Palisa Tuladhar

Test Automation in OMI Reference Implementation

Helsinki Metropolia University of Applied Sciences
Bachelor's Degree
Information Technology
Bachelor's Thesis
19 September 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages	Palisa Tuladhar Test Automation in OMI Reference Implementation 47 pages + 27 appendices
Degree	Bachelor in Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Käry Främing, Professor, Aalto University Manik Madhikermi, PhD Researcher, Aalto University Sami Sainio, Lecturer, Helsinki Metropolia UAS

OMI (Open Message Interface) model is a bridge and has its own cloud network among its subscribed members. The test automation of OMI interface aim to reduce development and production time by decreasing manual work.

The purpose of this thesis was to determine possible ways to implement the test automation of the OMI Reference model. As part of this research work, the test automation of the OMI Interface was done with the aim of evaluating the quality of the system and improving it. The testing complications, further enhancement of the standard OMI Interface and the test automation of the OMI interface along with technical proof is discussed in this paper.

The testing process of OMI Implementation codes were automated which was achieved by the development of a self-built testing tool with the help of JAVA programming language and setting up new required standards to assist the automation processes.

The project outcome is “Automated Test Cases” of the OMI Reference Implementation in JAVA build environment, which advances processes and practices to minimize development and production time. The quality of the products increases as testing is done continuously without human interaction. Hence, test automation minimizes the risks involved in testing and reduces the production cost.

Keywords	IoT, OMI, test automation process, automated test cases

DECLARATION

I hereby declare that all the texts written in this research paper are originally documented by myself. The sources and references used for the elaboration of the work are properly listed with the complete reference source.

I would like to thank my supervisor from Helsinki Metropolia UAS, Sami Sainio, for providing guidance in making this thesis possible and supervisors from Aalto University, Käry Främing, Manik Madhikermi for providing me with all the implementation details and invaluable teaching about software testing and all other Aalto staff involved who have helped me throughout my work placement period. Also, I would like to thank all the teachers from Metropolia UAS who have helped me throughout my study period.

- Palisa Tuladhar

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

IoT	Internet of Things
HTTP	Hypertext Transfer Protocol
OMI	Open Messaging Interface
ODF	Open Data Format
OMI-MF	Open Messaging Interface – Message Format
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
UDEF	Universal Data Element Framework
URL	Uniform Resource Locator
XML	Extensible Mark-up Language
WSDL	Web Services Description Language

1 Introduction

This thesis was written during work placement in the Computer Science Department of Aalto University with Professor Kary Främling, who is a Professor of Practice in Building Information Model, Dr. Tech. During the work placement, there arose a need for automated testing of OMI reference implementation after OMI interface to improve the quality of the software and test its functions. Hence, this project was carried out to improve the testing processes and shorten the testing lifecycle of the testing part by automating the test cases for the OMI Model. The main aim of this thesis work is to shorten the overall application testing process by reducing the time spent on regression testing. [1]

The thesis has three sub-objectives in order to achieve the main general objective. The first sub-objective is to make sure the designed test plan has all the required features to be tested and to implement the design of the automated testing framework in such a way that it covers the maximum test areas. The second sub-objective is to design and build an automated testing framework that improves the testing process and shortens the software development life cycle of the OMI model.

The framework must also enable the end to end testing of the OMI Model. The third sub-objective is to compare the time taken by the automated testing framework against the time taken by the manual testing process. Throughout the process, the main aim is to develop the framework by keeping in mind the future improvements in the test automation design and implementation. [1]

2 Software Testing

The actual software development process which includes coding, compiling and debugging of the software is called software testing. It is the main part of the software development where actual executable program code is generated as the output of the implementation stage. Furthermore, the output of the implementation is verified by the testing part. [2]

A software consists of huge entities and several sub systems that are assigned for various functions and procedures. After implementation, programmers debug programming codes to find out bugs. For debugging and fixing bugs, testing and validation need to be carried out in unit, module, sub/system and system level testing. The more detailed software testing process is discussed in the next chapter. [2]

The software development process that involves verification and validation of certain technical requirements of the software application is called software testing. The main aim of the software testing is to insure if the application is designed and developed as expected. The testing process also identifies the errors and defects in the application code that need to be fixed by reviewing the usability or functionality of the application. [2]

The four main purposes of software testing are explained below:

Verification
The process of evaluating whether the output of the software development satisfies the required condition is called verification. It is also called quality check process to find out if the final product meet the proposed product. [2]

Validation
The process that evaluates the final product (system) to determine whether it satisfies the required condition is called validation. It has a quality assurance role to verify the requirements. [2]

Defect finding
A defect is a deviance of result with an unexpected value. Usually, defects are the resultant of faults in specification, designing or development lifecycle of the software. [2]

Quality improvement

The testing process is not just about finding the faults in the product, as today testing is more than just debugging. Various strategies have been used for the verification and functional testing of the software which is called positive testing. Some tests are carried out to find out faults in the program which is called negative testing. Such strategies / tests help to improve the quality of the software. [1]

2.1 Software Testing Methods

Software testing is the process that determines the quality of the software under test. The main aim of software testing is to find various risks during software implementation. Many methods have been introduced with a motivation of making the testing process easier. The most common testing method that helps to correct functionality of the software is “The box approach”. Two of the common box approaches are described below. [3]

White box testing

White box testing, also known as transparent box testing, is the method that tests internal structure and mechanism of the software. In order to design test cases for this approach, knowledge of the overall functionality of the software and programming skills is required. In this method, the test cases are designed by testing the paths within the unit, between the units, during the integration of the system and between subsystems. This method can uncover most of the errors with the implementation rules and software codes. It is used in unit testing, integration testing and system testing these days. [4]

White box testing tests the software at the source code level. The main aim of using this method is to create an error free environment.

These test cases are generated following the design techniques:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage
- Modified condition/decision coverage. [4]

Black box testing

Black box testing is a software testing method that tests the functionality of the software without the knowledge of its internal structure/code and programming knowledge. This method can be applied in all levels of software testing such as unit testing, integration testing, system testing and acceptance testing. The tester should be aware of the outlook of the software but unaware of how it works. Although the tester does not know how the software works, one should be fully alert that a particular input returns a certain kind of output.

Black box testing tests the software at the external description of the software, specification, requirement and design pattern. The tester selects valid as well as invalid set of tests to find out the correct output per the description.

The test cases are generated following the following design techniques:

- Decision table testing
- All-pairs testing
- Equivalence partitioning
- Boundary value analysis
- Cause–effect graph
- Error guessing [5]

2.2 Software Testing Life Cycle

The testing life cycle of a software is basically a process with various steps that are used for testing the software application. Various stages are involved in the testing lifecycle to decide if the software is free of bugs and malfunctions. Even though several steps are involved in software testing processes, there isn't any specific set of patterns which need to be followed. The steps and orders in which they need to be executed depend on the mutual acceptance by the designer and the management team. [1] [6]

The general six steps that are involved in the software testing lifecycle are described below:

Step one: Review the Software Pattern

Before starting any test plan, the first step in the software testing lifecycle is to review the software pattern that involve its design/outlook, programming languages used, potential usability of the

product and its features. While examining the software, one should thoroughly examine the possible defects and report to the developer team. [6]

Step two: Planning the tests

The next and one of the most important part of software testing is to plan the test patterns. In this step, one should collect all the necessary details about the application to be tested. While planning the test, one should consider factors such as estimated budget, schedule and the requirements of the test. [6]

Step three: Designing the test data

After the proper planning of the test, the next step is to design test data. The design is made based on the test plan and review of the software. While designing, one should think of possible test data and test cases that need to be included. Creativity and imagination play a great role in this step. [6]

Step four: Building the test environment

After designing the test data, the next step is to build the test environment. These days, various functional testing tools such as Selenium, Canoo Web Test are used in software developing companies. But if there is a limited number of testing data, one can design his/her own testing environment. [6]

Step five: Testing and documenting results

After building the test environment, the next step is testing. The software should be tested in the test environment by following the test plan. After that, the obtained results should be properly documented as test cases. This step defines the pass and fail stage of the testing data. The failed test cases mean that there are bugs in the software which should be properly mentioned in the test cases documentation. [6]

Step six: Final reporting

After documenting all the possible test cases, the next and the final step is to prepare a report including all the test cases and submit the final report to the company. [6]

2.3 Testing Types

The test types clearly define the objective of a certain test level for a project. Not all programs require to test only the functionality of the system. Therefore, one should think of using different type of testing according to the objective of the level of the project or program. According to the test level, the test cases are classified as follows: [7]

Functional Testing

Software testing that involves testing of developed or under-developed software against the requirement specification of the software is known as functional testing. The testing data may be a piece of code or a whole project depending on the type and structure of functional testing. Furthermore, functional testing can be classified into the following categories depending on the tests, codes, data structures and logics involved. [2]

Unit testing
Unit testing follows the white box testing methodology in which code functionality is tested in function and/or class level. Mostly, unit testing is done by developers to test a certain part of the software. [2]

Regression testing
Regression testing also follows the white box testing methodology in which new functionality added to the program is tested. In this testing type, at first all the unit tests are tested and if they pass then the new functionality is introduced at the base of the code. [2]

Integration testing
Integration testing involves testing of code modules, client and server applications in a network, individual applications and so forth. The main purpose of integration testing is to find defects in connection between integrated components and interfaces. [2]

Acceptance testing
Acceptance testing follows the black-box testing methodology to test the system on the basis of customer requirements. The main purpose of this testing technique is to check if the completed sub system meets the desired level of requirements that are specified with certain conditions. [2]

System testing
System testing also follows the black-box testing methodology in which the entire system is tested comparing the requirement specification of the project. The person carrying out the tests does not have to have any knowledge of the inner design of the system. [2]

Beta testing

Beta testing follows the black-box testing methodology to find out bugs or defects in a system. It is done outside of the development team. Hence it is also known as third party testing. [2]

Non- functional Testing

Non-functional testing involves testing of the non-functional attributes of the software per several criteria. [2] Non-functional testing is performed in all software testing levels which check memory leaks, robustness or performance of the software under test. Based on attributes, non-functional testing is classified into the following categories:

Performance testing

Performance testing is done to check the quality and reliability of the software. Testing is done in various combinations and/or scenarios to check the speed, system load, time of response and so on. [2]

Stability testing

Stability testing is done to prevent software from crashes. It tests various unacceptable conditions that may be responsible for the malfunctioning of the software. It is also known as endurance testing. [2]

Usability testing

Usability testing is done to test the usability of user interfaces. The input/output screens are printed and report screen is tested and feedback is collected to determine the usability of the software. [2]

Security testing

The purpose of security testing is to ensure that confidential data can be accessed only by limited users and are out of reach of irrelevant system users. [2]

3 Introduction to Test Automation

Companies tend to adopt automation to shorten the delivery schedule and decrease the production cost. [8] Manual testing is time consuming and requires many software testers whereas a single tester can cover many test cases in a limited time with the help of an automated testing environment.

