Name: Vlaskovski, Radoslav
Matrikelnummer: 358306

Berlin, 27. Juli 2016

Project: Design and Implementation of an OneM2M Testing Framework

Implementation Documentation

Module Next Generation Networks & Future Internet Technologies Projects

Chair "Next Generation Networks (NGN)"
Supervisor : Ronald Steinke
Technische Universität Berlin

Sommersemester 2016

1. Introduction

This document aims to describe how the OneM2M Testing Framework is structured and provide insight for developers how to go about further developing the software. In this document whenever referring to Resource, Primitive, CRUD operation, Request and Response it will be in the sense of the OneM2M standard. The software is divided in three main parts :
1) Helper classes – containing the front-end functionality and some helper classes for the main test classes
2) Mappings – the mapping classes represent the primtives, resources and status codes as described in the OneM2M standard.
3) Test classes – representing the different test implementations

In the next parts each class collection is reviewed in more detail.

2. Helper Modules

The helper classes are in the etc and sender modules.

2.1. Senders

The sender classes implemented are two – a sender class for HTTP and a sender class for MQTT.
The sender class for mqtt has no interface implementation and therefore it has never been used.
The sender class for http is fully functional and can send all CRUD Requests described in the OneM2M standard.

2.2. Logger

Instead of using a library for logging tests there is an implemented Logger class that logs every test session. The logging class is created at the start of each testing session (when the software is launched). When te software is closed the testing session ends and the logging class saves every test in a file with a unique ID for each testing session. The Logfile are stored in the **Logs** folder. For each test it is displayed a simple description, the time of when the test was called, the time it took to test, the result. At the end of the test session the tester can see how much tests were successful and the total amount of executed tests.

2.3. ConsoleReader

The ConsoleReader as the name suggests is used to read tests from the command line. For the description of the commands refer to the User Documentation.

2.4. ParameterChecking

Before each test is executed, each parameter. If all paramaters are acceptable then the test is executed, otherwise an error message is displayed. The error messages are not stored in the log file.

2.5. Config

The config file is read at the start of each session and the ConfigReader has three implemented options that a tester can adjust if he needs to. The options are:

1) console_print – 1 if the tester wants to print the test results to the console, otherwise 0

2) log_to_file – 1 if the tester wants to save the test results to a log file (Recommended), otherwise 0
3) gui – 1 if the tester wants to use the gui, 0 if the tester wants to use the console

2.6. TestFileReader

The test file reader reads a testfile and executes the tests. Test files should be stored in main/testfiles. On how to create a proper test file refer to the User Documentation.

3. Mappings

3.1. Primitive

The Primitive describes the parameters of an OneM2M primitive. Every instance of the class will have three lists – Mandatory, Optional, Not Provided.

3.1.1. Basic Primitives

In the BasicPrimitives module there are five instances of the Primitive class defined – request create, request retrieve, request update, request delete, response. They are defined as described in the OneM2M standard and are meant to be used through out the software mainly for syntax tests.
Changing the parameters of a basic primitive is straight-forward.

3.1.2. PrimitiveHTTP

The PrimitiveHTTP class implements the mapping of pure a pure OneM2M Primitive to an OneM2M HTTP primitive. An instance of the PrimitiveHTTP class has a 6 lists – each three discribed above in the Primitive class but for the content and the headers.
If there are any changes to be made to the mapping they need to be added manually.

3.2. Resources

This class provides a list of all resources described in the OneM2M standard and for each Resource a list of its parameters.
If a new resource is to be added it must be provided with the OneM2M short name e.g. “m2m:cb” and the list of the parameters.

3.3. StatusCodes

This class implements mappings for status codes from number to lexical or from http to m2m format e.g. 2001: 'CREATED' and 2001: 201. This class is used when doing Functional or Behaviour tests.
If the status codes change they need to be edited manually in the dictionaries provided in the class.

4. Tests

4.1. Behaviour

The Behaviour tests are implemented in the BehaviourTest module. The main function takes lists of methods for each request, URLs, expected status codes of the responses, the requests each in JSON format and if provided descriptions for each test. Each list must contain the same number of

elements e.g. if there are five URLs there should be five status codes and five requests in JSON.
A test executed in the here described way:

For the n-th test will be executed with the following parameters:
the n-th URL, JSON, status code and method.

4.2. Functional Tests

4.2.1 Functional test that check status code

The first kind of functional tests send one request and test the status code of the response. The four types are the same as the CRUD operations. The Create and Update tests need a request in either JSON or XML to be executed while read and delete need only an URL and the from the tester expected status code.

4.2.2. Functional test that check a given parameter

This kind of functional tests work the same way as the above mentioned functional tests but instead of checking the status code of the response they check a parameter from the content of the response. The Parameter is not checked before executing the test. If the parameter provided does not exist in the content of the response the test is evaluated as failed.

4.3. Syntax Tests

4.3.1 Syntax Tests for HTTP Responses

Syntax tests for HTTP Responses are structured in three main parts.

1) Check Mandatory Parameters

Each test first checks if all mandatory parameters for a Response Primitive are provided. Checking the Response is in compliance with the OneM2M standard for HTTP Responses and uses the above mentioned HTTP Primitive class. The Mandatory Parameters are checked from both the headers and the content.

2) Check for Not Provided Parameters

Testing for the not provided parameters is implemented the same way as checking the mandatory parameters.

3) Checking All Parameters

In this part all parameters are tested if they are in compliance with the OneM2M Standard or in other words if every parameter is part of the OneM2M Standard. If there is provided content then it is tested according to the resource it represents. For this purpose the above mentioned Resources Module is used.

The test is evaluated as one, this means that if only one mandatory parameter is missing or one parameter is not in the OneM2M standard the test is evaluated as failed.

4.3.2. Syntax tests for normal OneM2M Primitives

Syntax tests for pure OneM2M Primitives are structured in the same way, but instead of checking headers and content the tests check only one json.