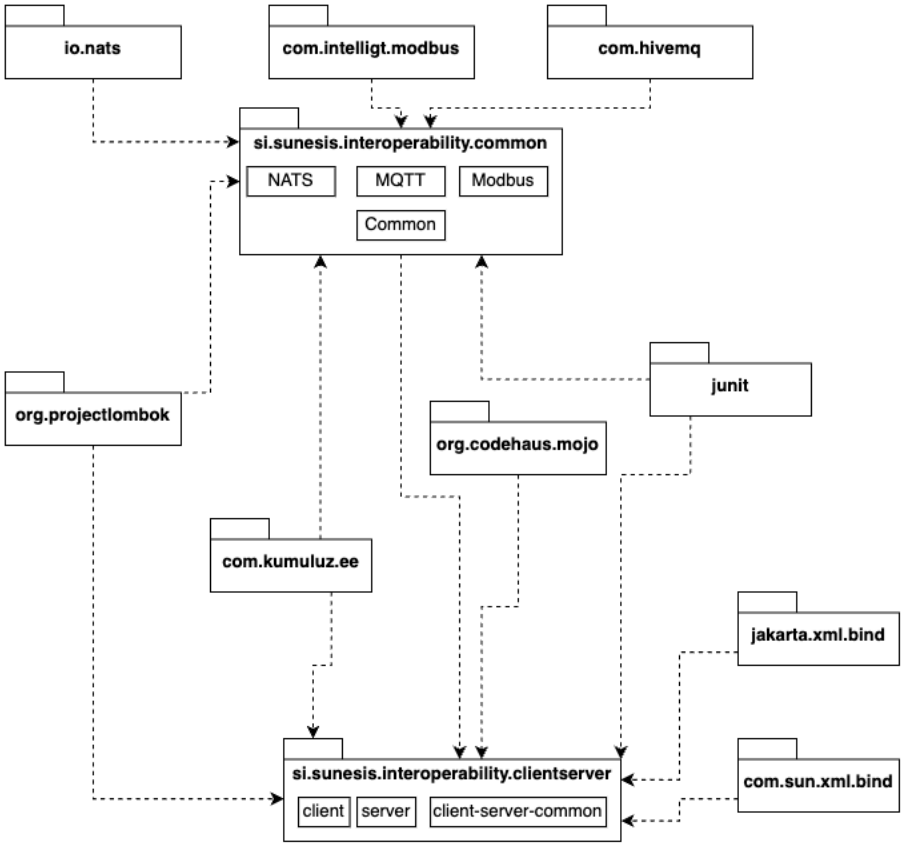


*Figure 1: Client/server package diagram.*

Figure 1: Client/server package diagram shows which packages are used within the interoperability common and client/server. For each of these two projects sub-modules are listed.

### 2.1.1 Serializing/deserializing

In package si.sunesis.interoperability.clientserver.common is a class IEEEObjectFactory that exposes static methods for serializing XML/JSON message to specified IEEE2030.5 type and for deserializing IEEE2030.5 types to XML/JSON message.

XML serialization is done with the help of Java's built-in JAXB methods.

Code snippet for serializing/deserializing XML message:

```
public static <T> T fromXMLToIEEE(String xml, Class<T> type) {
    if (!xml.contains("xmlns")) {
        xml = xml.replace("<" + type.getSimpleName() + ">", "<"
                + type.getSimpleName() + "
xmlns=\"http://ieee.org/2030.5\">");
```

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.

```
    }

    JAXBContext jaxbContext =
JAXBContext.newInstance(ObjectFactory.class);
    return ((JAXBElement<T>) jaxbContext.createUnmarshaller()
            .unmarshal(new StringReader(xml)))
            .getValue();
}

public static <T> String IEEEToXML(T element, Class<T> objectClass)
{
    StringWriter stringWriter = new StringWriter();

    JAXBContext jaxbContext =
JAXBContext.newInstance(ObjectFactory.class);
    Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

    // format the XML output
    jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
true);

    QName qName = new QName("http://ieee.org/2030.5",
objectClass.getSimpleName());
    JAXBElement<T> root = new JAXBElement<>(qName, objectClass,
element);

    jaxbMarshaller.marshal(root, stringWriter);

    return stringWriter.toString();
}
```

Example usage of methods for XML:

```
Event event = IEEEObjectFactory.fromXMLToIEEE(XMLInput,
Event.class);
log.info("Event: {}", event);

String xml = IEEEObjectFactory.IEEEToXML(event, Event.class);
log.info("XML: {}", xml);
```

JSON serialization is done with the help of Gson library (https://github.com/google/gson), same classes generated from XSD can be used, but keys must start with lowercase character, that is why the step with setting the first character to lowercase is necessary.