Test automation is fundamentally different from manual testing. Test automation is the automated process in which software is used to manage the execution of test cases, comparison of actual outcomes to expected outcomes, setting up of test conditions and other test reporting functions. [9] The application of software testing tools and processes is to execute test results in order to improve the quality of the software being tested. The testing management, test design, test execution and result evaluation in an automated way are the processes involved in a test automation process [10] Test automation helps save time, increase efficiency and accuracy and also reduce the cost of development.

3.1 Test Automation Framework

The test framework is like an application architecture: it outlines the overall structure for the automated test environment, defines common functions, standard tests, provides templates for test structure, and spells out the ground rules for how tests are named, documented and managed, leading to a maintainable and transferable test library. [15, 9]

A Test Automated Framework is a set of theories defined by the software tester, ideas involved and tools used for software testing. While designing the framework, one should insure that the correct logic is implemented in the right way to establish an automated generating test environment by keeping in mind its maintenance cost and efficiency. It cannot be considered as a direct rule because various factors need to be considered while designing and planning the system. [15, 9]

Automated software testing provides a solution to several problems at once which is impossible with a manual or regression testing process. When the right test automation framework is implemented, it offers many advantages and eases the overall development process. The regression tests need to be modified when the version of the application is modified. When such regression tests are automated, with a minimum effort of manual configuration, the same regression tests can be used in different versions of the application. [15, 9]

The development of the framework itself is a complete software development process which includes planning, requirement management and implementation processes with the use of right scripts, tools and procedures required. Therefore, a good planning of a framework before designing a Test Automation Framework is essential and one should have a clear understanding of the result that should be automated, the reason why it is being automated and the expected outputs and goals of the proposed test automation framework. [15, 9] Later in this paper, in chapters 4 and 5, the actual framework that is used for developing test cases will be discussed.

4 OMI Reference Model

IoT which stand for Internet of Things is the network of embedded physical objects with sensors, software, electronics and network connection which can exchange data. The definition of OMI (Open Messaging Interface) is fully based on IoT network where physical objects have some degrees of “Artificial Intelligence”. The intelligence ranges from a simple barcode identifier to smart houses which have smart sensing capacities, powerful processing, memory, communication skill and so forth. [11]

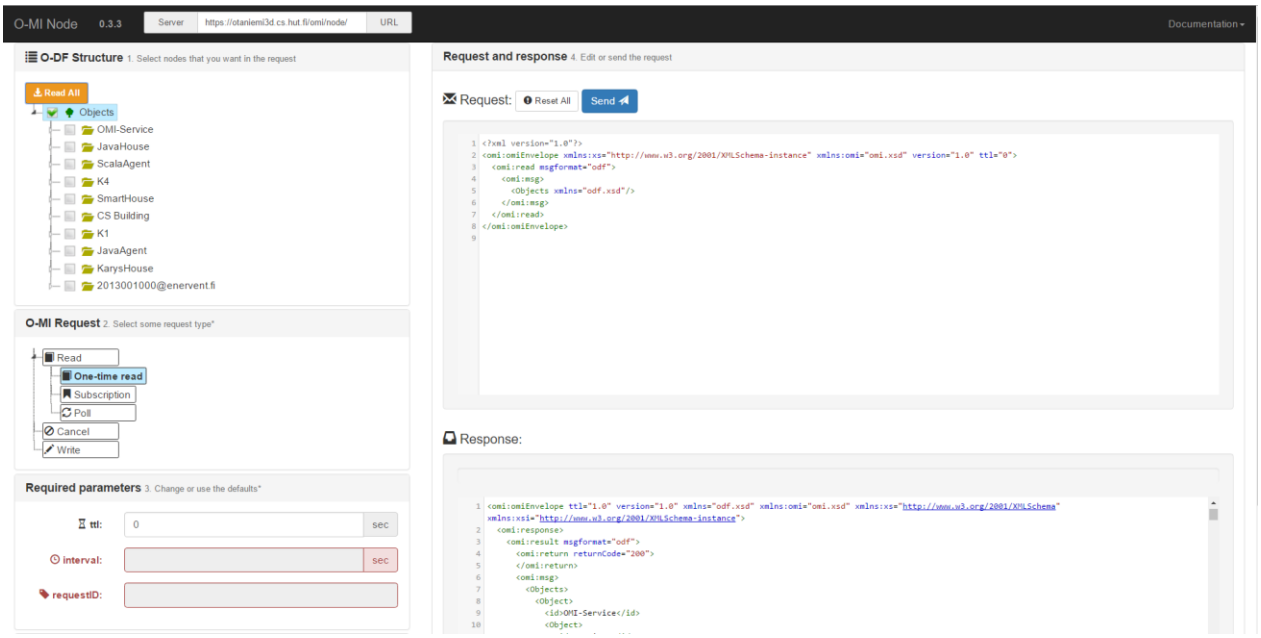
The life of OMI is the Messaging interface that connects smart objects with information systems and forms a communication network to exchange information between them. The OMI standards intend to focus on the lifecycle of “anything”. The word “anything” refers to embedded objects, networks as well as human beings. Therefore, the OMI Messaging Interface itself is a wide network cloud which has been specified in as generic way as possible. [12]

In the OMI world, the communication between the products and backend system is done by interchange of messages between the nodes. Figure 4.1 shows the OMI cloud where the HTTP protocol is used for the transmission of HTML coded information. The Open Messaging interface (OMI) for IoT has the same purpose as that of HTTP for the Internet. The exchanged data can be sensor readings, lifecycle events, reading of historic data, notifications of new data, changes in existing data and so forth. For transporting, XML is the common text-based payload but other formats such as JSON, CSV etc. may be used. The OMI nodes are not pre-defined; it follows a peer-to-peer communication model. [13]

4.1Interconnection of Devices using OMI

(Image source: O-MI, an Open Group Internet of Things Standard)

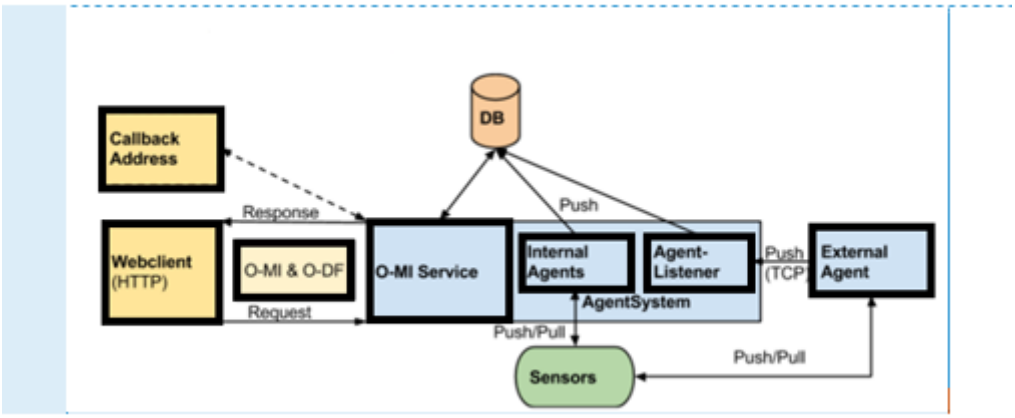
The conceptual framework as shown in Figure 4.2 is the OMI Messaging Interface.



4.2OMI messaging Interface

The basic read and write operations (fully described in the next chapter) are used for querying and writing information (such as GET/POST in HTTP or get/set-methods in JAVA). The subscription concept-t is considered as a cornerstone of the QLM Messaging Interface.

Figure 4.3 shows the design Pattern or architecture of the OMI node. HTTP or HTTPS communication protocol with POST functionality is the recommended communication protocol. OMI enabled systems should implement such a communication method.



4.3OMI architecture

As shown in Figure 4.3, OMI messages can be exchanged through any means of communication, storage media or anything that have access to handle XML file format. OMI messages can also communicate using WSDL/SOAP protocols but all systems need not to implement WSDL/SOAP as protocol for OMI messages. [14]

4.1 Communication Protocols of OMI Model

In the OMI model, communication protocols are further divided as various interfaces. Those interfaces are HTTP, RESTful and WSDL/SOAP interfaces which are explained below:

4.1.1 HTTP Interface

The OMI Messaging Interface communicates using plain HTTP interface. The requests and callback messages are sent using HTTP POST messages, where the parameter that contains the OMI message is called “msg”.

Figure 4.4 shows how a minimal OMI message can be sent with HTTP POST with the help of Unix “curl” utility. The URL of the OMI node is “http://dialog.hut.fi/qlm/”. When the message is received correctly, a correct reply as shown in Figure 4.5 should be received. [12]

```
curl http://dialog.hut.fi/qlm/ --data msg="<?xml version="1.0"
encoding="UTF-8"?><qlm:qlmEnvelope version="1.0"
 ttl="10">
</qlm:read>
</qlm:qlmEnvelope>"
```

4.4 Unix “curl” to send minimal OMI message

The GET operation of HTTP with “msg” parameter that has OMI request may be implemented by OMI nodes. This is not recommended for OMI requests due to the limitations in the size of HTTP GET request that varies depending with the type of HTTP server. [12]

4.1.2 RESTful Interface

The OMI messaging interface can be RESTful whenever possible which signifies that messages intend to be self-contained as far as possible. Such behaviour results in OMI nodes having little information of communication and session states whenever possible. [12]

Figure 4.5 shows path for issuing HTTP GET request to OMI node at http://dialog.hut.fi/qlm/. The paths used in URLs of OMI rely on its domain. The paths can be defined as a XML schema that is in message payloads. The OMI data format or ODF, which is the “accompanying standard” of OMI, provides a payload format based on the similarities in requirements and target applications like in the OMI messaging interface. [12]

```
wget http://dialog.hut.fi/qlm/Objects/
```

4.5 Issuing HTTP GET request to OMI node at http://dialog.hut.fi/qlm/

Figure 4.6 shows ODF format for object list which include objects, infoitems and meta data. OMI nodes are always required to implement the URL-based HTTP GET mechanism so that it can retrieve a list of available information according to the designed class hierarchy.

```
<Objects>
        <Object>
                <id>Refrigerator123</id>
        </Object>
        <Object>
                <id>HeatingController321</id>
        </Object>
        <Object>
                <id>WeatherStation651</id>
        </Object>
</Objects>
```

4.6 ODF format for Object list

Unlike most RESTful specifications, OMI does not use PUT and DELETE methods in order to avoid an explicit link between the OMI messaging interface and HTTP protocol so that it would not go

against the functional requirements designed for the messaging interface.

