Name: Vlaskovski, Radoslav
Matrikelnummer: 358306

Berlin, 27. Juli 2016

Project Design and Implementation of an OneM2M Testing Framework

User 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 documentation is focused on explaining how the OneM2M Testing Framework can be used. It will not provide any deeper technical insight on how the software is structured. In this document whenever referring to Resource, Primitive, CRUD operation, Request and Response it will be in the sense of the OneM2M standard. For the most of the documentation we will be focusing on using the Testing Framework by sending HTTP requests with JSON content. At the end there will be a brief explanation of an underdeveloped version for sending XML content and using the MQTT Protocol.

2. Setup

2.1. Download

The program can be downloaded by cloning from the repository :
PUT REPO HERE

2. Running it

Before running the program the tester needs to run the following command :

python(python3) setup.py install

After that the tester can go to the main folder and execute the main.py as follows:

python(python3) main.py

The setup should install all needed third party packages.

3. Interface

For using the software the user has two options:
1) Use the GUI
2) Use the Console

Both testing options save each testing session separately in a log file.

3.1. Using the GUI

The testing framework provides a simple GUI. Every result is printed on a message box below the test and also on the log file for the current session. How to use the GUI for a specific test types will be explained in the part for that test type. When entering a json in the text box it should be in JSON format and not a file.

3.2. Using the Console

When running a test on the console the result will be printed after the command. At the end the tester will be provided with information regarding how much tests were done and how much were successful. Very important for the console interface is that whenever a JSON needs to be provided it should be created in JSON format and stored in the **jsons_for_testing** folder.

3.3. Test file

When using the console the tester also has the option of running a whole test file. Test files should be stored in the **testfiles** folder, located in the main folder and should be executed in the following

way :

test file <name of the file>

After running a test file all tests will be executed one after the other, starting with the first line.

For creating a test file there are rules that need to be followed in order for it to execute as desired:

1) All tests must be written in the described console form.

2) Each test should be on a separate line

3) the JSON content should be provided from a file and the file should be placed in the **jsons_for_testing** folder

4. Tests

In this part will provide an overview of the types of tests implemented and how they should be executed.

4.1. Test Types

In this part I will briefly explain the implemented test types and after that each one will be further addressed with examples.

1) Syntax Tests

Syntax tests check if a given Primitive (either Request or Response) is compliant with the OneM2M standard.

2) Functional Tests

Functional tests are meant to check either the status code or a special parameter of a Response. The Response is acquired after sending an OneM2M compliant request.

3) Behaviour Tests

Behaviour tests use functional tests that check only the status codes of a series of responses.

4) Resource Specific Tests

Resource Specific tests check if a given implementation reacts according the OneM2M standard for specific resource requests.

4.2. Syntax Tests

4.2.1 Description

Syntax tests test the compliance of a given Primitive with the OneM2M standard. The discussed

syntax tests are implemented to test Primitive that are compliant to the OneM2M HTTP Mapping only. Later Syntax tests for pure OneM2M format will be also discussed so for now when referring to Syntax Tests we will address only HTTP Primitives.

When testing a response the tester must provide everything needed for the execution of the operation. Each test is standalone and no “clean up” is implemented. The testing is structured in three steps.
1) Check that all mandatory parameters are given
2) Check that all not provided parameters are not given
3) Check all parameters that they in compliance with the OneM2M standard.

Although this is one test every parameter is tested and the test log shows the test for every parameter in the response.

4.2.2. Execution

GUI execution requires for the tester to provide all needed information to get the Response before testing it.

Console execution requires a special format
CREATE Response: syntax response create <url> <json>
RETRIEVE Response: syntax response retrieve <url>
UPDATE Response: syntax response update <url> <json>
DELETE Response: syntax response delete <url>

4.3. Functional Tests

4.3.1. Description

Functional tests are based on CRUD operations. The tester needs to provide what type of request he will be sending to the server, the URL of the server and if the request needs any additional content. As when testing the syntax of a response the operation is standalone and no “clean up” is implemented.

4.3.2. Execution

Executing a functional test can be done either through the GUI or from the console.

4.3.3. Execution of Functional Tests that check the status code

GUI execution is straight-forward: enter the URL, the type of the request, what status code you expect and if needed the JSON content and the result of the test will be displayed below.

Console execution needs a specific form :

CREATE : create <url> <expected status code> <json>
RETRIEVE : retrieve <url> <expected status code>
UPDATE: update <url> <expected status code> <json>
DELETE : delete <url> <expected status code> <json>

4.3.4. Execution of Functional Tests that check a given parameter

These tests also run a CRUD operation but instead of checking the status code of the response they check a given paramter/s.
GUI execution is the same as above.

Console execution needs a specific form :
For testing one parameter

parameter retrieve <url> <paramter to be checked>=<expected value>

When testing multiple parameters the parameters and the expected values should be stored in a file and that file placed in the folder **functional_parameters_checking** folder. Creating such a file the tester needs to follow a simple rule-set:

1) The file needs to be in the following format : <parameter 1>=<expected value 1>
<parameter 2>=<expected value 2> …

2) Parameter-value pairs can be written on different lines or multiple pairs can be on the same line

3) Each pair must be on one line or in other words if <parameter 1> is in line 1 <expected value 1> needs to be on the same line

After that running for checking those parameters the tester needs to run:

parameter file <url> <file name>

4.4. Behaviour Tests

4.4.1. Description

Behaviour tests as the name suggest aim to test how the given implementation reacts to a number of CRUD operations executed one after the other.

