|  |
| --- |
| CHANGE REQUEST |
| Meeting ID:\* |  |
| Source:\* |  |
| Date:\* |  |
| Reason for Change/s:\* |  |
| CR against: Release\* |  |
| CR against: WI\* | [x]  Active <WI-108> [ ]  MNT maintenance / < Work Item number(optional)>Is this a mirror CR? Yes [ ]  No [x] mirror CR number: [ ]  STE Small Technical Enhancements / < Work Item number (optional)>Only ONE of the above shall be ticked |
| CR against: TS/TR\* |  |
| Clauses \* |  |
| Type of change: \* | [ ]  Editorial change[x]  Bug Fix or Correction[ ]  Change to existing feature or functionality[ ]  New feature or functionalityOnly ONE of the above shall be ticked |
| Other TS/TR(s) impacted | None |
| Post Freeze checking:\* | This CR contains only essential changes and corrections? YES [x]  NO [ ] This CR may break backwards compatibility with the last approved version of the TS? YES [ ]  NO [ ]  |
| Template Version: January 2019 (do not modify) |

**oneM2M Notice**

The document to which this cover statement is attached is submitted to oneM2M. Participation in, or attendance at, any activity of oneM2M, constitutes acceptance of and agreement to be bound by terms of the Working Procedures and the Partnership Agreement, including the Intellectual Property Rights (IPR) Principles Governing oneM2M Work found in Annex 1 of the Partnership Agreement.

GUIDELINES for Change Requests:

Provide an informative introduction containing the problem(s) being solved, and a summary list of proposals.

Each CR should contain changes related to only one particular issue/problem.

In case of a correction, and the change apply to previous releases, a separate “mirror CR” should be posted at the same time of this CR

Mirror CR: applies only when the text, including clause numbering are exactly the same.

Companion CR: applies when the change means the same but the baselines differ in some way (e.g. clause number).

Follow the principle of completeness, where all changes related to the issue or problem within a deliverable are simultaneously proposed to be made E.g. A change impacting 5 tables should not only include a proposal to change only 3 tables. Includes any changes to references, definitions, and acronyms in the same deliverable.

Follow the drafting rules.

All pictures must be editable.

Check spelling and grammar to the extent practicable.

Use Change bars for modifications.

The change should include the current and surrounding clauses to clearly show where a change is located and to provide technical context of the proposed change. Additions of complete clauses need not show surrounding clauses as long as the proposed clause number clearly shows where the new clause is proposed to be located.

Multiple changes in a single CR shall be clearly separated by horizontal lines with embedded text such as, start of change 1, end of change 1, start of new clause, end of new clause.

When subsequent changes are made to content of a CR, then the accepted version should not show changes over changes. The accepted version of the CR should only show changes relative to the baseline approved text.

## Introduction

Some adaptations from the converted version 2.8.0.

<https://git.onem2m.org/specifications/ts-0019/-/merge_requests/26>

----------------------- Start of change 1 -----------------------

##### 5.4.2.2.2 UtTrigger and UtTriggerAck Primitives

The UtTrigger primitive is initialized by the Test System to send triggering message to the target IUT as depicted in figure 5.4.2.2.2-1. The IUT will send acknowledgement message back to the Test System using UtTriggerAck primitive if trigger message is successfully transported to the IUT. Then IUT starts interaction with Test System through oneM2M request and response primitives.



Figure 5.4.2.2.2-1: Trigger message flow

Table 5.4.2.2.2-1 defines UtTrigger and UtTriggerAck primitives including oneM2M data types to which are mapped as well as examples to show how to implement UtTrigger and UtTriggerAck primitives.