4.1.3 WSDL/SOAP Interface

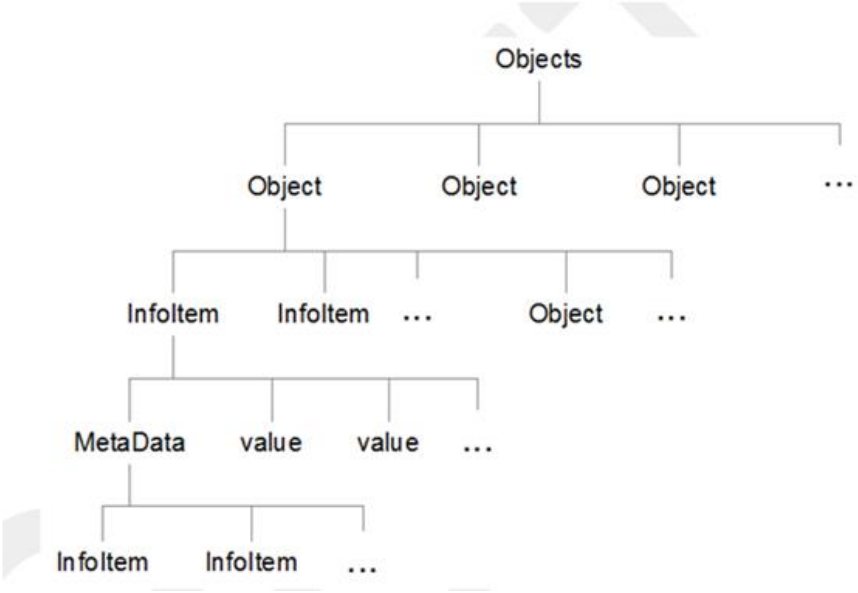
When the OMI node uses WSDL/SOAP protocols, the WSDL interface defines a function called “notify”. Usually, the “notify” function is used whenever a system wants to communicate OMI interface based messages with the help of SOAP communication. The “notify” function can be used in requests as well as responses just like HTTP POST protocol. If WSDL/SOAP is used as communicating messages, the call-back URL should point to a WSDL service with a simple “notify” function. [12].

“notify” function: “String notify(String msg)”
The “notify” function should return XML structure complying with the OMI Messaging Interface XML Schema.

4.2 OMI Messaging Format

The OMI messaging format (OMI-MF) is a simple yet extensible ontology which is specified using an “xml” schema for representation of any objects along with their information which are needed for the exchange of information in the OMI cloud.

Figure 4.8 is the OMI Data Format with a hierarchy of “Object” at the top most layer, followed by “Sub Objects”, ”Info items” and their “Metadata”.



4.7 OMI Messaging Format XML element hierarchy

As shown in figure 4.8, The “Object" element can contain any number of “Object" sub-elements. “Object" elements can have any number of properties, called “InfoItem", as well as “Object" sub-elements. The resulting Object tree can contain any number of levels. [12] “Type” is the most important attribute of an Object which specifies the kind of object. “udef”, which is an optional attribute may be used to specify the object class using the “Universal Data Element Framework (UDEF)”, which is a standard way of indexing enterprise information. [14]

UDEF: UDEF is recommended to be included when appropriate but taxonomies may be used instead. Each Object has a sub-element called “id” which is the identity of the Object. The “id” should always be globally unique or at least unique for specific application or network. [15] InfoItems contain the following sub-elements (optional):

Name: The additional name given to the same InfoItem so that they may be used if the same InfoItem is given different names in different organizations. [15]

Description: The text that gives information about the InfoItem for human users.
MetaData: The sub-element which gives meta-data details of the InfoItem. The details/information include value type and unit. [15]

Value: The sensor calculated data possibly with timestamps. [15]

Although these sub-elements can be included in the InfoItem, it is usually not recommended. MetaData is usually requested for a single time only when checking a previously not known InfoItem. The MetaData element may include arbitrary numbers of element. MetaData elements also belong to the InfoItem type because their syntax is similar to that of Object InfoItems even if MetaData and InfoItems are conceptually different from Object and InfoItems. The “display” element is also considered as MetaData but it can be considered as a “free-form” text element for user interface and debugging purposes. [14]

In IoT cloud, the information of any devices can be stored and transferred to any information systems. Due to the hierarchical nature of the OMI data format and its unique identify of IoT objects with “id” as sub-element, it can perform RESTful, and URL based, information discovery and Queries as in Figure 4.10 which work through Unix “wget” utility. [15]

4.3 Messaging Objects

The messaging object of the OMI body consists of Read Request, Write Request, Cancel Request and Response Element which are further explained below:

4.3.1 Read Request

Read requests are used to request data or to subscribe data. It should be answered with either requested data or with an error message in exceptional cases. The response element is responsible for data persistency of requests. Read request has two features, One Time Read and Subscription Read.

Table 4.1 shows the attributes of Read Request which are explained below:

Attribute	Description
callback	-may be present for subscription requests -should not be present for immediate requests - URL to a call-back service
msgformat	-should be present
targetType	-may be present.

	- defined types are "node" or "device"
interval	-shall be present for subscription requests -shall not be present for other requests. -positive value subscriptions should be answered as interval between responses(in seconds) - responding system should answer as close to the interval time -if data is not available, the request should be answered with an error for those intervals -value (0) subscriptions may be answered as often as data can be read from the target -value (-1) subscription should be answered every time the target pushes a new value -value (-2) subscriptions should be answered every time the subscribed target connects to a OMI node
oldest	-may be present -should retrieve the oldest available historical data
begin	-may be present -should retrieve data from the begin date -if no end value is provided, then it retrieve all available data after begin value
end	-may be present -should retrieve data until the given end date -if no begin value is provided, then it retrieve all data available until end value
newest	-may be present -should retrieve the newest available number of historical data

Table 4.1 Attributes of Read request

Table 4.2 shows the child elements of Read Request which are explained below:

Child Element	Description
nodeList	-may be present if the request is predefined OMI nodes other than OMI node that received the request -list of recipient OMI nodes

requestID	-may be present if read request is sent for retrieving data from an existing subscription without a call-back address
msg	-actual payload of the request -specify what data is being requested

Table 4.2 Child elements of Read request

4.3.2 Write Request

Write request is used to write various data to systems that may be from sensor values to information extracted from databases. The response should be a confirmation that defines if the write request was successful or with an error message to the given call-back address. The responding system is responsible for data persistency of requests. The request shall be answered either with an OMI Messaging Interface compatible message or with an HTTP error code during failed request on communication connection level.

Table 4.3 shows the attributes of Write Request which are explained below:

Attribute	*Description*
ttl	-ttl (Time To Live) shall be present -request should be written to the targets or answered with an error response before the TTL expires - response may contain both error messages and confirmation messages if the write request was successful for some targets and failed for others
callback	-shall be present for call-back requests -shall not be present for immediate requests -shall define a URL to a call-back service
msgformat	-should be present
targetType	-may be present - defined types are "node" or "device"

Table 4.3 Attributes of Write Request

Table 4.4 shows the Child Elements of Write Request which are explained below:

Child Element	Description
nodeList	-may be present if the request is predefined OMI nodes other than OMI node that received the request -list of recipient OMI nodes
requestID	-list of request IDs of earlier requests -may be present - current specification does not define how list of requestIDs should be used in write requests -future specifications may include recommendations for this feature
msg	-actual payload of the request -specify what data is being written along with corresponding values

Table 4.4 Child Elements of Write Request

Cancel Request

The cancel request should be answered with a confirmation message that states either that the cancellation of a request was successful or with an error message if it was not successful. All cancel requests should be directed to node recipients. Such request defines the IDs of the request to be cancelled. The response should have an OMI Messaging Interface compatible message or an HTTP error code in case the request failed on communication connection level.

Table 4.5 shows the Child Elements of Cancel Request which are explained below:

Child Element	Description
requestID	-shall be present -list of request IDs to be cancelled
List of Recipient Nodes	-may be present if the requests are cancelled at pre-defined OMI nodes

Table 4.5 Child Elements of Cancel Request

4.3.3 Response Element

The response element may vary with the type of request (read / write). However, the generic structure of the response element must be the same as the structure defined by the OMI. All responses shall contain at least one result. Every result object should have a return object that indicates if the request was successful or not by using HTTP status codes, “200” if successful.

Table 4.6 shows the attributes of Response Element which are explained below:

Attribute (result)	Description
msgformat	-should be present
targetType	-may be present -defined types are "node" or "device" -the value of targetType indicates if requested value come from the device directly or it is the last data known by the node
format (requestId)	-may contain text string that specifies the format of the request id used, such as “UDEF, …”

Table 4.6 Attributes of Response Element

Table 4.7 shows the Child Elements of Write Request which are explained below:

Child Element	Description
return	-shall be present -return code of the request -may contain a “description” attribute describing the error.
requestID	-shall be present if the message is the result of a call-back request, else shall not be present.
msg	-may be present -actual payload of the result
nodeList	-may be present -list of OMI nodes with results
omiEnvelope	-may be present -OMI envelope containing a new request from OMI node

	-enable a synchronous, “real-time” dialogue between two OMI-nodes -useful for communicating with OMI nodes behind firewalls

Table 4.7 Child Elements of Response Element

Error Handling and Test Automation of OMI Reference Implementation is discussed in the next chapter which describe the handling of OMI Messaging Interface Level errors and overall processes, tools used, design and working mechanism of the test automation environment.

5 Error Handling and Test Automation of OMI Reference Implementation

There may be several different error situations and faults in the OMI Messaging Interface communication. The errors related to communication between the OMI Messaging Interface requester and the OMI Messaging interface request handler should deal with the communication level error. Communication level errors are usually handled by the communication layer, i.e. an HTTP layer. During such errors, HTTP error messages and codes should be used. Other error messages related to communication protocol may also be used in addition to the “HTTP” error message. Similarly, the information about errors affecting the OMI Messaging Interface requests are known as OMI Messaging Interface content level errors. The handling of OMI Messaging Interface Level errors should be included in the OMI Messaging Interface responses.

The overall test automation processes of OMI reference implementation are described in a step by step manner and the test plan is made for developing automated test cases of OMI Reference Implementation which are mentioned below.

