|  |
| --- |
|  |

|  |  |
| --- | --- |
| CHANGE REQUEST | |
| Meeting ID:\* | SDS #48 |
| Source:\* | Peter Niblett, IBM |
| Date:\* | 2020-12-14 |
| Reason for Change/s:\* | Clarify expected behaviour if non-confirmable messages are used (R4) |
| CR against: Release\* | Release 4 |
| CR against: WI\* | Active WI-xxxx  MNT maintenance / < Work Item number(optional)>  Is this a mirror CR? Yes  No  mirror CR number: (Note to Rapporteur - use latest agreed revision)  STE Small Technical Enhancements / < Work Item number (optional)>  Only ONE of the above shall be ticked |
| CR against: TS/TR\* | TS-0008 v.4.0.0 |
| Clauses \* | Modified clauses: 3, 5.1, 6.2.1, 6.3.0, 6.3.1, 6.3.2, 6.3.3, 6.3.4 |
| Type of change: \* | Editorial change  Bug Fix or Correction  Change to existing feature or functionality  New feature or functionality  Only ONE of the above shall be ticked |
| Impacted other TS/TR(s) |  |
| Post Freeze checking:\* | This CR contains only essential changes and corrections? YES  NO  This CR may break backwards compatibility with the last approved version of the TS? YES  NO |
| Template Version: January 2017 (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

TS-0008 clauses 6.3.1, 6.3.2, 6.3.3 and 6.3.4 describe how the oneM2M request/response patterns map to the CoAP messaging model. All four of the say that requests shall be sent using Confirmable messages and the flows described assume this.

They do not say what happens if an Originator chooses to send a Request using a Non-confirmable message. If the receiver gets this message, should it attempt to respond with an error, accept the request or ignore the request without attempting to send a response at all?

They are also not 100% clear whether the receiver is required to use Confirmable messages when responding. In some cases the text does say this (or the diagrams say CON) but this requirement is missing in some places, for example 6.3.1 (blocking case).

This CR clarifies these points:

Originators should use Confirmable messages when sending requests, but they can use non-confirmable if there’s a good reason for doing this (e.g. they aren’t interested in the reply or whether the requested operation actually happened)

The consequence of this is that a CoAP receiver should accept incoming non-confirmable messages (if it gets them)

In Blocking Mode, if a request is sent as Non-Confirmable then the response is sent as Non-Confirmable

In Non-Blocking Asynch, if a request is sent as Non-Confirmable the acknowledgement of that request is sent as Non-confirmable but the actual response notification is sent as Confirmable

In Non-Blocking Synch the immediate response is sent as Confirmable, but the originator could choose to use a Non-confirmable request to retrieve that actual response (since this is a blocking retrieve).

[Also “Confirmable Method” has been changed to “Confirmable message”]

### R01/02.

Issues raised when reviewing R0.

1. Text in 6.3.0 that says “it shall resend that request until it has been acknowledged”. This implies that this process continues indefinitely whereas section 4.2 of the CoAP RFC describes a number of parameters that control the retransmissions, including one called MAX\_RETRANSMIT  
     
   Clause 6.2.1 of TS-0008 handles this by saying “The Originator and Receiver shall set the 16 bit MessageId in accordance with the CoAP specification [1] and shall retry transmission of all unacknowledged Confirmable messages, as required by that specification.”  
     
   Wording of 6.3.0 paragraph 4 has been adjusted to refer to [1].
2. The current text assumes that every request generates a response. There is no mention of what should happen if ***ResponseType*** is set to noResponse  
     
   According to TS-0004 table 6.3.4.2.6‑1 the noResponse ***ResponseType*** “shall only be used for procedures related to 3GPP Interworking defined in oneM2M TS-0026 [43].” However there isn’t any explicit mention of noResponse in that TS, so it’s not clear what when it can be used. For the purposes of this CR we will ignore noResponse and consider only the Blocking, FlexBlocking and the 2 non-blocking ResultTypes – all of which require there to be a Response (even if ***ResultContent*** is Nothing).   
     