Table 5.4.2.2.2-1: UtTrigger and UtTriggerAck Primitive

| Ut Control Primitive | Mapping to oneM2M data types | Description | Reference | Triggering Message | HTTP message |
| --- | --- | --- | --- | --- | --- |
| *UtTrigger* *Primitive* | *requestPrimitive* | ONLY essential parameters included for certain test caseSee note 1 | oneM2MTS-0004 [2] | EXAMPLE 1 :If the test objective is to test *“* *Test* *System* *triggers* *IUT* *to* *execute* *a* *test* *case* *for* *creation* *of* *<* *AE* *>* *with* *labels* *attribute* *under* *a* *CSEBase* *resource* “, then the triggering message would be serialized as following. | EXAMPLE 1 :If the test objective is to test *“* *Test* *System* *triggers* *IUT* *to* *execute* *a* *test* *case* *for* *creation* *of* *<* *AE* *>* *with* *labels* *attribute* *under* *a* *CSEBase* *resource* “, then the triggering message would be serialized as following. |
| *UtTrigger* *Primitive* | *requestPrimitive* | ONLY essential parameters included for certain test caseSee note 1 | oneM2MTS-0004 [2] | Request{“m2m:rqp” :{ “op”: 1, //indicate CREATE operation “ty”: 2, //indicate AE resource type “to”: {TEST\_SYSTEM\_ADDRESS}, “pc”: { “m2m:ae”: { “lbl”:“UNINITIALIZED” //indicate that attribute labels needs to be included }, } “rvi”: “2a”}} | Reques tPOST /{SUT\_UT\_APPLICATION\_URL} HTTP/1.1Host : {SUT\_IP\_ADDRESS:PORT}Content-Length : {PAYLOAD\_LENGTH}Content-Type : application/ json{“m2m:rqp” :{ “op”: 1, //indicate CREATE operation “ty”: 2, //indicate AE resource type “to”: {TEST\_SYSTEM\_ADDRESS}, “pc”: { “m2m:ae”: { “lbl”:“UNINITIALIZED” //indicate that attribute labels needs to be included } }, “rvi”: “2a”}} |
| *UtTrigger* *Primitive* | *requestPrimitive* | ONLY essential parameters included for certain test caseSee note 1 | oneM2MTS-0004 [2] | EXAMPLE 2 : If the test objective is to test “*Test* *System* *triggers* *IUT* *to* *execute* *a* *test* *case* *for* *delete* *of* *a* *<* *AE* *>* *resource.*”, then the triggering message would be serialized as following. | EXAMPLE 2 : If the test objective is to test “*Test* *System* *triggers* *IUT* *to* *execute* *a* *test* *case* *for* *delete* *of* *a* *<* *AE* *>* *resource.*”, then the triggering message would be serialized as following. |
| *UtTrigger* *Primitive* | *requestPrimitive* | ONLY essential parameters included for certain test caseSee note 1 | oneM2MTS-0004 [2] | Request{“m2m:rqp” :{ “op”: 4, //indicate DELETE operation “to”: {TARGET\_AE\_RESOURCE\_ADDRESS}, //indicate Target AE resource address “rvi”: “2a”}} | RequestPOST /{SUT\_UT\_APPLICATION\_URL} HTTP/1.1Host : {SUT\_IP\_ADDRESS:PORT}Content-Length : {PAYLOAD\_LENGTH}Content-Type : application/ json{“m2m:rqp” :{ “op”: 4, //indicate DELETE operation “to”: {TARGET\_AE\_RESOURCE\_ADDRESS}, //indicate Target AE resource address “rvi”: “2a”}} |
| *UtTrigger* *Ack* *Primitive* | *responsePrimitive* | ONLY responseStatusCode attribute includedSee note 2 | oneM2M TS-0004 [2] | Response{ “m2m:rsp”: { “rsc”: 2000 }}For any triggering response, it only contains a response status code, and the response status code for the triggering operation can only be set to either 2000 (OK) or 4000 (BAD\_REQUEST) according to the rules for triggering operations. | ResponseHTTP/1.1 200 OKX-M2M-RSC: 2000 |
| NOTE 1: Additional rules defined in table 5.4.2.2.2-3 are also applied.NOTE 2: Attribute response status code is defined at table 5.4.2.2.2-3. | NOTE 1: Additional rules defined in table 5.4.2.2.2-3 are also applied.NOTE 2: Attribute response status code is defined at table 5.4.2.2.2-3. | NOTE 1: Additional rules defined in table 5.4.2.2.2-3 are also applied.NOTE 2: Attribute response status code is defined at table 5.4.2.2.2-3. | NOTE 1: Additional rules defined in table 5.4.2.2.2-3 are also applied.NOTE 2: Attribute response status code is defined at table 5.4.2.2.2-3. | NOTE 1: Additional rules defined in table 5.4.2.2.2-3 are also applied.NOTE 2: Attribute response status code is defined at table 5.4.2.2.2-3. | NOTE 1: Additional rules defined in table 5.4.2.2.2-3 are also applied.NOTE 2: Attribute response status code is defined at table 5.4.2.2.2-3. |

The rules for defining UtTrigger and UtTriggerAck primitives are:

1. UtTrigger primitive is represented in requestPrimitive serialized in JSON format.

1. UtTrigger primitive shall be interpreted as follows:

- Any attribute/parameter containing a value shall be present and equal in the triggered request primitive.

- Any attribute/parameter containing “UNINITIALIZED” value shall be present in the triggered request primitive.