Step One: Understanding OMI Interface

Step Two: Studying Implementation Details

Step Three: Selection of Implementation Codes to be tested

Step Four: Setting of test specific properties

Step Five: Designing Test Automation Framework

Step Six: Documenting generated outputs

Step Seven: Designing and documenting test cases

5.1 Tools Used

The software / environment / tools used for building the test automation environment are listed below:

1. OMI node
   Version: 0.1.2
   Library used: - Akka 2.3.6 for threads
   - SPRAY 1.3.2 for networking

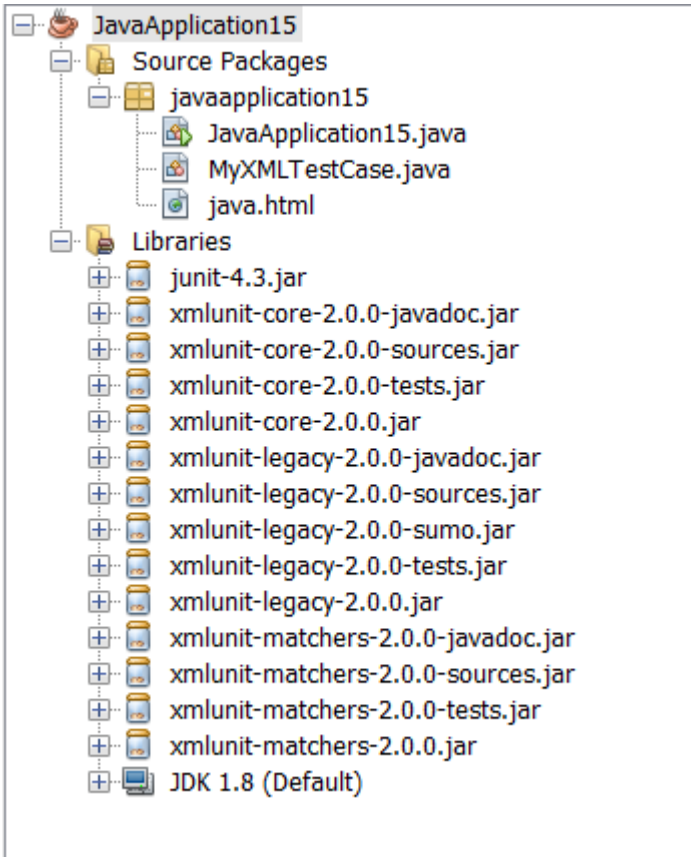
- Slick 3.0.0 for database
- Scalaxb 1.3.0 for parsing XML

2. NetBeand IDE
   Version: 8.0.1
   Library used: - junit-4.3
   - xmlunit-core-2.0.0-javadoc.jar
   - xmlunit-coore-2.0.0-sources.jar
   - xmlunit-core-2.0.0-tests.jar
   - xmlunit-core-2.0.0.jar
   - xmlunit-legacy-2.0.0-javadoc.jar
   - xmlunit-legacy-2.0.0-sources.jar
   - xmlunit-legacy-2.0.0-sumo.jar
   - xmlunit-legacy-2.0.0-tests.jar
   - xmlunit-legacy-2.0.0.jar
   - xmlunit-matchers-2.0.0-javadoc.jar
   - xmlunit-matchers-2.0.0-sources.jar
   - xmlunit-matchers-2.0.0-tests.jar
   - xmlunit-matchers-2.0.0.jar
   - JDK 1.8 (Default)

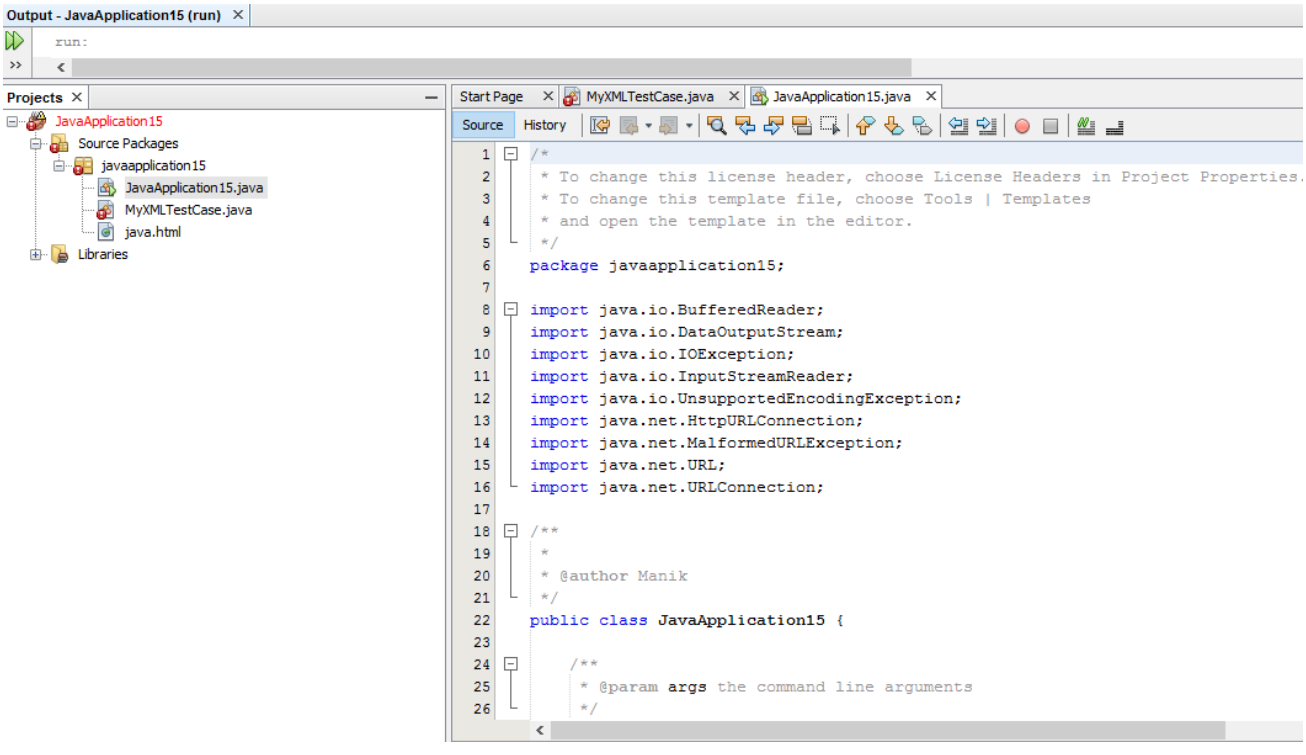
5.2 Framework Design and Working Mechanism

The whole Test Automation environment is developed by using JAVA as a main programming language in “NetBeans IDE 8.0.2”. “JavaApplication15” is a JAVA project which typically includes java files (javaApplication15.java and MyXMLTestCase.java) and java libraries as shown in “Figure 4.1”. The Java file “javaApplication15.java” include XML codes to be tested and logics used to generate test case results. The Java file “MyXMLTestCase,java” acts as a backend file for most of the functions of “JavaApplication15.java”.

The whole structure of the built environment is shown in Figure 6.1 and Figure 6.2.



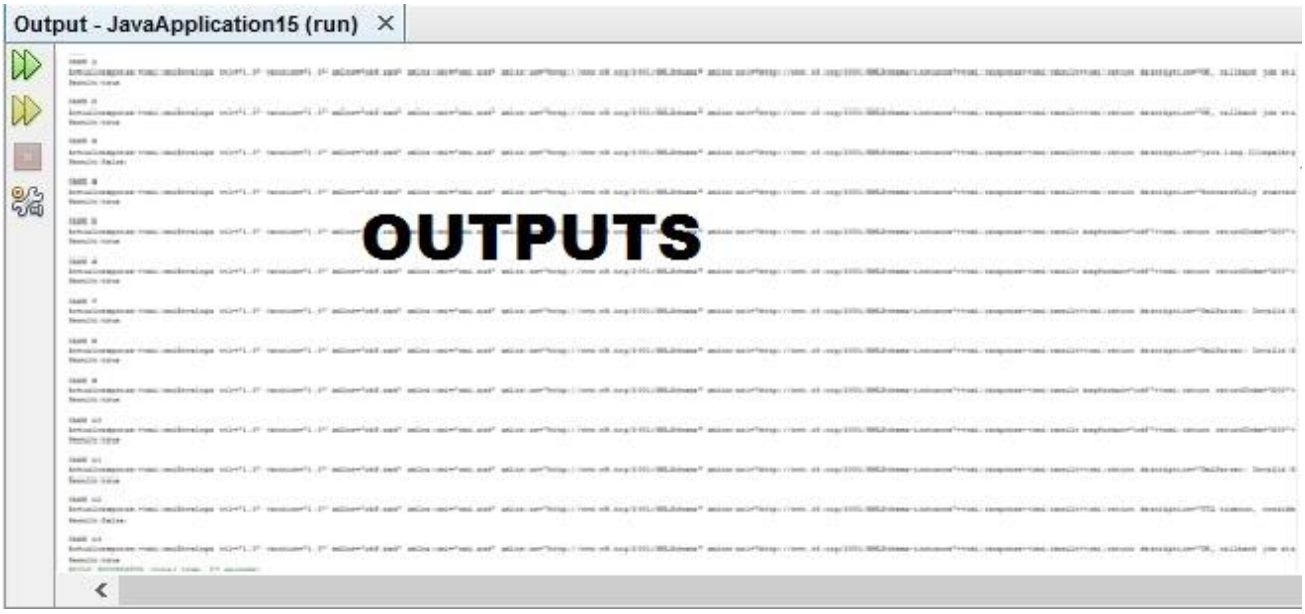
5.1 JAVA Application file format for OMI Testing



5.2 Structure of Testing framework

The testing “XML codes” are included and with the help of logics tests are executed. When the program is executed, it basically gives two outputs, the testing result in an output section and a separate table of test cases. It is not a straightforward and easy process. The testing mechanism involves numbers of logical operations for choosing the right set of files, example: callback functions to connect with the OMI node, using comparisons, test data files: their source and destination, designing the displayed output and so forth. After that, based on the given information, and output is generated which follows the predefined structure.

Figure 6.3 and figure 6.4 are two generate outputs which shows the outputs of test data that have been tested by the system.



5.3 Output 1

TEST CASES	RESULTS
Case 1	*true*
Case 2	*true*
Case 3	*false*
Case 4	*true*
Case 5	*true*
Case 6	*true*
Case 7	*true*
Case 8	*true*
Case 9	*true*
Case 10	*true*
Case 11	*true*
Case 12	*false*
Case 13	*true*

5.4 Output 2

The next chapter include the documentation of test cases and the outputs of the tests generated by the system.

6 Test Cases and Result analysis

This chapter shows the clear documentation of thirteen test cases and the results of the test data. According to the test plan and test automation of the OMI Reference model, the outputs that have been documented are as follows:

Test Case 1

Purpose:
To verify that Subscription job is started when “Callback ID” is given in “One time Read” request.

Requirement Traceability:
OMI Node is functioning from local host.

Setup:
Object/Info item/Metadata, One time Read, ttl, Call back ID

Table 7.1 shows Test Data 1*:*

Action	Launch the application
Request 1	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="0"> <omi:read msgformat="odf" callback="http://localhost:8080/"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 1	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result>

	<omi:return description="OK, callback job started" returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 1	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OK, callback job started" returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 1	Pass

Table 6.1 Test Data 1

As shown in Table 7.1, the result for Test Data 1 is Pass.

Test Case 2

Purpose:

To verify that Subscription is started when “Callback ID” is given in “Subscription Read” request.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Metadata, Subscription Read, ttl, Call back ID

Table 7.2 shows Test Data 2*:*

Action	Launch the application
Request 2	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="0"> <omi:read msgformat="odf" callback="http://localhost:8080/"> <omi:msg> <Objects xmlns="odf.xsd">

	<Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 2	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OK, callback job started" returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 2	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OK, callback job started" returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 2	Pass

Table 6.2 Test Data 2

As shown in Table 7.2, the result for Test Data 2 is Pass.

Test Case 3

Purpose:

To verify that Subscription started with “Interval value 0”.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, Subscription Read, ttl, Interval: 0

Table 7.3 shows Test Data 3*:*

Action	Launch the application
Request 3	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="0"> <omi:read msgformat="odf" callback="http://localhost:8080/" interval="0"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 3	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="Successfully started subscription" returnCode="200"> </omi:return> <omi:requestID>204</omi:requestID> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 3	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="java.lang.IllegalArgumentException: Negative Interval, diffrent than -1 isn't allowed. thrown when parsed." returnCode="400"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 3	Fail (does not accept Interval value other than -1)

Table 6.3 Test Data 3

As shown in Table 7.3, the result for Test Data 1 is Fail.

