D2.3 – TESTING PROCEDURES AND SOFTWARE TOOLS

WP2 - Open-source Interoperability Toolkit

T2.3 – Testing Procedures and Software Tools

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	RWTH Aachen	
Checked by		
Reviewed by	Marcantonio La Franca (ENG) Matjaz Juric (SUN)	21/03/2024 25/03/2024
Approved by	Antonello Monti (RWTH) – Project Coordinator Francesco Guaraldi (ENX)	
Status	Final	

DISSEMINATION LEVEL

CO	Confidential	
CL	Classified	
PU	Public	X

VERSIONS

Date	Version	Author	Comment
19-03-2023	0.10	Nithin Manuel(RWTH)	Added the UML diagram
19-03-2023	0.20	Nithin Manel	Added test case results
20-03-2023	0.30	Nithin Manuel	Added chapter 6, 7, 8 , 9
21 -03-2023	0.40	Nithin Manuel	Submitted for review
22-03-2023	0.50	Nithin Manuel	Revisioned on review
25-03-2023	0.60	Nithin Manuel	Added Executed Summery
27-03-2023	0.70	Nithin Manuel	Explained code in detail
27-03-2023	0.80	Nithin Manuel	Architecture comments added
27-03-2023	0.90	Nithin Manuel	Addressed all comments of reviewers
27-03-2023	1.0	Nithin Manuel	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

NATS	Neural Autonomic Transport System
IEEE	Institute of Electrical and Electronics Engineers
RESTful	Representational State Transfer
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JDK	Java Development Kit
UML	Unified Modelling Language

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.

EXECUTIVE SUMMARY

The Energy Engineering and industry experiencing tremendous of updates and demands to meet the daily challenges in various energy sectors. There are many energy management strategies available in the market to keep up with the demands, however the most important challenges are the interoperability between devices and how can it become realizable in real time conditions considering all stakeholders involved in the energy supply chain form producer to consumer. To link all resources and means the common point is the interoperability to achieve the this, there is need of robust protocols which enable all stages of the energy production to consumption. In the light of the need for such a protocol the one which address the interoperability is IEEE 2030.5. To briefly comment about it, it's a protocol that communicate between with different energy devices and systems this allows to board different manufactures, platforms to common ground. Not only this the integration of renewable energy into the grid, but the protocol also seamlessly communicates between the renewable energy systems and grid. The protocol support demand response program this allows the use of energy in most efficient manner. When it considers from the perspective of the energy management it's import that, control, monitor, and optimization of energy consumption in buildings and grids.

The IEEE 2030.5 protocol is wide protocol and it address from grid level to consumer as mentioned above, to ensure the interoperability, the protocol needs undergone conformance tests for this reason we developed an initial version of the prototype test software with core tests. The software is fully automated and at the end of the test it compares with expected and actual outcome with reporting pass or fail status. The testing software is composed of open-source tools such as Cucumber, a tool to control the flow of testing from client side and the server side is done with tools in Java Ecosystem. A vital point to mention that the test software is brave attempt to test a restful architecture-based IEEE 2030.5 with a message system called NATS, to put in simple words instead of testing using http it uses message broker. This novelty makes the test software is totally unique and different from other attempts to test the same protocol. The software will be capable to run regardless of the operating system environment to make it as like cross platform that ensure more flexibility and more user friendly. Since the approach to test IEEE 2030.5 using light weight and fast message broker system the testing procedures must adapted to the messaging system, the testing procedures implemented in this test software is in this regard.

To put it in a nutshell, the test software and testing procedures is a prototype and an initial version to test the core functionalities of IEEE 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.

1 Introduction

This Document addresses the testing procedures and corresponding developed test software to Test IEEE 2030.5 using NATS.

NATS is a high-performance messaging system for IOT messaging, cloud native applications and microservices, it's capable of lightweight and secure way to send and receive messages across distributed systems, facilitate applications to communicate each other in real time, NATS is known for its speed and reliability. The testing procedures were adapted and inspired by Sunspec, a standard defined for testing the IEEE 2030.5 protocol, the Sunspec is alliance of developers, manufactures, operators, and service providers together pursuing open information standards for distributed energy industry. SunSpec Standards address most operational aspects of PV, Storage, and other distributed energy power plants on the smart grid, including residential, commercial, and utility-scale systems, thus reducing cost promotion innovation. Testing IEEE 2030.5 protocol is challenging because of its nested resource specifications and many interconnected features. Even though the architecture of the software is designed in Restful way the test software designed with message system called NATS, it acts as the communication medium between the testing client and server where the IEEE 2030.5 resources developed.