Code snippet for serializing/deserializing JSON message:

```
public static <T> T fromJSONToIEEE(String json, Class<T> objectClass)
{
        Gson gson = new GsonBuilder()
                .registerTypeHierarchyAdapter(TimeType.class, new
TimeTypeAdapter())
                .create();

        Type type = new TypeToken<Map<String, Object>>() {
        }.getType();
```

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.

```
        Map<String, Object> originalMap = gson.fromJson(json, type);

        // Set first character to lowercase if it is not
        Map<String, Object> newMap = new HashMap<>();
        for (Map.Entry<String, Object> entry :
originalMap.entrySet()) {
            newMap.put(entry.getKey().substring(0, 1).toLowerCase() +
entry.getKey().substring(1), entry.getValue());
        }

        String newJsonInput = gson.toJson(newMap);

        return gson.fromJson(newJsonInput, objectClass);
    }

    public static String IEEEToJSON(Object object) {
        Gson gson = new GsonBuilder()
                .registerTypeHierarchyAdapter(TimeType.class, new
TimeTypeAdapter())
                .setPrettyPrinting()
                .create();

        return gson.toJson(object);
    }
```

Example usage of methods for JSON:

```
Event event = IEEEObjectFactory.fromJSONToIEEE(jsonInput,
Event.class);
log.info("Event: {}", event);

String json = IEEEObjectFactory.IEEEToJSON(ev);
log.info("JSON: {}", json);
```

### 2.1.2 Communication with NATS

NATS module for communication is in the interoperability-common project as it is also used in the Legacy Protocol Converter.

For this, RequestHandler class was developed. It exposes a simple interface for receiving and sending messages. Users must define their own parameters for the connection to the NATS.

Example of connecting to NATS and subscribing/publishing:

```
Options options = new Options.Builder()
        .server("nats://localhost:4222")
        .build();

NatsConnection natsConnection = new NatsConnection();
try {
    natsConnection.connectSync(options, false);

    Server server = new Server(natsConnection);

    server.subscribe("devices.event", message -> {
        String msg = new String(message);
```

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.

```
        log.info("Received message: {}", msg);

        Event event = IEEEObjectFactory.fromXMLToIEEE(msg,
Event.class);
        // Do something with the event

        server.publish("events.event",
IEEEObjectFactory.fromIEEEToJSON(event));
    });
} catch (IOException | InterruptedException e) {
    log.error("Failed to connect to NATS server", e);
}
```

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

Interoperable client/server developed in Java. Java 17 or higher is required, with plans to upgrade to Java 21 in the future releases.

The source code is based on Java Microprofile specification. KumuluzEE (https://ee.kumuluz.com/) framework is used for the project as it is an 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 Client/Server is also using a common package from group si.sunesis.interoperability.common for managing connections with NATS.

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
- Jaxb2-maven plugin, version 3.1.0

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

There are 321 available data models in the Client/Server which are defined in IEEE2030.5's sep.xsd. Those models come are generated in the project with the help of Jaxb2-maven plugin for converting XML to Java classes. These models are used for serializing and deserializing XML and JSON structures.

An example of generated Java class for IEEE2030.5 Event:

```
public class Event
        extends RespondableSubscribableIdentifiedObject {

    @XmlElement(required = true)
    protected TimeType creationTime;
    @XmlElement(name = "EventStatus", required = true)
    protected EventStatus eventStatus;
    @XmlElement(required = true)
    protected DateTimeInterval interval;

    /**
     * Gets the value of the creationTime property.
     *
     * @return possible object is
     * {@link TimeType }
     */
    public TimeType getCreationTime() {
        return creationTime;
    }

    /**
     * Sets the value of the creationTime property.
     *
     * @param value allowed object is
     *              {@link TimeType }
     */
    public void setCreationTime(TimeType value) {
        this.creationTime = value;
    }

    /**
     * Gets the value of the eventStatus property.
     *
     * @return possible object is
     * {@link EventStatus }
     */
    public EventStatus getEventStatus() {
        return eventStatus;
    }

    /**
     * Sets the value of the eventStatus property.
     *
     * @param value allowed object is
     *              {@link EventStatus }
     */
    public void setEventStatus(EventStatus value) {
```

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.

```
        this.eventStatus = value;
    }

    /**
     * Gets the value of the interval property.
     *
     * @return possible object is
     * {@link DateTimeInterval }
     */
    public DateTimeInterval getInterval() {
        return interval;
    }

    /**
     * Sets the value of the interval property.
     *
     * @param value allowed object is
     *              {@link DateTimeInterval }
     */
    public void setInterval(DateTimeInterval value) {
        this.interval = value;
    }
}
```