Test Case 4

Purpose:

To verify that Subscription started with “Interval value -1”.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, subscription Read, ttl, Interval:-1

Table 7.4 shows Test Data 4*:*

Action	Launch the application
Request 4	<?xml version=\"1.0\"?> <omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\"> <omi:read msgformat=\"odf\" interval=\"-1\"> <omi:msg> <Objects xmlns=\"odf.xsd\"> <Object> <id>SmartHouse</id> <InfoItem name=\"BackDoor\"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 4	<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\" xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <omi:response> <omi:result> <omi:return description=\"Successfully started subscription\" returnCode=\"200\"> </omi:return> <omi:requestID>3</omi:requestID> </omi:result> </omi:response> </omi:omiEnvelope>

Actual Response 4	<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\" xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <omi:response> <omi:result> <omi:return description=\"Successfully started subscription\" returnCode=\"200\"> </omi:return> <omi:requestID>3</omi:requestID> </omi:result> </omi:response> </omi:omiEnvelope>
Result 4	Pass

Table 6.4 Test Data 4

As shown in Table 7.4, the result for Test Data 4 is Pass.

Test Case 5

Purpose:

To verify that a given number of the oldest “InfoItem” details is displayed when Oldest value is requested.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, One time read, ttl, Oldest

Table 7.5 shows Test Data 5*:*

Action	Launch the application
Request 5	<?xml version=\"1.0\"?>\n" <omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\"> <omi:read msgformat=\"odf\" oldest=\"10\"> <omi:msg> <Objects xmlns=\"odf.xsd\"> <Object> <id>SmartHouse</id> <InfoItem name=\"BackDoor\"> <MetaData/>

	</InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 5	<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\" xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <omi:response> <omi:result msgformat=\"odf\"> <omi:return returnCode=\"200\"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name=\"BackDoor\"> <value unixTime=\"1458482049\" dateTime=\"2016-03-20T15:54:09.299+02:00\" type=\"xs:double\">1.0557321203738752</value> <value unixTime=\"1458482059\" dateTime=\"2016-03-20T15:54:19.300+02:00\" type=\"xs:double\">0.9394669286950502</value> <value unixTime=\"1458482069\" dateTime=\"2016-03-20T15:54:29.300+02:00\" type=\"xs:double\">1.0072965947229988</value> <value unixTime=\"1458482079\" dateTime=\"2016-03-20T15:54:39.301+02:00\" type=\"xs:double\">0.9012483219313834</value> <value unixTime=\"1458482089\" dateTime=\"2016-03-20T15:54:49.302+02:00\" type=\"xs:double\">0.9032312518537794</value> <value unixTime=\"1458482099\" dateTime=\"2016-03-20T15:54:59.303+02:00\" type=\"xs:double\">1.0297121845334147</value> <value unixTime=\"1458482109\" dateTime=\"2016-03-20T15:55:09.303+02:00\" type=\"xs:double\">1.2324325289126734</value> <value unixTime=\"1458482119\" dateTime=\"2016-03-20T15:55:19.304+02:00\" type=\"xs:double\">1.1624208943389351</value> <value unixTime=\"1458482129\" dateTime=\"2016-03-20T15:55:29.304+02:00\" type=\"xs:double\">1.107521763090082</value> <value unixTime=\"1458482139\" dateTime=\"2016-03-20T15:55:39.305+02:00\" type=\"xs:double\">1.1026626159272843</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 5	<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\" xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <omi:response>

	<omi:result msgformat=\"odf\"> <omi:return returnCode=\"200\"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name=\"BackDoor\"> <value unixTime=\"1458482049\" dateTime=\"2016-03-20T15:54:09.299+02:00\" type=\"xs:double\">1.0557321203738752</value> <value unixTime=\"1458482059\" dateTime=\"2016-03-20T15:54:19.300+02:00\" type=\"xs:double\">0.9394669286950502</value> <value unixTime=\"1458482069\" dateTime=\"2016-03-20T15:54:29.300+02:00\" type=\"xs:double\">1.0072965947229988</value> <value unixTime=\"1458482079\" dateTime=\"2016-03-20T15:54:39.301+02:00\" type=\"xs:double\">0.9012483219313834</value> <value unixTime=\"1458482089\" dateTime=\"2016-03-20T15:54:49.302+02:00\" type=\"xs:double\">0.9032312518537794</value> <value unixTime=\"1458482099\" dateTime=\"2016-03-20T15:54:59.303+02:00\" type=\"xs:double\">1.0297121845334147</value> <value unixTime=\"1458482109\" dateTime=\"2016-03-20T15:55:09.303+02:00\" type=\"xs:double\">1.2324325289126734</value> <value unixTime=\"1458482119\" dateTime=\"2016-03-20T15:55:19.304+02:00\" type=\"xs:double\">1.1624208943389351</value> <value unixTime=\"1458482129\" dateTime=\"2016-03-20T15:55:29.304+02:00\" type=\"xs:double\">1.107521763090082</value> <value unixTime=\"1458482139\" dateTime=\"2016-03-20T15:55:39.305+02:00\" type=\"xs:double\">1.1026626159272843</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Result 5	Pass

Table 6.5 Test Data 5

As shown in Table 7.5, the result for Test Data 5 is Pass.

Test Case 6

Purpose:

To verify that the given number of the latest “InfoItem” details is displayed when Newest value is requested.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/ Meta data, One time read, ttl, Newest

Table 7.6 shows Test Data 6*:*

Action	Launch the application
Request 6	<?xml version=\"1.0\"?>\n" <omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\"> <omi:read msgformat=\"odf\" newest=\"5\"> <omi:msg> <Objects xmlns=\"odf.xsd\"> <Object> <id>SmartHouse</id> <InfoItem name=\"FrontDoor\"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 6	<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\" xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <omi:response> <omi:result msgformat=\"odf\"> <omi:return returnCode=\"200\"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name=\"FrontDoor\"> <value unixTime=\"1458748590\" dateTime=\"2016-03-23T17:56:30.153+02:00\" type=\"xs:double\">-2.914536227845562</value> <value unixTime=\"1458748600\" dateTime=\"2016-03-23T17:56:40.153+02:00\" type=\"xs:double\">-3.063640707087419</value> <value unixTime=\"1458748610\" dateTime=\"2016-03-23T17:56:50.155+02:00\" type=\"xs:double\">-3.070800589128009</value> <value unixTime=\"1458748620\" dateTime=\"2016-03-23T17:57:00.156+02:00\" type=\"xs:double\">-3.208021098173108</value> <value unixTime=\"1458748630\" dateTime=\"2016-03-23T17:57:10.157+02:00\"

	type=\"xs:double\">-3.2234440046016415</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 6	<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\" xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> <omi:response> <omi:result msgformat=\"odf\"> <omi:return returnCode=\"200\"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name=\"FrontDoor\"> <value unixTime=\"1458748590\" dateTime=\"2016-03-23T17:56:30.153+02:00\" type=\"xs:double\">-2.914536227845562</value> <value unixTime=\"1458748600\" dateTime=\"2016-03-23T17:56:40.153+02:00\" type=\"xs:double\">-3.063640707087419</value> <value unixTime=\"1458748610\" dateTime=\"2016-03-23T17:56:50.155+02:00\" type=\"xs:double\">-3.070800589128009</value> <value unixTime=\"1458748620\" dateTime=\"2016-03-23T17:57:00.156+02:00\" type=\"xs:double\">-3.208021098173108</value> <value unixTime=\"1458748630\" dateTime=\"2016-03-23T17:57:10.157+02:00\" type=\"xs:double\">-3.2234440046016415</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Result 6	Pass

Table 6.6 Test Data 6

As shown in Table 7.6, the result for Test Data 6 is Pass.

Test Case 7

Purpose:

To verify that “InfoItem” details are displayed from the given “Begin date and time” to the latest available InfoItem details.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, One time read, ttl, Begin

Table 7.7 shows Test Data 7*:*

Action	Launch the application
Request 7	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="0"> <omi:read msgformat="odf" begin="2016-03-24T09:54:19.127Z"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 7	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result msgformat="odf"> <omi:return returnCode="200"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <value unixTime="1458813259" dateTime="2016-03-24T11:54:19.555+02:00"

	type="xs:double">1.6292965387782696</value> <value unixTime="1458813259" dateTime="2016-03-24T11:54:19.555+02:00" type="xs:double">1.6292965387782696</value> <value unixTime="1458813259" dateTime="2016-03-24T11:54:19.555+02:00" type="xs:double">1.6292965387782696</value> <value unixTime="1458813269" dateTime="2016-03-24T11:54:29.560+02:00" type="xs:double">1.5475976341565763</value> <value unixTime="1458813269" dateTime="2016-03-24T11:54:29.560+02:00" type="xs:double">1.5475976341565763</value> <value unixTime="1458813269" dateTime="2016-03-24T11:54:29.560+02:00" type="xs:double">1.5475976341565763</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 7	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result msgformat="odf"> <omi:return returnCode="200"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <value unixTime="1458813259" dateTime="2016-03-24T11:54:19.555+02:00" type="xs:double">1.6292965387782696</value> <value unixTime="1458813259" dateTime="2016-03-24T11:54:19.555+02:00" type="xs:double">1.6292965387782696</value> <value unixTime="1458813259" dateTime="2016-03-24T11:54:19.555+02:00" type="xs:double">1.6292965387782696</value> <value unixTime="1458813269" dateTime="2016-03-24T11:54:29.560+02:00" type="xs:double">1.5475976341565763</value> <value unixTime="1458813269" dateTime="2016-03-24T11:54:29.560+02:00" type="xs:double">1.5475976341565763</value> <value unixTime="1458813269" dateTime="2016-03-24T11:54:29.560+02:00" type="xs:double">1.5475976341565763</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>

Result 7	Pass

Table 6.7 Test Data 7

As shown in Table 7.7, the result for Test Data 7 is Pass.

Test Case 8

Purpose:

To verify that “InfoItem” details are displayed from the given “Begin date and time” till the given “End date and time”.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, One time read, ttl, Begin, End

Table 7.8 shows Test Data 8*:*

Action	Launch the application
Request 8	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0"> <omi:read msgformat="odf" end="2016-03-24T09:54:19.127Z" begin="2016-03-24T09:54:19.127Z"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 8	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result>

	<omi:return description="OmiParser: Invalid XML, schema failure: cvc-complex-type.4: Attribute 'ttl' must appear on element 'omi:omiEnvelope'." returnCode="400"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 8	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OmiParser: Invalid XML, schema failure: cvc-complex-type.4: Attribute 'ttl' must appear on element 'omi:omiEnvelope'." returnCode="400"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 8	Pass

Table 6.8 Test Data 8

As shown in Table 7.8, the result for Test Data 8 is Pass.

Test Case 9

Purpose:

To verify that new Infoitem details are displayed within given ttl value (if available).