The testing software is automated for tests, it verifies with the expected outcome with actual outcome of the test, once it got verified the outcome is stated as testes passed with depicts expected and actual outcome, if not it shows the steps where the tests failed. The resources within resources with IEEE 2030.5 are structured in a nested manner, interconnected via hyperlinks for accessibility. The execution of many tests requires the successful completion of preceding tests, as the outcomes of earlier tests are crucial for subsequent ones. Therefore, the software is handling this dependency by seamlessly navigating between related tests. This is achieved through the software's ability to automatically identify the resources on the server and synchronize interrelated tests. One important aspect to consider is that since this software is intended and made of standard IEEE 2030.5 specifications however different clients (users) can configure the resources according to it needs , in this condition the user has to update corresponding values in the test software to make sure that the test environment is similar to the one close to the production, by this way it can achieve maximum efficiency.

Testing procedure is the one of the important aspects of this deliverable owing to the fact that it should be align and similar to that of the Sunspec procedures, but also it has to be compactable with the test tool itself. The testing procedure must adapt to the NATS message systems instead of http-based communication, this shall include the omission in testing procedure the parts which address the http communication form the Sunspec set up the NATS part for it. The software tool consists of ways to automatically carry out the tests once it's starts, to achieve this there is a need to use other open-source libraries and tools to build a robust testing tool. Considering that fact there are different tools such as Cucumber, which is an automated test library where the user can specify the test cases in plain English . Other libraries which handle the internal working of the software to integrate different part of the software, from the start of the test (cucumber) then with the NATS and its connection with IEEE 2030.5. The integration and proper binding between different part of the software tool plays a vital role in proper functioning of the software.

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

The below UML diagram (Figure 1) shows how the testing software is developed.

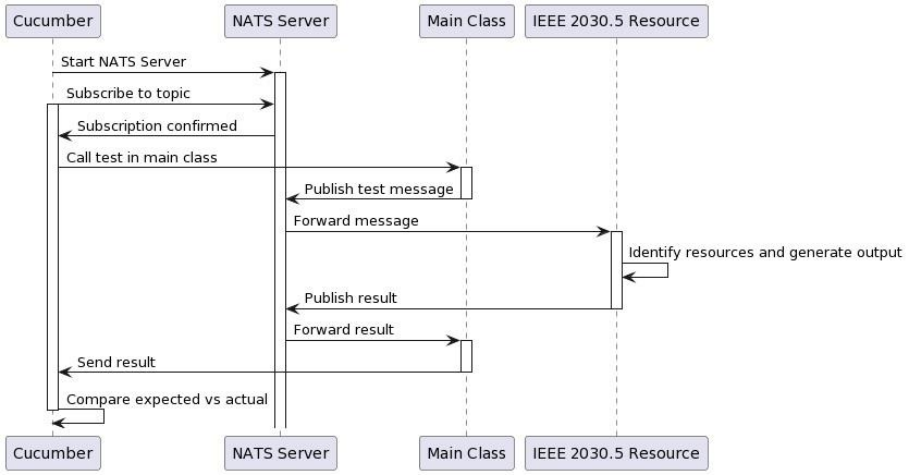

Figure 1: UML Diagram of test Software

The Software architecture is with a client server model, where the client is referred to the cucumber in the left side of the figure 1 and the communication between client and server is carried out by a messaging system called NATS, indicated in the diagram from cucumber to Main Class. The server part is where the IEEE 2030.5 resource is present although the names such as client or server not explicitly mentioned in the figure 1 instead of that directly specified the components such as cucumber and IEEE 2030.5 resources.

2.1 The Test software components.

2.1.1 Automated Testing

The Cucumber is an open source software tool for testing for black box testing in a behavior driven manner-, this way of testing includes the mocks , integration and test the feature without any external input from the user, the only thing is necessary is that the expected outcome has to be specified in the cucumber feature file , there will be a corresponding step file for feature where it programmatically set up the needs for testing . Feature file is a plain English file, and the step file is corresponding programmed version. There is one more file for Configuration, where it specifies the locations and annotations to run the cucumber. The automated nature of the cucumber is the reason

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.

why it was chosen as the first place to develop the testing software. Not only this but also the IEEE 2030.5 testing is multistep testing several steps involved in a logical order, for instance it works like given, when, then manner, in given set up, when the tests happen then the outcome and this pattern repeats for multistage.