- Any other attribute/parameter shall comply with oneM2M TS-0004 [2].

1. Parameters within UtTrigger are listed as following:

- operation: (mandatory) operation type that IUT is triggered to perform.

- resourceType: (optional) resource type of a target resource against which IUT is triggered to perform certain operation

- to: (mandatory) target resource against which IUT is triggered to perform certain operation.

- primitiveContent: (optional) represents the resource attributes that shall be included in the requestPrimitive.

| |

Table 5.4.2.2.2-3: Definition of ResponseStatusCode for UtTriggerAck primitive

| Response Status Code Description | Response Status Code Value | Interpretation |
| --- | --- | --- |
| OK | 2000 | The SUT receives successfully the triggering message from Test System |
| BAD\_REQUEST | 4000 | The SUT does not interpret correctly the UtTrigger primitive |
| NOTE: Only above two response status codes are allowed to use in UtTriggerAck primitive. | NOTE: Only above two response status codes are allowed to use in UtTriggerAck primitive. | NOTE: Only above two response status codes are allowed to use in UtTriggerAck primitive. |

----------------------- End of change 1 -----------------------

----------------------- Start of change 2 -----------------------

## 5.6 Test strategy

This clause introduces the test strategy being used for the TTCN-3 test cases. The chosen strategy permits to have a clear structure of the code that facilitates an easy navigation throw the different test steps.

The use of the TTCN-3 MTC and PTC(s) is as depicted in figure 5.6-1.



Figure 5.6-1: Use of TTCN-3 components

At the start of the test case execution, the MTC is created. Then, the MTC executes the following steps:

* Step 1) initialization of the master PTC.
* Step 2) initialization of some parameters if required for the permutation test cases.
* Step 3) running of the appropriate function on the master PTC. The function run on the master PTC implements a given Test Purpose. Such function follows a code structure as indicated here below:
	+ Local Variables, declaration of local variables.
	+ Test Control, checking IUT capability parameters required for the proper execution of the test.
	+ Test Component Configuration, that initializes the given test component and other test components acting as slave PTC(s) as required by a given configuration.
	+ Test adapter configuration, that configures the test adapter throw the acPort if required.
	+ Preamble, that implements the necessary test steps as described in the Initial conditions of a Test Purpose. It may also implement additional test steps which are required for the correct execution of the test.
	+ Test body, that implements the test steps as described in the Expected behaviour of a Test Purpose.
	+ Postamble, that implements the necessary test steps to bring the IUT back to the initial state.
	+ Tear down, that finalizes properly the TTCN-3 ports used by the different test components depending on the configuration.

While master PTC follows the test structure described above, slave PTC(s) run only certain procedures, usually one by one, as mandated by the master PTC.

A procedure usually implements a oneM2M request-response exchange between a given PTC and the IUT, although it can implement any other specific action (sending or reception of a message, several request-response exchanges, etc.).

* Step 4) checking of some parameters if required for the permutation test cases.

This test strategy may slightly vary for certain cases where specific requirements need to be fulfilled.

----------------------- End of change 2 -----------------------

----------------------- Start of change 3 -----------------------

## 5.1 Abstract protocol tester

An abstract protocol tester (APT) is a process that provides behaviours for testing an IUT by emulating a peer IUT at the same layer, and enabling to address a single test objective.

APTs used by the oneM2M test suite are described in figure 5.1-1. The test system will simulate valid and invalid protocol behaviour, and will analyse the reaction of the IUT.



Figure 5.1-1: Abstract protocol testers - oneM2M

As figure 5.1-1 illustrates, the corresponding ATS needs to use lower layers to establish a proper connection to the system under test (SUT) over a physical link (Lower layers link). Four different lower layers have been specified corresponding to the binding protocols considered in oneM2M: HTTP, CoAP, WebSocket and MQTT.

----------------------- End of change 3 -----------------------

----------------------- Start of change 4 -----------------------

##### 5.4.2.2.1 Introduction

The upper tester triggering message is used to transport control commands between Test System and the Upper Tester Application. The control command will contain essential parameters that are required for certain test case.

The upper tester triggering message type maps to particular message formats for exchanging data and those message formats are defined by TTCN-3 primitive as shown at table 5.4.2.2.1-1, *U* *tTrigger* and *U* *tTriggerAck* primitive.