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item, One time read, ttl:10

Table 7.9 shows Test Data 9*:*

Action	Launch the application
Request 9	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="10"> <omi:read msgformat="odf"> <omi:msg> <Objects xmlns="odf.xsd">

	<Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"/> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>
Expected Response 9	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result msgformat="odf"> <omi:return returnCode="200"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <value unixTime="1458814590" dateTime="2016-03-24T12:16:30.073+02:00" type="xs:double">0.5613194999203899</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 9	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result msgformat="odf"> <omi:return returnCode="200"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <value unixTime="1458814590" dateTime="2016-03-24T12:16:30.073+02:00" type="xs:double">0.5613194999203899</value> </InfoItem> </Object> </Objects> </omi:msg>

	</omi:result> </omi:response> </omi:omiEnvelope>
Result 9	Pass

Table 6.9 Test Data 9

As shown in Table 7.9, the result for Test Data 9 is Pass.

Test Case 10

Purpose:

To verify that “InfoItem” details are displayed from the given “Begin date and time” till the given “End date and time” (ttl: -1 is infinity).

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, One time read, Begin, End, ttl: -1

Table 7.10 shows Test Data 10*:*

Action	Launch the application
Request 10	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="-1"> <omi:read msgformat="odf" end="2016-03-24T09:34:38.851Z" begin="2016-03-24T09:34:00.000Z"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>

Expected Response 10	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result msgformat="odf"> <omi:return returnCode="200"> </omi:return> <omi:msg> <Objects> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <value unixTime="1458812049" dateTime="2016-03-24T11:34:09.049+02:00" type="xs:double">2.361104579336621</value> <value unixTime="1458812049" dateTime="2016-03-24T11:34:09.049+02:00" type="xs:double">2.361104579336621</value> <value unixTime="1458812049" dateTime="2016-03-24T11:34:09.049+02:00" type="xs:double">2.361104579336621</value> <value unixTime="1458812059" dateTime="2016-03-24T11:34:19.052+02:00" type="xs:double">2.2441146426723844</value> <value unixTime="1458812059" dateTime="2016-03-24T11:34:19.052+02:00" type="xs:double">2.2441146426723844</value> <value unixTime="1458812059" dateTime="2016-03-24T11:34:19.052+02:00" type="xs:double">2.2441146426723844</value> <value unixTime="1458812069" dateTime="2016-03-24T11:34:29.060+02:00" type="xs:double">2.2819006321774853</value> <value unixTime="1458812069" dateTime="2016-03-24T11:34:29.060+02:00" type="xs:double">2.2819006321774853</value> <value unixTime="1458812069" dateTime="2016-03-24T11:34:29.060+02:00" type="xs:double">2.2819006321774853</value> </InfoItem> </Object> </Objects> </omi:msg> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 10	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="Such item/s not found." returnCode="404"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>

Result 10	Fail (does not allow ttl -1)

Table 6.10 Test Data 10

As shown in Table 7.10, the result for Test Data 10 is Pass.

Test Case 11

Purpose:

To verify that Invalid response is displayed when ttl value is empty.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/Meta data, One time read

Table 7.11 shows Test Data 11*:*

Action	Launch the application
Request 11	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance xmlns:omi="omi.xsd" version="1.0"> <omi:read msgformat="odf" end="2016-03-23T16:05:00.000Z" begin="2016-03-22T22:00:00.000Z" callback="http://localhost:8080/"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>SmartHouse</id> <InfoItem name="FrontDoor"> <MetaData/> </InfoItem> </Object> </Objects> </omi:msg> </omi:read> </omi:omiEnvelope>

Expected Response 11	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi=" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OmiParser: Invalid XML, schema failure: cvc-complex-type.4: Attribute 'ttl' must appear on element 'omi:omiE returnCode="400"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 11	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi=" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OmiParser: Invalid XML, schema failure: cvc-complex-type.4: Attribute 'ttl' must appear on element 'omi:omiE returnCode="400"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 11	Pass

Table 6.11 Test Data 11

As shown in Table 7.11, the result for Test Data 11 is Pass.

Test Case 12

Purpose:

To verify that either response or error message is displayed before ttl expires when “Write” job is requested.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/ Meta data, Write, ttl: 0/-1/positive value (in this case ttl:10)

Table 7.12 shows Test Data 12:

Action	Launch the application
Request 12	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="10"> <write xmlns="omi.xsd" msgformat="odf"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>K1</id> <InfoItem name="K1 Building"/> </Object> </Objects> </omi:msg> </write> </omi:omiEnvelope>
Expected Response 12	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>

Actual Response 12	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 12	Pass

Table 6.12 Test Data 12

As shown in Table 7.12, the result for Test Data 12 is Pass.

Test Case 13

Purpose:

To verify if Callback job is started in Write request when Callback ID (any valid url) is given.

Requirement Traceability:

OMI Node is functioning from local host.

Setup:

Object/Info item/ Meta data, Write, Call back ID: url

Table 7.13 shows Test Data 13*:*

Action	Launch the application
Request 13	<?xml version="1.0"?> <omi:omiEnvelope xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xmlns:omi="omi.xsd" version="1.0" ttl="0"> <write xmlns="omi.xsd" msgformat="odf" callback="http://localhost:8080/"> <omi:msg> <Objects xmlns="odf.xsd"> <Object> <id>K1</id> <InfoItem name="K1 Building"> <MetaData/> </InfoItem> </Object> </Objects>

	</omi:msg> </write> </omi:omiEnvelope>
Expected Response 13	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <omi:response> <omi:result> <omi:return description="OK, callback job started" returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Actual Response 13	<omi:omiEnvelope ttl="1.0" version="1.0" xmlns="odf.xsd" xmlns:omi="omi.xsd" xmlns:xs="ht <omi:response> <omi:result> <omi:return description="OK, callback job started" returnCode="200"> </omi:return> </omi:result> </omi:response> </omi:omiEnvelope>
Result 13	Pass

Table 6.13 Test Data 13

As shown in Table 7.13, the result for Test Data 13 is Pass.

The main objective in this thesis was to design and build an automated testing framework that improves the testing procedure of the OMI model and shortens the entire development life cycle by making the testing process relatively fast. The designed testing framework enabled the automation of OMI test cases and improved the performance of the software by helping in the debugging process. The automated testing also minimized the errors that are common during manual testing.

The final step was taking sample automated test cases and manual test cases and measuring the time taken by the manual testing and automated testing. The comparison was done to demonstrate the benefits of automated testing over manual testing. Finally, it was concluded that test case automation provides a fast way of running test cases that are independent of complexity and avoid possible human errors.

7 Conclusion

This thesis has demonstrated the test case automation framework of “OMI Reference Implementation” for the improvement of a software testing process and its impact on the efficiency of the development and debugging the life-cycle of the OMI model. This thesis is also the abstraction of an automated testing tool/platform to be designed.

In the automatic testing platform, when the script is written, the testers can use the test suites and testing will take place in each line of the script. The chance of missing any key part of testing is very rare unless there are exceptional cases that are not defined. A clear and well-written script will have complete test coverage that makes automatic testing cover everything it is designed for.

The overall automated testing of the OMI Reference Implementation has been a slow process due to studying of the testing software and planning of the automation tool to be used. The advantage of designing a software testing tool instead of using the available testing tool is that it was possible to include the testing preferences according to personal preferences.

In addition to the advantages, some of the OMI test cases could not be automated because the tool was unable to perform high level testing. In such cases, testing tools such as Selenium, AutoTester could be used. The development or further improvement of this tool can be extended to support such performances.

8 References

[1] S. Haile, “Automation of Test Cases for Web Applications,” 14 September, 2011. [Online]. Available:http://theseus.fi/bitstream/handle/10024/57876/Alazar_BEng_Thesis.pdf?sequence=1

[2] S. A. Ali, “Continuous Integration and Test Automation for Symbian Device Software,” Helsinki, 2010

[3] J. O. Paul Ammann, “Introduction to Software Testing”, 2008

[4] C. K. J. M. B. Bret Pettichord, “Lessons Learned in Software Testing”, 2002

[5] D. G. Mark Fewster, “Software Test Automation”, 2000

[6] G. J. Myers, “The Art of Software Testing, 2004

[7] D. Graham, E. v. Veenendaal, I. Evans and R. Black, “Foundations of Software Testing”

[8] L. G. Hayes, “The Automated Testing Handbook”, 2004

[9] A. Kolawa and D. Huizinga, “ Automated Defect Prevention: Best Practices in Software”, October 2017

[10] Kent Beck, “Test-Driven Development by Example”, 2000

[11] V.M. Arshdeep Bahga, “Internet of Things: A Hans-on Approach”, 2015

[12] K. Främling, S. Kubler and A. Buda, “Universal Messaging Standards for the IoT from a Lifecycle Management Perspective”, IEEE Internet of Things Journal, vol. 1, 4 August 2014

[13] C. Pfister, “Getting Started with the Internet of Things”, 2011

[14] M. Madhikermi, “Implementation and Assessment of Quantum Lifecycle Management Messaging Standard for Internet of Things”, 2014

[15] K. Främling, T. Ala-Risku, M. Kärkkäinen, J. Holmström, ”Design Patterns for Managing Product Lifecycle Information”, 2007

Appendix 1: JavaApplication15.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication15;

import static com.sun.xml.internal.fastinfoset.alphabet.BuiltInRestrictedAlphabets.table;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.TableCellRenderer;

/**
 *
 *
 */
public class JavaApplication15 {

    /**
     * @param args the command line arguments
```

```
     */
    public static void main(String[] args) {
        Boolean result1 = false;
        String request1 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"
                + "  <omi:read msgformat=\"odf\" callback=\"http://localhost:8080/html/webclient/in-
dex.html?msg=\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";

        String actualresponse1 = (SendCallbackdata(request1, "http://localhost:8080"));
        String expectedresponse1 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return description=\"OK, callback job started\" returnCode=\"200\">"
                + "      </omi:return>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 1");
```

```
        System.out.println("Actualresponse:" + actualresponse1);


        MyXMLTestCase t1 = new MyXMLTestCase("test");
        t1.setXml(expectedresponse1, actualresponse1);
        try {
            result1 = t1.testForEquality();
            System.out.println("Result:" + result1);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result2 = false;
        String request2 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"
                + "  <omi:read msgformat=\"odf\" callback=\"http://localhost:8080/html/webclient/in-
dex.html?msg=\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";


        String actualresponse2 = (SendCallbackdata(request2, "http://localhost:8080"));
```

```
        String expectedresponse2 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return description=\"OK, callback job started\" returnCode=\"200\">\n"
                + "      </omi:return>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 2");
        System.out.println("Actualresponse:" + actualresponse2);


        MyXMLTestCase t2 = new MyXMLTestCase("test");
        t2.setXml(expectedresponse2, actualresponse2);
        try {
            result2 = t2.testForEquality();
            System.out.println("Result:" + result2);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result3 = false;
        String request3 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"
                + "  <omi:read msgformat=\"odf\" callback=\"http://localhost:8080/\" inter-
val=\"0\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
```

```
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";

        String actualresponse3 = (SendCallbackdata(request3, "http://localhost:8080"));
        String expectedresponse3 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return description=\"Successfully started subscription\" return-
Code=\"200\">\n"
                + "      </omi:return>\n"
                + "      <omi:requestID>204</omi:requestID>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 3");
        System.out.println("Actualresponse:" + actualresponse3);