From the Cucumber feature files it first does do the necessary steps to do the communication by set up the NATS, after that it trigger the test that must be tested in IEEE 2030.5 resources and receive the response. If it's matched with the expected outcome, it will show that the test is passed or shows the tests passed and not passed, it makes a report of its test describing the test outcome. The cucumber feature file and step file depict in the next pages.

Cucumber feature file for testing IEEE 2030.5 Device Capability

```
Feature: Device Capability Test

  @DeviceCapability

  Scenario Outline: Verifying Device Capability Test Execution

    Given I have a device capability test setup

    When I execute the device capability test with service name "dcapmanager" and subject "<natsSubject>"

    Then the test should complete successfully with DeviceCapability response containing:

    """

    {

      "MirrorUsagePointListLink": "http://localhost/interstore/MirrorUsagePointListLink",

      "endDeviceListLink": "http://localhost/interstore/endDeviceListLink",

      "selfDeviceLink": "http://localhost/interstore/selfDeviceLink"

    }

    """
```

Formatted: Font: (Default) Courier New

Cucumber Step file for testing IEEE 2030.5 Device Capability

```
import static org.junit.jupiter.api.Assertions.*;

import interstore.App;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.fasterxml.jackson.databind.JsonNode;
```

Formatted: Font: (Default) Courier New

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.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


public class DeviceCapabilitySteps {


    @Autowired
    private App app;
    private static final Logger LOGGER = LoggerFactory.getLogger(DeviceCapabilitySteps.class);
    private String response;



    @Given("^I have a device capability test setup$")
    public void i_have_a_device_capability_test_setup() throws Exception {
        String natsUrl = "nats://localhost:4222";
        app = new App(natsUrl);
    }


    @When("^I execute the device capability test with service name \"([^\"]*)\" and subject \"([^\"]*)\"$")
    public void i_execute_the_device_capability_test_with_service_name_and_subject(String serviceName, String natsSubject) throws Exception {


        response = app.DeviceCapabilityTest(natsSubject);
    }


    @Then("^the test should complete successfully with DeviceCapability response containing:$")
    public void the_test_should_complete_successfully_with_DeviceCapability_response_containing(String expectedJson) throws Exception {
        ObjectMapper objectMapper = new ObjectMapper();
        JsonNode expected = objectMapper.readTree(expectedJson);
        JsonNode actual = objectMapper.readTree(response);
```

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.

```
        LOGGER.info("Expected response: {}", expected + "and " +
"actual response is "
        + actual);
        assertEquals(expected, actual, "The actual response does not
match the expected response.");
    }



}
```

2.1.2 NATS for Communication

Although the IEEE 2030.5 protocol developed with REST full architecture, the testing software adapted the messaging system called NATS which can set up in different ways for communication. In the test software is to leverage the Publish / Subscribe feature of NATS communication mechanism. NATS has a server and client, where NATS server does the connection with publisher and subscriber, the NATS server must present in the software or in the running environment of the software to coordinate the actions.

Since NATS communication is through messaging each message requires a subject in the test software there is a provision to use custom subject or default Once the message is constructed with a given subject and a payload, it first must do the subscription first, implies subscribe to the subject first after that carry out the publish. Once the NATS message reaches the subscriber it finds it's right path to forward the message. The below code snippet demonstrates how the Nats communication works, Message to Publish class will publish the message, to do so first it establish a connection with nats server after this it does a subscribe to the message first, followed by a publish on the message with a subject , the subject is a generic name this name can be any name but it's logical to choose a subject that complement the chosen test . The resource has to be tested will automatically identify the message and return a response, the response will be again a subscribe and publish and deliver the test results to compare with the expected results.

Nats Publish and Subscribe Module.