4.4.2. Execution

There are two types of behaviour tests implemented: custom and built-in. Their execution varies so both types will be explained separately.

4.4.3. Execution of Built-in Behaviour Tests

This is a collection of tests that were considered to be more likely to be used often. Each test is written with the intention for all CRUD operations to be used on the same Resource e.g. create a Resource and afterwards delete it. The list is here described with each CRUD operation that will be executed starting from left to right.

1) CREATE DELETE
    expected status code of the response: 2001 2000
    console execution : behaviour cd <url> <json>

2) CREATE RETRIEVE DELETE
    expected status code of the response: 2001 2000 2000

console execution : behaviour crd <url_create> <json> <url retrieve | delete>

3) CREATE UPDATE RETRIEVE DELETE
expected status code of the response: 2001 2000 2000 2000
console execution :
behaviour curd <url_create> <json create> <url update | retrieve | delete> <json update>

4) CREATE RETRIEVE UPDATE RETRIEVE DELETE
expected status code of the response: 2001 2000 2000 2000 2000
console execution :
behaviour crurd <url_create> <json create> <url update | retrieve | delete> <json update>

5) CREATE DELETE UPDATE
expected status code of the response: 2001 2000 4004
console execution :
behaviour cdu <url_create> <json create> <url update | delete> <json update>

6) CREATE DELETE RETRIEVE
expected status code of the response: 2001 2000 4004
console execution : behaviour cdr <url_create> <json> <url retrieve | delete>

7) CREATE DELETE DELETE
expected status code of the response: 2001 2000 4004
console execution : behaviour cdd <url_create> <json> <url delete>

8) CREATE CREATE
expected status code of the response: 2001 4105
console execution : behaviour cc <url> <json>

In the GUI the execution is similar to the execution of the Functional Tests. For each test from the list above there are fields that should be filled out and after that executed.

4.4.4. Execution of Custom Behaviour Tests

When executing a Custom Behaviour Test it is left to the programmer to decide how many Operations he wants to be executed, the type of each operation and the status code he expects from the response to each request.

GUI executions:

The tester can pick how many operations he intends to execute and after clicking apply that same number of blocks for each operation will be displayed.

Console execution:

Executing a custom behaviour test from the console requires a file. The file should be stored in the folder **behaviour_custom_tests** and should follow these rules:

1) Each operation part of the behaviour test should be in the form :
<method> <url> <expected status code> [json for update and create]

2) Only one operation per line
3) Each JSON should be a JSON file stored in the **jsons_for_testing** folder

After creating the file the execution is :
behaviour custom <test file>

5. Resource Specific Tests

There is a small number of resource specific tests implemented, but they have no interface support. When creating such tests my recommendation is to use a test file and label the test file with the name of the response.

6. MQTT and XML support

Executing tests that send XML requests is also provided but as such have not been executed it may not function properly. An option for sending MQTT requests is implemented but it is not included in the interface and therefore not tested and not functional.

7. Syntax Tests for the pure OneM2M Standard

The testing framework provides also the function to test the syntax of Primitives compliant to the OneM2M standard and not subject to any mapping. The tester needs to provide the Primitive in JSON format and the type of the Primitive. The Primitive types are summed up in : Request create, request update, request retrieve, request delete and response. The output of this kind of test is the same as a normal syntax test.

Console execution :

syntax json <method> <json>

Where the JSON should be a file stored in the **jsons_for_testing** folder.

8. Table of Console Commands

Test type	Console command	Additional Info
FUNCTIONAL THAT CHECK STATUS CODE		
CREATE	create <url> <expected status code> <json>	
RETRIEVE	retrieve <url> <expected status code>	
UPDATE	update <url> <expected status code> <json>	
DELETE	delete <url> <expected status code> <json>	
FUNCTIONAL THAT CHECK PARAMETER/S		
RETRIEVE and check parameter	parameter console <url> <parameter>=<value>	
RETRIEVE and check parameters	paramter file <url> <file name>	Test file stored in : functional_parameters_ checking
BEHAVIOUR		
Custom	behaviour custom <test file>	Test file stored in : behaviour_custom_tests
CREATE DELETE	behaviour cd <url> <json>	Expected: 2001 2000
CREATE RETRIEVE DELETE	behaviour crd <url_create> <json> <url retrieve | delete>	Expected: 2001 2000 2000
CREATE UPDATE RETRIEVE DELETE	behaviour curd <url_create> <json create> <url update | retrieve | delete> <json update>	Expected: 2001 2000 2000 2000
CREATE RETRIEVE UPDATE RETRIEVE DELETE	behaviour crurd <url_create> <json create> <url update | retrieve | delete> <json update>	Expected: 2001 2000 2000 2000 2000
CREATE DELETE DELETE	behaviour cdd <url_create> <json> <url delete>	Expected: 2001 2000 4004
CREATE DELETE RETRIEVE	behaviour cdr <url_create> <json> <url retrieve | delete>	Expected: 2001 2000 4004
CREATE DELETE UPDATE	behaviour cdu <url_create> <json create> <url update | delete> <json update>	Expected: 2001 2000 4004
CREATE CREATE	behaviour cc <url> <json>	Expected: 2001 2000 4105
SYNTAX		
Test Syntax of Response	syntax response <method><url> [json]	Method is CRUD