        MyXMLTestCase t3 = new MyXMLTestCase("test");
        t3.setXml(expectedresponse3, actualresponse3);
        try {
            result3 = t3.testForEquality();
            System.out.println("Result:" + result3);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
```

```
        Boolean result4 = false;

        String request4 = "<?xml version=\"1.0\"?>\n"

                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"

                + "  <omi:read msgformat=\"odf\" interval=\"-1\">\n"

                + "    <omi:msg>\n"

                + "      <Objects xmlns=\"odf.xsd\">\n"

                + "        <Object>\n"

                + "          <id>SmartHouse</id>\n"

                + "          <InfoItem name=\"BackDoor\">\n"

                + "            <MetaData/>\n"

                + "          </InfoItem>\n"

                + "        </Object>\n"

                + "      </Objects>\n"

                + "    </omi:msg>\n"

                + "  </omi:read>\n"

                + "</omi:omiEnvelope>";


        String actualresponse4 = (SendCallbackdata(request4, "http://localhost:8080"));

        String expectedresponse4 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"

                + "  <omi:response>\n"

                + "    <omi:result>\n"

                + "      <omi:return description=\"Successfully started subscription\" return-
Code=\"200\">\n"

                + "      </omi:return>\n"

                + "      <omi:requestID>3</omi:requestID>\n"

                + "    </omi:result>\n"

                + "  </omi:response>\n"

                + "</omi:omiEnvelope>";

        System.out.println("CASE 4");

        System.out.println("Actualresponse:" + actualresponse4);
```

```
        MyXMLTestCase t4 = new MyXMLTestCase("test");
        t4.setXml(expectedresponse4, actualresponse4);
        try {
            result4 = t4.testForEquality();
            System.out.println("Result:" + result4);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }

        System.out.println("");
        Boolean result5 = false;
        String request5 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"
                + "  <omi:read msgformat=\"odf\" oldest=\"10\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"BackDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";

        String actualresponse5 = (SendCallbackdata(request5, "http://localhost:8080"));
        String expectedresponse5 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
```

```
                + "  <omi:response>\n"

                + "    <omi:result msgformat=\"odf\">\n"

                + "      <omi:return returnCode=\"200\">\n"

                + "      </omi:return>\n"

                + "      <omi:msg>\n"

                + "        <Objects>\n"

                + "          <Object>\n"

                + "            <id>SmartHouse</id>\n"

                + "            <InfoItem name=\"BackDoor\">\n"

                + "              <value unixTime=\"1458482049\" dateTime=\"2016-03-
20T15:54:09.299+02:00\" type=\"xs:double\">1.0557321203738752</value>\n"

                + "              <value unixTime=\"1458482059\" dateTime=\"2016-03-
20T15:54:19.300+02:00\" type=\"xs:double\">0.9394669286950502</value>\n"

                + "              <value unixTime=\"1458482069\" dateTime=\"2016-03-
20T15:54:29.300+02:00\" type=\"xs:double\">1.0072965947229988</value>\n"

                + "              <value unixTime=\"1458482079\" dateTime=\"2016-03-
20T15:54:39.301+02:00\" type=\"xs:double\">0.9012483219313834</value>\n"

                + "              <value unixTime=\"1458482089\" dateTime=\"2016-03-
20T15:54:49.302+02:00\" type=\"xs:double\">0.9032312518537794</value>\n"

                + "              <value unixTime=\"1458482099\" dateTime=\"2016-03-
20T15:54:59.303+02:00\" type=\"xs:double\">1.0297121845334147</value>\n"

                + "              <value unixTime=\"1458482109\" dateTime=\"2016-03-
20T15:55:09.303+02:00\" type=\"xs:double\">1.2324325289126734</value>\n"

                + "              <value unixTime=\"1458482119\" dateTime=\"2016-03-
20T15:55:19.304+02:00\" type=\"xs:double\">1.1624208943389351</value>\n"

                + "              <value unixTime=\"1458482129\" dateTime=\"2016-03-
20T15:55:29.304+02:00\" type=\"xs:double\">1.107521763090082</value>\n"

                + "              <value unixTime=\"1458482139\" dateTime=\"2016-03-
20T15:55:39.305+02:00\" type=\"xs:double\">1.1026626159272843</value>\n"

                + "            </InfoItem>\n"

                + "          </Object>\n"

                + "        </Objects>\n"

                + "      </omi:msg>\n"

                + "    </omi:result>\n"

                + "  </omi:response>\n"
```

```
                + "</omi:omiEnvelope>";
        System.out.println("CASE 5");
        System.out.println("Actualresponse:" + actualresponse5);

        MyXMLTestCase t5 = new MyXMLTestCase("test");
        t5.setXml(expectedresponse5, actualresponse5);
        try {
            result5 = t5.testForEquality();
            System.out.println("Result:" + result5);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result6 = false;
        String request6 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"
                + "  <omi:read msgformat=\"odf\" newest=\"5\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";

        String actualresponse6 = (SendCallbackdata(request6, "http://localhost:8080"));
```

```
        String expectedresponse6 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result msgformat=\"odf\">\n"
                + "      <omi:return returnCode=\"200\">\n"
                + "      </omi:return>\n"
                + "      <omi:msg>\n"
                + "        <Objects>\n"
                + "          <Object>\n"
                + "            <id>SmartHouse</id>\n"
                + "            <InfoItem name=\"FrontDoor\">\n"
                + "              <value unixTime=\"1458748590\" dateTime=\"2016-03-
23T17:56:30.153+02:00\" type=\"xs:double\">-2.914536227845562</value>\n"
                + "              <value unixTime=\"1458748600\" dateTime=\"2016-03-
23T17:56:40.153+02:00\" type=\"xs:double\">-3.063640707087419</value>\n"
                + "              <value unixTime=\"1458748610\" dateTime=\"2016-03-
23T17:56:50.155+02:00\" type=\"xs:double\">-3.070800589128009</value>\n"
                + "              <value unixTime=\"1458748620\" dateTime=\"2016-03-
23T17:57:00.156+02:00\" type=\"xs:double\">-3.208021098173108</value>\n"
                + "              <value unixTime=\"1458748630\" dateTime=\"2016-03-
23T17:57:10.157+02:00\" type=\"xs:double\">-3.2234440046016415</value>\n"
                + "            </InfoItem>\n"
                + "          </Object>\n"
                + "        </Objects>\n"
                + "      </omi:msg>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 6");
        System.out.println("Actualresponse:" + actualresponse6);


        MyXMLTestCase t6 = new MyXMLTestCase("test");
        t6.setXml(expectedresponse6, actualresponse6);
```

```
        try {
            result6 = t6.testForEquality();
            System.out.println("Result:" + result6);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result7 = false;
        String request7 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\">\n"
                + "  <omi:read msgformat=\"odf\" begin=\"2016-03-20T14:09:16.835Z\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"BackDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";

        String actualresponse7 = (SendCallbackdata(request7, "http://localhost:8080"));
        String expectedresponse7 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return description=\"OmiParser: Invalid XML, schema failure: cvc-com-
plex-type.4: Attribute 'ttl' must appear on element 'omi:omiEnvelope'.\" returnCode=\"400\">\n"
```

```
                + "      </omi:return>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 7");
        System.out.println("Actualresponse:" + actualresponse7);

        MyXMLTestCase t7 = new MyXMLTestCase("test");
        t7.setXml(expectedresponse7, actualresponse7);
        try {
            result7 = t7.testForEquality();
            System.out.println("Result:" + result7);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result8 = false;
        String request8 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\">\n"
                + "  <omi:read msgformat=\"odf\" end=\"2016-03-24T09:54:19.127Z\" begin=\"2016-03-
24T09:54:19.127Z\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
```

```
                + "</omi:omiEnvelope>";


        String actualresponse8 = (SendCallbackdata(request8, "http://localhost:8080"));

        String expectedresponse8 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"

                + "  <omi:response>\n"

                + "    <omi:result>\n"

                + "      <omi:return description=\"OmiParser: Invalid XML, schema failure: cvc-com-
plex-type.4: Attribute 'ttl' must appear on element 'omi:omiEnvelope'.\" returnCode=\"400\">\n"

                + "      </omi:return>\n"

                + "    </omi:result>\n"

                + "  </omi:response>\n"

                + "</omi:omiEnvelope>";

        System.out.println("CASE 8");

        System.out.println("Actualresponse:" + actualresponse8);


        MyXMLTestCase t8 = new MyXMLTestCase("test");

        t8.setXml(expectedresponse8, actualresponse8);

        try {

            result8 = t8.testForEquality();

            System.out.println("Result:" + result8);

        } catch (Exception ex) {

            System.out.println(ex.getCause());

        }

        System.out.println("");

        Boolean result9 = false;

        String request9 = "<?xml version=\"1.0\"?>\n"

                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"10\">\n"

                + "  <omi:read msgformat=\"odf\">\n"

                + "    <omi:msg>\n"

                + "      <Objects xmlns=\"odf.xsd\">\n"
```

```
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\"/>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";


        String actualresponse9 = (SendCallbackdata(request9, "http://localhost:8080"));
        String expectedresponse9 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result msgformat=\"odf\">\n"
                + "      <omi:return returnCode=\"200\">\n"
                + "      </omi:return>\n"
                + "      <omi:msg>\n"
                + "        <Objects>\n"
                + "          <Object>\n"
                + "            <id>SmartHouse</id>\n"
                + "            <InfoItem name=\"FrontDoor\">\n"
                + "              <value unixTime=\"1458814590\" dateTime=\"2016-03-
24T12:16:30.073+02:00\" type=\"xs:double\">0.5613194999203899</value>\n"
                + "            </InfoItem>\n"
                + "          </Object>\n"
                + "        </Objects>\n"
                + "      </omi:msg>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 9");
        System.out.println("Actualresponse:" + actualresponse9);
```

```
        MyXMLTestCase t9 = new MyXMLTestCase("test");
        t9.setXml(expectedresponse9, actualresponse9);
        try {
            result9 = t9.testForEquality();
            System.out.println("Result:" + result9);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result10 = false;
        String request10 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"-1\">\n"
                + "  <omi:read msgformat=\"odf\" end=\"2016-03-24T09:34:38.851Z\" begin=\"2016-03-
24T09:34:00.000Z\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";

        String actualresponse10 = (SendCallbackdata(request10, "http://localhost:8080"));
        String expectedresponse10 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
```