```
 public class MessageToPublish {
   private ServiceDiscoveryVerticle serviceDiscoveryVerticle;
   private ServiceDiscoveryVerticle serviceDiscovery;
   private  Connection natsConnection;
   private       static       final       Logger       LOGGER       =
Logger.getLogger(MessageToPublish.class.getName());
```

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 MessageToPublish() {
    this.serviceDiscovery = serviceDiscoveryVerticle;
    }

    public MessageToPublish(String natsUrl,ServiceDiscoveryVerticle
serviceDiscoveryVerticle ) {

        this.serviceDiscoveryVerticle = serviceDiscoveryVerticle;
        LOGGER.info("Servcie discovery verticle is : " +
this.serviceDiscoveryVerticle);
        try
        {
            this.natsConnection = Nats.connect(natsUrl);
        } catch(Exception e)
        {
            LOGGER.severe("Error connecting to nats server: " +
e.getMessage());
            e.printStackTrace();
            throw new RuntimeException(e);
        }


    }


    public void PublishToSubject(String natSubject, String message)

    {

        byte[] messageBytes =
message.getBytes(StandardCharsets.UTF_8);
        natsConnection.publish(natSubject, messageBytes);
```

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 void subscribeMessage(String natsSubject)
   {
    try
     {
        this.serviceDiscoveryVerticle.setupBridge(natsSubject);


     } catch(Exception e)
     {
        LOGGER.severe("Error subscribing to message: " +
e.getMessage());
        e.printStackTrace();
     }
   }


   public void reSubscribeMessage(String natsSubject, String natsUrl,
String serviceName )
   {
    try
     {
        ServiceDiscoveryVerticle msg = new
ServiceDiscoveryVerticle(natsUrl);
        msg.setupBridgeForResponse(natsSubject, serviceName);


     } catch(Exception e)
     {
        LOGGER.severe("Error subscribing to message: " +
e.getMessage());
        e.printStackTrace();
     }
   }
```

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 String responseToSender(String serviceName, String
responseMessage)
   {
    MessageFactory messageFactory = new MessageFactory();
    messageFactory.selfDeviceEndDeviceTests(serviceName,
responseMessage);
    return responseMessage;
   }
```

2.1.3 Automatic Service Discovery

Upon subscribing to a NATS message, the system promptly forwards it to an automatic service discovery, facilitated by dependency injection. Within this framework, the IEEE 2030.5 resource has been pre-established and maintained via dependency injection. When a NATS message payload aligns, the system dynamically identifies the appropriate resource. This dynamic identification process leverages a technique known as Reflection, enabling the system to adaptively locate the correct resource.

2.1.4 IEEE 2030.5 Resource

The IEEE 2030.5 consists of many resources, these resources in IEEE 2030.5 protocol are developed as per Restful architecture. To test the resources, the resources must be developed, the important tests are core tests of IEEE 2030.5, to do so developed the part addressing the core test and it's tested as per sun spec doc.

In the resource is treated as service so the resource part or server part of this application consist of a manager class and a service class along with data transfer object. For simplicity the data is stored in an inbuilt data type.

```
package interstore.DeviceCapability;
import java.util.List;
import interstore.Identity.Link;
import interstore.Identity.ListLink;
import interstore.Types.UInt32;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;
```

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 class DeviceCapabilityDto {
    public UInt32 pollRate = new UInt32(900);
    private String id;
    private ListLink listLink;
    private String endDeviceListLink;
    private String MirrorUsagePointListLink;
    private String selfDeviceLink;
    private List<String> endDeviceList;
    private List<String> MirrorUsagePointList;
    private Link link ;
    private Map<String , String> deviceCapabilityLinks;
    private static final Logger LOGGER =
Logger.getLogger(DeviceCapabilityDto.class.getName());



    public DeviceCapabilityDto() {
        this.link = new Link();
        this.listLink = new ListLink();
        this.deviceCapabilityLinks = new HashMap<>();
        this.getAllLinks();
    }


    public String getId() {
        return id;
    }

    public Map<String , String> deviceCapabilityOfLinks() {
        this.endDeviceListLink =
this.listLink.createListOfLink("/endDeviceListLink");
        this.deviceCapabilityLinks.put("endDeviceListLink" ,
this.endDeviceListLink);
        this.MirrorUsagePointListLink =
this.listLink.createListOfLink("/MirrorUsagePointListLink");
        this.deviceCapabilityLinks.put("MirrorUsagePointListLink" ,
this.MirrorUsagePointListLink);
        this.selfDeviceLink =
this.link.createLink("/selfDeviceLink");
        this.deviceCapabilityLinks.put("selfDeviceLink" ,
this.selfDeviceLink);
        return this.deviceCapabilityLinks;

    }

    public Map<String , String> getAllLinks()  {
        return this.deviceCapabilityOfLinks();
    }


    public List<String> getEndDeviceList(String endDeviceListLink) {
        if (endDeviceListLink == this.endDeviceListLink.toString())
{
            return this.endDeviceList;
```

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.

