ONAP MUSIC Functional Test Cases

Author: Leonardo Bellini, Tom Nelson

Table of Contents

Scope

This document contains the description of Functional Test Cases planned for Onap MUSIC in the scope of Beijing release.
From now on you will find the software product being identified by both “ONAP MUSIC” of by “MUSIC” acronym.

This document is subjected to evolve/change during ONAP Beijing release lifetime according to detailed commitments agreed during product consolidation.

Information reported here try to give one effective response to Functional Testing items listed in ONAP MUSIC Release Planning Checklist Beijing (R2) – see link below

https://wiki.onap.org/to_be_completed

OOF Functional Test Cases

With reference to the following architectural diagram, we are focusing Functional Testing on MUSIC API interface.

All Functional Test Cases described here below will be automatized in the CSIT ONAP integration environment.

Test Case id	Description	Pre-Conditions	Test-Steps	Expected Results
1	**Name: CheckVersion** Interface MUSIC API Perform sanity check verifying Music version through REST API	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to check SW Version Method - GET See Test detail section for URL descrioption	MUSIC should respond with HTTP 200 and body containing MUSIC VERSION
2	**Name: Keyspace creation** Interface MUSIC API Perform sanity check creating a new keyspace	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to create anew keyspace Method - POST	MUSIC should respond with HTTP 200
3	**Name: Table Creation** Interface MUSIC API Perform sanity check creating a new keyspace.table	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to create a new table within previously created keyspace Method - POST	MUSIC should respond with HTTP 200
4	**Name: Insert Row** Interface MUSIC API Perform sanity check creating a new keyspace.table.row	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to insert a new row into previously created table Method - POST	MUSIC should respond with HTTP 200
5	**Name: Read inserted Row** Interface MUSIC API Perform sanity check reading a new keyspace.table.row	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to get a row content from previously inserted row Method - GET	MUSIC should respond with HTTP 200 and with body reporting row content
6	**Name: Update Row** Interface MUSIC API Perform sanity check creating a new keyspace.table.row	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to insert a new row into previously created table Method - PUT	MUSIC should respond with HTTP 200
7	**Name: Read updated Row** Interface MUSIC API Perform sanity check reading a new keyspace.table.row	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to get a row content from previously updated row Method - GET	MUSIC should respond with HTTP 200 and with body

				reporting row content
8	**Name: Delete Row** Interface MUSIC API Perform sanity check deleting a new keyspace.table.row	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to delete row from previously created table Method - DELETE	MUSIC should respond with HTTP 200
9	**Name: Table Drop** Interface MUSIC API Perform sanity check dropping a new keyspace.table	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to drop a table table Method - DELETE	MUSIC should respond with HTTP 200
10	**Name: Keyspace Drop** Interface MUSIC API Perform sanity check dropping a new keyspace	1. MUSIC docker image is up and running	Robot Framework is sending a REST call to MUSIC to drop a keyspace Method - DELETE	MUSIC should respond with HTTP 200

Test Cases Details

Test Case N.1 – Check Version

REST Method: GET
URL:
`http://$(hostname):8080/MUSIC/rest/{interface_version}/version`

where:
`{interface_version}: should be set to a valid MUSIC API interface version`

Test Case N.2 – Keyspace Creation

REST Method: POST
URL:
`http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}`

where:
```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.3 – Table Creation

REST Method: POST
URL:
```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/tables/{table_name}
```

where:
```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.4 – Insert Row

REST Method: POST
URL:
```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/tables/{table_name}/rows/{row_key}
```

where:
```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
{row_key}: it is a string representing the primary key for the new row
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.5 – Read Inserted Row

REST Method: GET
URL:
```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/tables/{table_name}/rows/{row_key}
```

where:

```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
{row_key}: it is a string representing the primary key for the new row
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.6 – Update Row

REST Method: PUT
URL:

```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/ta
bles/{table_name}/rows/{row_key}
```

where:

```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
{row_key}: it is a string representing the primary key for the new row
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.7 – Read Updated Row

REST Method: GET
URL:

```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/ta
bles/{table_name}/rows/{row_key}
```

where:

```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
{row_key}: it is a string representing the primary key for the new row
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.8 – Delete Row

REST Method: DELETE
URL:
```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/tables/{table_name}/rows/{row_key}
```

where:
```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
{row_key}: it is a string representing the primary key for the row to delete
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.9 – Table Drop

REST Method: DELETE
URL:
```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}/tables/{table_name}
```

where:
```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
{table_name}: it is a string representing the cassandra table name
```

REST Body:
To be added

REST RESPONSE:
To be added

Test Case N.10 – Keyspace Drop

REST Method: DELETE
URL:
```
http://$(hostname):8080/MUSIC/rest/{interface_version}/keyspaces/{keyspace_name}
```

where:
```
{interface_version}: should be set to a valid MUSIC API interface version
{keyspace_name}: it is a string representing the cassandra keyspace name
```

REST Body:
To be added

REST RESPONSE:
To be added