Table 5.4.2.2.1-1: Mapping of TTCN-3 Primitives to oneM2M Service Primitives

| Upper Tester Control Message Type | TTCN-3 Primitives | Direction | Direction |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Trigger | UtTrigger Primitive | TS | UT |
| Trigger Acknowledgement | UtTriggerAck Primitive | UT | TS |

----------------------- End of change 4 -----------------------

----------------------- Start of change 5 -----------------------

# Contents

[Contents](#contents)
[1 Scope]
[2 References]
    [2.1 Normative references]
    [2.2 Informative references]
[3 Definition of terms, symbols and abbreviations]
    [3.1 Terms]
    [3.2 Symbols]
    [3.2 Abbreviations]
[4 Conventions]
[5 Abstract Test Method (ATM)]
    [5.1 Abstract protocol tester]
    [5.2 Test Configuration]
        [5.2.1 AE Test Configuration]
    [5.3 Test architecture]
    [5.4 Ports and ASPs (Abstract Services Primitives)]
        [5.4.0 Introduction]
        [5.4.1 mcaPort, mcaPortIn, mccPort, mccPortIn]
        [5.4.2 utPort]
            [5.4.2.0 Introduction]
            [5.4.2.1 Usage for Automated AE Testing]
            [5.4.2.2 Upper Tester Control Primitives]
                [5.4.2.2.1 Introduction]
                [5.4.2.2.2 UtTrigger and UtTriggerAck Primitives]
                [5.4.2.2.3 Control Communication Protocol]
                [5.4.2.2.4 Control Message Serialization]
        [5.4.3 acPort]
        [5.4.4 infoPort]
    [5.5 Test components]
        [5.5.1 Tester]
        [5.5.2 AeSimu]
        [5.5.3 CseSimu]
    [5.6 Test strategy]
[6 Untestable Test Purposes]
[7 ATS Conventions]
    [7.0 Introduction]
    [7.1 Testing conventions]
        [7.1.1 Testing states]
            [7.1.1.1 Initial state]
            [7.1.1.2 Final state]
    [7.2 Naming conventions]
        [7.2.1 General guidelines]
        [7.2.2 oneM2M specific TTCN-3 naming conventions]
        [7.2.3 Usage of Log statements]
        [7.2.4 Test Case (TC) identifier]
    [7.3 IXIT]
[8 TTCN-3 Verifications]
[Annex A (normative): TTCN-3 library modules]
    [A.1 Electronic annex, zip file with TTCN-3 code]
[Annex B (informative): Bibliography]
[History]

----------------------- End of change 5 -----------------------

----------------------- Start of change 6 -----------------------

### 5.4.1 mcaPort, mcaPortIn, mccPort, mccPortIn

These ports are used to send and receive the following message sets:

* Request Primitives messages in accordance with oneM2M TS-0004 [2].
* Response Primitives messages in accordance with oneM2M TS-0004 [2].

Two primitives are currently defined for these ports indicated as table 5.4.1-1:

1. The M2MRequestPrimitive - to send or receive oneM2M messages to/from the IUT. Depending on the IUT to be tested:
	1. If the IUT is an AE, these messages are either received or sent by the tester which is associated with the CSE role through the mcaPortIn or the mcaPort respectively.
	2. If the IUT is a CSE, these messages are either sent or received by the tester when it plays the AE role through the mcaPort or the mcaPortIn respectively, or sent or received by the tester when it plays the CSE role through the mccPort or the mccPortIn respectively.
2. The M2MResponsePrimitive - to send or receive oneM2M messages to/from the IUT. Depending on the IUT to be tested:
	1. If the IUT is an AE, these messages are either sent or received by the tester which is associated with the CSE role through the mcaPortIn or the mcaPort respectively.
	2. If the IUT is a CSE, these messages are either sent or received by the tester when it plays the CSE role through the mccPortIn or the mccPort respectively, sent or received by the tester when it plays the AE role through the mcaPortIn or mcaPort respectively.

Both primitives contain another parameters that permits to dynamically configure the test adaptor for every single sending. These parameters are:

* Host: IP address of the IUT
* XML Namespace
* Protocol binding
* Serialization
* ForceFields: used to force invalid or empty values to certain attributes. This behaviour shall be implemented by the System Adaptor.

Table 5.4.1-1: Mapping of TTCN-3 Primitives to oneM2M Service Primitives

| TTCN-3 Primitive | oneM2M Message | Direction | IUT |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| M2MRequestPrimitive | Request Primitive | <=> | AE |
| M2MRequestPrimitive | Request Primitive | <=> | CSE |
| M2MResponsePrimitive | Response Primitive | <=> | AE |
| M2MResponsePrimitive | Response Primitive | <=> | CSE |

----------------------- End of change 6 -----------------------