```
        }
        else {
            return null;
        }

    }

    public List<String> getMirrorUsagePointList(String
MirrorUsagePointListLink) {
        if (MirrorUsagePointListLink ==
this.MirrorUsagePointListLink.toString()) {
            return this.MirrorUsagePointList;
        }
        else {
            return null;
        }
    }



    public String getEndDeviceListLink() {
        return this.endDeviceListLink;
    }

    public String getMirrorUsagePointListLink() {
        return this.MirrorUsagePointListLink;
    }

    public String getSelfDeviceLink(){
        return this.selfDeviceLink;
    }



}
```

3 Languages, Technologies and Tools

Component	Version/Description
Component	Java,io.nats:jnats, com.google.inject:guice, org.json:json, com.fasterxml.jackson.core:jackson-databind, org.modelmapper:modelmapper, io.cucumber:cucumber-spring, io.cucumber:cucumber-java, io.cucumber:cucumber-core, org.springframework.boot:spring-boot-

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.

	starter-test, Gradle Version, JVM version, JDK version
Version/Description	17.0.10, 2.16.12: Nats client for java, 5.1.0: A lightweight dependency injection framework, Library for working with JSON, 2.16.1: JSON library for Java, 2.4.5: Simplifies the mapping between Java models, 7.1.0: Cucumber support for Spring, 7.1.0: Core Cucumber library for Java, 7.1.0: Core Cucumber library, 3.2.2: Starter for testing Spring Boot applications with libraries including JUnit, Hamcrest, and Mockito, 8.0.2, 17.0.10, 17.0.10

4 Testing Procedures

The Development of this software is a continuous development according to the Sunspec, there are tests categorized core tests, communication tests, basic tests. The most important among these tests is Core tests and among them there must test cases, in this version of the software is focused on this part.

Here is the testing procedure for an IEEE 2030.5 features, having said that to test these there is a need to develop many resources to facilitate the testing.

4.1 Testing Procedure example

The test name: Device Capability

Purpose: The Device capability test verifies that the Client (Cucumber) can retrieve Device Capability (DCAP) resource from an IEEE 2030.5 server.

Procedure

1. Perform a GET operation on the Device Capability Resource
2. Process the retrieved Device Capability resource and verify there is at least one resource in the Device Capability

PASS/FAIL Criteria

a) The Client (Cucumber) Requested and received the Device Capability resource from the Server, the Server responded with a conformant payload for its Device Capability.

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.

b) The Client (Cucumber) found at least one resource included in the Device Capability, the Server include at least one resource link in returned Device Capability to the Client (Cucumber)

c) Client (Cucumber) Successfully received the payload of the found resource from the Device Capability resource.

4.2 Demonstration of the Test Outcome

To execute the test via command line

```
./gradlewcucumber-
Dcucumber.feature=src=/test/resources/interstore/DeviceCapability.f
eature
```

Above mentioned Device Capability is executed in the program and the result of the execution is given below.

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.

```
Mar 19, 2024, 4:52:58 PM interstore.ServiceDiscoveryVerticle
lambda$getMessageHandler$0
INFO: Received message from
NATS{"action":"get","servicename":"dcapmanager"}
Mar 19, 2024, 4:52:58 PM interstore.ServiceDiscoveryVerticle
subscribeFromNats
INFO: Dispatcher created and subscribed to subject: "<natsSubject>"
  When I execute the device capability test with service name
"dcapmanager" and subject "<natsSubject>"
# interstore.stepdefinitions.DeviceCapabilitySteps.i_execute_
the_device_capability_test_with_service_name_and_subject(java.lang.String
,java.lang.String)
16:52:58.193 [main] INFO interstore.stepdefinitions.DeviceCapabilitySteps
-- Expected response:
{"MirrorUsagePointListLink":"http://localhost/interstore/
MirrorUsagePointListLink","endDeviceListLink":"http://localhost/interstor
e/
endDeviceListLink","selfDeviceLink":"http://localhost/interstore/selfDevi
ceLink"}
and actual response is
{"MirrorUsagePointListLink":"http://localhost/interstore/
MirrorUsagePointListLink","endDeviceListLink":"http://localhost/interstor
e/
endDeviceListLink","selfDeviceLink":"http://localhost/interstore/selfDevi
ceLink"}
Then the test should complete successfully with DeviceCapability response
containing:
# interstore.stepdefinitions.DeviceCapabilitySteps.the_test_should_
complete_successfully_with_DeviceCapability_response_containing(java.lang
.String)