```
                + "  <omi:response>\n"
                + "    <omi:result msgformat=\"odf\">\n"
                + "      <omi:return returnCode=\"200\">\n"
                + "      </omi:return>\n"
                + "      <omi:msg>\n"
                + "        <Objects>\n"
                + "          <Object>\n"
                + "            <id>SmartHouse</id>\n"
                + "            <InfoItem name=\"FrontDoor\">\n"
                + "              <value unixTime=\"1458812049\" dateTime=\"2016-03-
24T11:34:09.049+02:00\" type=\"xs:double\">2.361104579336621</value>\n"
                + "              <value unixTime=\"1458812049\" dateTime=\"2016-03-
24T11:34:09.049+02:00\" type=\"xs:double\">2.361104579336621</value>\n"
                + "              <value unixTime=\"1458812049\" dateTime=\"2016-03-
24T11:34:09.049+02:00\" type=\"xs:double\">2.361104579336621</value>\n"
                + "              <value unixTime=\"1458812059\" dateTime=\"2016-03-
24T11:34:19.052+02:00\" type=\"xs:double\">2.2441146426723844</value>\n"
                + "              <value unixTime=\"1458812059\" dateTime=\"2016-03-
24T11:34:19.052+02:00\" type=\"xs:double\">2.2441146426723844</value>\n"
                + "              <value unixTime=\"1458812059\" dateTime=\"2016-03-
24T11:34:19.052+02:00\" type=\"xs:double\">2.2441146426723844</value>\n"
                + "              <value unixTime=\"1458812069\" dateTime=\"2016-03-
24T11:34:29.060+02:00\" type=\"xs:double\">2.2819006321774853</value>\n"
                + "              <value unixTime=\"1458812069\" dateTime=\"2016-03-
24T11:34:29.060+02:00\" type=\"xs:double\">2.2819006321774853</value>\n"
                + "              <value unixTime=\"1458812069\" dateTime=\"2016-03-
24T11:34:29.060+02:00\" type=\"xs:double\">2.2819006321774853</value>\n"
                + "            </InfoItem>\n"
                + "          </Object>\n"
                + "        </Objects>\n"
                + "      </omi:msg>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 10");
```

```
        System.out.println("Actualresponse:" + actualresponse10);


        MyXMLTestCase t10 = new MyXMLTestCase("test");
        t10.setXml(expectedresponse10, actualresponse10);
        try {
            result10 = t10.testForEquality();
            System.out.println("Result:" + result10);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result11 = false;
        String request11 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\">\n"
                + "  <omi:read msgformat=\"odf\" end=\"2016-03-23T16:05:00.000Z\" begin=\"2016-03-
22T22:00:00.000Z\" callback=\"http://localhost:8080/\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>SmartHouse</id>\n"
                + "          <InfoItem name=\"FrontDoor\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </omi:read>\n"
                + "</omi:omiEnvelope>";


        String actualresponse11 = (SendCallbackdata(request11, "http://localhost:8080"));
```

```
        String expectedresponse11 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return description=\"OmiParser: Invalid XML, schema failure: cvc-com-
plex-type.4: Attribute 'ttl' must appear on element 'omi:omiEnvelope'.\" returnCode=\"400\">\n"
                + "      </omi:return>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 11");
        System.out.println("Actualresponse:" + actualresponse11);


        MyXMLTestCase t11 = new MyXMLTestCase("test");
        t11.setXml(expectedresponse11, actualresponse11);
        try {
            result11 = t11.testForEquality();
            System.out.println("Result:" + result11);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result12 = false;
        String request12 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"10\">\n"
                + "  <write xmlns=\"omi.xsd\" msgformat=\"odf\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>K1</id>\n"
                + "          <InfoItem name=\"K1 Building\"/>\n"
```

```
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </write>\n"
                + "</omi:omiEnvelope>";

        String actualresponse12 = (SendCallbackdata(request12, "http://localhost:8080"));
        String expectedresponse12 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return returnCode=\"200\">\n"
                + "      </omi:return>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 12");
        System.out.println("Actualresponse:" + actualresponse12);

        MyXMLTestCase t12 = new MyXMLTestCase("test");
        t12.setXml(expectedresponse12, actualresponse12);
        try {
            result12 = t12.testForEquality();
            System.out.println("Result:" + result12);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
        System.out.println("");
        Boolean result13 = false;
        String request13 = "<?xml version=\"1.0\"?>\n"
                + "<omi:omiEnvelope xmlns:xs=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:omi=\"omi.xsd\" version=\"1.0\" ttl=\"0\">\n"
```

```
                + "  <write xmlns=\"omi.xsd\" msgformat=\"odf\" callback=\"http://lo-
calhost:8080/\">\n"
                + "    <omi:msg>\n"
                + "      <Objects xmlns=\"odf.xsd\">\n"
                + "        <Object>\n"
                + "          <id>K1</id>\n"
                + "          <InfoItem name=\"K1 Building\">\n"
                + "            <MetaData/>\n"
                + "          </InfoItem>\n"
                + "        </Object>\n"
                + "      </Objects>\n"
                + "    </omi:msg>\n"
                + "  </write>\n"
                + "</omi:omiEnvelope>";

        String actualresponse13 = (SendCallbackdata(request13, "http://localhost:8080"));
        String expectedresponse13 = "<omi:omiEnvelope ttl=\"1.0\" version=\"1.0\" xmlns=\"odf.xsd\"
xmlns:omi=\"omi.xsd\" xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n"
                + "  <omi:response>\n"
                + "    <omi:result>\n"
                + "      <omi:return description=\"OK, callback job started\" returnCode=\"200\">\n"
                + "      </omi:return>\n"
                + "    </omi:result>\n"
                + "  </omi:response>\n"
                + "</omi:omiEnvelope>";
        System.out.println("CASE 13");
        System.out.println("Actualresponse:" + actualresponse13);

        MyXMLTestCase t13 = new MyXMLTestCase("test");
        t13.setXml(expectedresponse13, actualresponse13);
        try {
            result13 = t13.testForEquality();
```

```
            System.out.println("Result:" + result13);
        } catch (Exception ex) {
            System.out.println(ex.getCause());
        }
```

```
        Object[] column = {"TEST CASES", "RESULTS"};
        Object[][] data = {{"Case 1", result1}, {"Case 2", result2}, {"Case 3", result3}, {"Case 4", result4}, {"Case 5", result5}, {"Case 6", result6}, {"Case 7", result7}, {"Case 8", result8}, {"Case 9", result9}, {"Case 10", result10}, {"Case 11", result11}, {"Case 12", result12}, {"Case 13", result13}};

        JTable toDoTable = new JTable(data, column) {
            @Override
            public Component prepareRenderer(TableCellRenderer renderer, int rowIndex,
                    int columnIndex) {
                if (columnIndex == 1) {
                    setFont(new Font("Arial", Font.ITALIC, 12));
                } else {
                    setFont(new Font("Arial", Font.BOLD, 12));
                }

                return super.prepareRenderer(renderer, rowIndex, columnIndex);
            }
        };
        JScrollPane jpane = new JScrollPane(toDoTable);
        JPanel panel = new JPanel();
        JFrame frame = new JFrame();
        frame.setSize(new Dimension(300, 400));
        frame.setTitle("OMI Test Cases Results");
        panel.add(jpane);
```

```
        frame.add(new JScrollPane(panel));
        frame.setVisible(true);
        setBackground( Color.gray );
        JPanel topPanel = new JPanel();
                            topPanel.setLayout( new BorderLayout() );
        Component scrollPane = null;


                            topPanel.add( scrollPane, BorderLayout.CENTER );
    }


    private static String SendCallbackdata(String msg, String url1) {
        try {
            URL url;
            String recvbuff = "";
            String recv = "";
            URLConnection urlConnection;
            DataOutputStream outStream;
            url = new URL(url1);
            urlConnection = url.openConnection();
            ((HttpURLConnection) urlConnection).setRequestMethod("POST");
            urlConnection.setDoInput(true);
            urlConnection.setDoOutput(true);
            urlConnection.setUseCaches(false);
            urlConnection.setRequestProperty("Content-Type", "application/xml");
            outStream = new DataOutputStream(urlConnection.getOutputStream());
            outStream.writeBytes(msg);
            outStream.flush();
            outStream.close();
            if (((HttpURLConnection) urlConnection).getResponseCode() >= 400) {
                BufferedReader buffread = new BufferedReader(new InputStreamReader(((HttpURLConnec-
tion) urlConnection).getErrorStream()));
                while ((recv = buffread.readLine()) != null) {
```

```
                    recvbuff += recv;
                }
            } else {
                BufferedReader buffread = new BufferedReader(new InputStreamReader(urlConnection.get-
InputStream()));
                while ((recv = buffread.readLine()) != null) {
                    recvbuff += recv;
                }
            }
            return recvbuff;
        } catch (MalformedURLException ex) {
            System.out.println(ex.fillInStackTrace());
        } catch (UnsupportedEncodingException ex) {
            System.out.println(ex.fillInStackTrace());
        } catch (IOException ex) {
            System.out.println(ex.fillInStackTrace());
        }
        return null;

    }

    private static void setBackground(Color gray) {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated
methods, choose Tools | Templates.
    }

    private static Object getContentPane() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated
methods, choose Tools | Templates.
    }

}
```

Appendix 2: MyXMLTestCase.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package javaapplication15;


import java.io.IOException;
import jdk.internal.org.xml.sax.SAXException;
import org.custommonkey.xmlunit.*;
import org.junit.Test;


/**
 * Example XMLUnit XMLTestCase code Demonstrates use of:<br />
 * <ul>
 * <li>XMLTestCase: assertXMLEqual(), assertXMLNotEqual(), assertXpathExists(),
 * assertXpathNotExists(), assertXpathEvaluatesTo(), assertXpathsEqual(),
 * assertXpathsNotEqual(), assertNodeTestPasses()</li>
 * <li>Diff: similar(), identical()</li>
 * <li>DetailedDiff: getAllDifferences()</li>
 * <li>DifferenceListener: use with Diff class,
 * IgnoreTextAndAttributeValuesDifferenceListener implementation</li>
 * <li>ElementQualifier: use with Diff class, ElementNameAndTextQualifier
 * implementation</li>
 * <li>Transform: constructors, getResultDocument(), use with Diff class</li>
 * <li>Validator: constructor, isValid()</li>
 * <li>TolerantSaxDocumentBuilder and HTMLDocumentBuilder usage</li>
 * <li>NodeTest: CountingNodeTester and custom implementations</li>
 * <li>XMLUnit static methods: buildDocument(), buildControlDocument(),
 * buildTestDocument(), setIgnoreWhitespace()</li>
 * </ul>
```

```
 * <br />Examples and more at
 * <a href="http://xmlunit.sourceforge.net"/>xmlunit.sourceforge.net</a>
 */
public class MyXMLTestCase extends XMLTestCase {

    private String xml1;
    private String xml2;

    public MyXMLTestCase(String name) {
        super(name);
    }

    public void setXml(String a, String b) {
        xml1 = a;
        xml2 = b;
    }

    @Test
    public boolean testForEquality() throws Exception {
        XMLUnit.setIgnoreWhitespace(true);
        XMLUnit.setIgnoreAttributeOrder(true);
        try{
           // say you are comparing FileReader objects that
           // refer to XML documents
         DifferenceListener myDifferenceListener = new IgnoreTextAndAttributeValuesDifferenceLis-
tener();
        Diff myDiff = new Diff(xml1, xml2);
        myDiff.overrideDifferenceListener(myDifferenceListener);
        assertTrue("test XML matches control skeleton XML " + myDiff, myDiff.similar());
         return true;
        } catch (final IOException e) {
           return false;
```

```
        } catch (final AssertionError e) {
          return false;
        }
    }
}
```