An example of generated JAVA class for IEEE2030.5 DateTimeInterval:

```
public class DateTimeInterval {

    @XmlSchemaType(name = "unsignedInt")
    protected long duration;
    @XmlElement(required = true)
    protected TimeType start;

    /**
     * Gets the value of the duration property.
     *
     */
    public long getDuration() {
        return duration;
    }

    /**
     * Sets the value of the duration property.
     *
     */
    public void setDuration(long value) {
        this.duration = value;
    }

    /**
     * Gets the value of the start property.
     *
     * @return
     *     possible object is
     *     {@link TimeType }
```

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.

```
     *
     */
    public TimeType getStart() {
        return start;
    }

    /**
     * Sets the value of the start property.
     *
     * @param value
     *     allowed object is
     *     {@link TimeType }
     *
     */
    public void setStart(TimeType value) {
        this.start = value;
    }
}
```

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 Deployment, configuration, and usage

For building the code, Maven is needed as this is a Maven project. There are two ways this project can be integrated, by importing JAR or by importing from Maven Central.

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.

This project is not designed to be ran as a standalone, as handling of the messages must be implemented accordingly.

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 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 interoperable client/server is available here: https://github.com/Horizont-Europe-Interstore/Client-Server

A screenshot of the repository is shown in the figure below.

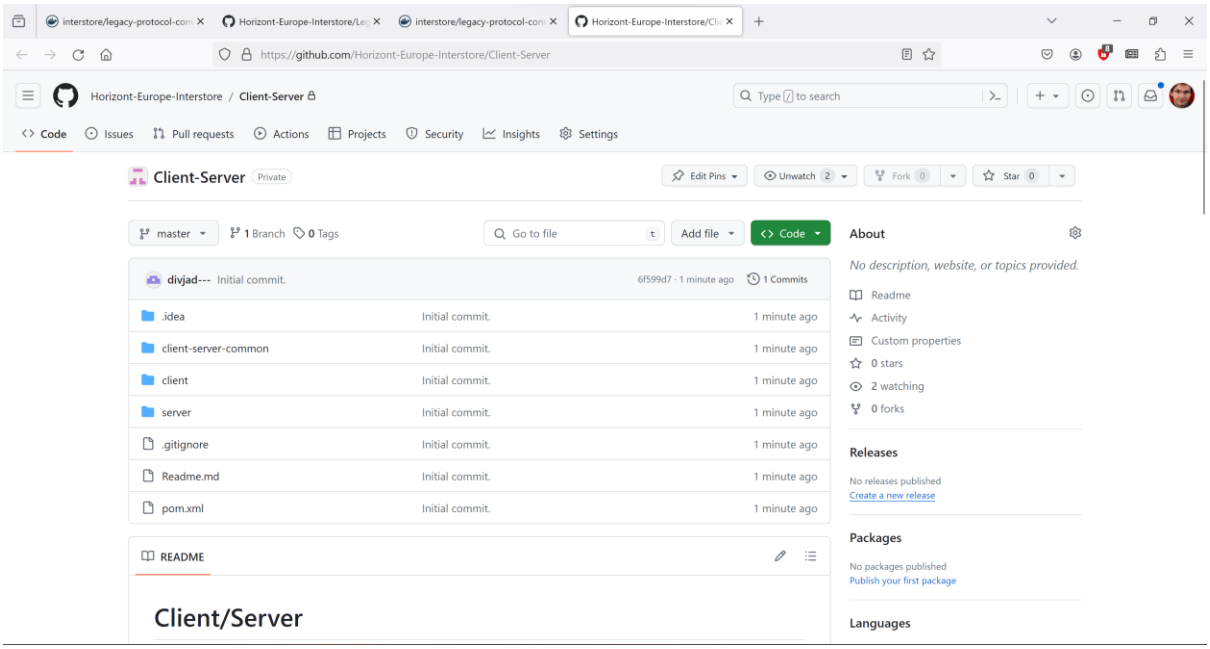


*Figure 2: Open-source repository on GitHub.*

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.

# 7 Conclusion

In this document, we have described the interoperable client/server (initial version), a software library that enables IEEE2030.5 communication between devices and EMS systems over NATS. The interoperable client/server has the following objectives and features:

- It provides an open source, out-of-the-box support for IEEE2030.5 messages in both XML and JSON formats.
- It supports next generation NATS messaging as a communication protocol, which offers better performance, scalability and reliability than other protocols.
- It provides a reference implementation as an open-source project, which can be used and modified by anyone.

The interoperable client/server is designed to facilitate the integration of IEEE2030.5 devices and EMS systems, and to promote the adoption of this standard in the smart grid domain. We hope that this document will serve as a useful guide for developers and users who want to use the interoperable client/server in their projects.

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 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.

# 9 LIST OF FIGURES



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.