1 Scenarios (1 passed)
3 Steps (3 passed)
0m3.072s
```

The above message successfully passed the test for Device Capability is one of the core tests in IEEE 2030.5. The type of the message is of kind JSON and returns the response below..

```
Expectedresponse:
{"MirrorUsagePointListLink":"http://localhost/interstore/MirrorUsage
PointListLink",

"endDeviceListLink":"http://localhost/interstore/endDeviceListLink",

"selfDeviceLink":"http://localhost/interstore/selfDeviceLink"}and
actual response is


{"MirrorUsagePointListLink":"http://localhost/interstore/MirrorUsage
PointListLink",
```

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.

```
"endDeviceListLink":"http://localhost/interstore/endDeviceListLink",
"selfDeviceLink":
"http://localhost/interstore/selfDeviceLink
```


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

1) For the Deployment you need to install OpenJDK (Java Development Kit) because it comes with JRE (Java Run Time environment) and JVM (Java Virtual Machine) in the machine of interest.
2) Link to download the Java 17 SE is: https://www.oracle.com/java/technologies/downloads/#java17
3) NATS Server must be installed in the machine of interest.
4) Link to download NATS server https://docs.nats.io/running-a-nats-service/introduction/installation
5) Clone the code from the git repository: https://github.com/Horizont-Europe-Interstore/testing_procedure_and_software_tools.git
6) Navigate to the folder called interstore and do a build using./gradlew build Linux machine.
7) Once the build finish run it using below command
8) . /gradlew cucumber -Dcucumber.feature=src/test/resources/interstore/.featurefile

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 Testing procedures and software tools available: https://github.com/Horizont-Europe-Interstore/testing_procedure_and_software_tools.git

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

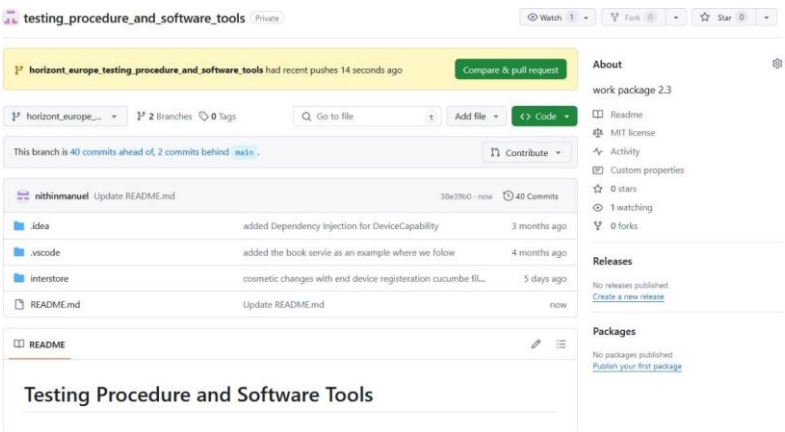

Figure 2: Open-source repository on GitHub.

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

The testing procedure and interoperable test tool described to test the IEEE 2030.5 resources and it demonstrates how the testing procedure seems to be and listed the outcome of the testing, by this way IEEE 2030.5 resources can be tested. Added to this the communication used is NATS which ensure that although the protocol is developed in other architecture the messaging system can deliver the results. By this way it can be conclude that messaging system shall be a middleware and potentially facilitate to achieve the goals of IEEE 2030.5.

In the next release of the software is aiming to add remaining tests.

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.

CSIPImplementationGuide2.1 03-15-2018

SunSpecCSIPCConformanceTestProcedureV1.0

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

Cucumber Documentation Retrived from https://cucumber.io/docs/cucumber/

Spring Boot Documentation Retrived from :

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

Google Guice Documentation : https://github.com/google/guice/wiki/GettingStarted

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.