   If there’s a requirement to permit the use of noResponse for Mca or Mcc running over CoAP that should be included in a separate CR.
3. Concerns over the requirements for Confirmable/Non-confirmable on responses generated by the Receiver and inconsistencies between the cases described in 6.3.1, 6.3.2 and 6.3.3. This included a concern that it makes things more complicated for the Receiver if it has to remember whether the request was Confirmable or not when it comes to send the response.

For issue 3, there are several options for the responses from Receiver to Originator, and we might decide to take different approaches depending on the ***ResponseType*** (blocking, non-blocking-sync etc).

1. Require that messages sent from the Receiver to the Originator (both CoAP responses and CoAP requests) are sent as Confirmable
2. Say that messages from Receiver to Originator match the confirmability of the originating request (if that Request is non-confirmable then they are non-confirmable and vice versa)
3. Specify that some of these messages are required to match the request, and that some are required to be Confirmable (this is what I attempted in R00 of this CR)
4. Say that the Receiver is free to choose confirmable or non-confirmable for any message it sends
5. Reject a non-confirmable request by sending a CoAP Reset (RST) response

Notes from the CoAP RFC 7252

* The RFC gives a description of Non-confirmable containing a possible use case: “Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor.”
* The RFC has normative language that says that a response to a Confirmable message must be sent as Confirmable:
  + A recipient MUST either (a) acknowledge a Confirmable message with an Acknowledgement message or (b) reject the message if the recipient lacks context to process the message properly… Rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it.
* The RFC has text that suggests that a response to a Non-confirmable should be Non-confirmable but it does not use normative language for this (it has ‘is’ rather than ‘MUST’):
  + If a request is sent in a Non-confirmable message, then the response is sent using a new Non-confirmable message, although the server may instead send a Confirmable message.

In the light of this, I can see the following cases where the Originator might want to use Non-Confirmable:

1. The Originator doesn’t mind if the request doesn’t get executed and is never going to wait for or examine a response.   
     
   This is the case that is hinted in RFC 7252 where an AE posts a sensor reading every so often. It isn’t using a response to pace these posts, it isn’t (implicitly or explicitly) going to retry any failed posts and it doesn’t have any error handling or logging code that would want to know about failed posts.   
     
   This case would be best handled by the ***ResponseType*** of noResponse, which I suggest we do in a separate CR.
2. The Originator doesn’t mind if the request doesn’t get executed. It would make use of a response, but it doesn’t mind if it doesn’t get one.  
     
   For example an AE that issues a Retrieve request but abandons it and moves on with something else if it doesn’t get a timely response.

For blocking requests, this matches the RFC-suggested behaviour “If a request is sent in a Non-confirmable message, then the response is sent using a new Non-confirmable message”

1. The Originator doesn’t mind if the request doesn’t get executed, but would like to know its result (including failure) if it did.   
     
   This is the case where the Originator wants to log the failure, or has some recovery action that it could take, like doing its own explicit retry.  
     
   For blocking requests, this matches the RFC-alternative behaviour “the server may instead send a Confirmable message”

If we assume that A (which is the most likely case, in my opinion) is going to be handled by a different CR that leaves us with deciding whether (and how) to support B and C. It’s also difficult to determine whether the Originator wants the B or C behaviour, although we could assume that if it’s a Retrieve request then it’s likely to be B.

I therefore suggest we handle the different modes as follows:

1. Blocking Mode

If a request is sent as Confirmable then the response, if not sent in the ACK, shall be sent as Confirmable. This is also the intention of the current spec.

If a request is sent as Non-Confirmable then the response should be sent as Non-Confirmable.

Note: This matches the approach recommended in the CoAP RFC, and is in effect asserting behaviour B.

We could add some “local policy” wording” to allow the Receiver to send a Confirmable response, as this would still be allowed by the CoAP RFC. That would mean that the Receiver could support case C, if it has some out-of-band knowledge that that’s what the originator wants. Alternatively anyone wanting case C could use a non-blocking ResponseType. I have not added the local policy wording.

Non-blocking Synch  
  
This mode is a bit like the CoAP “separate response” pattern in that it’s to be used if the request is going to take a while to complete. The Originator sends two (or more) CoAP request messages. The first of these is the oneM2M request itself, and subsequent ones are requests to retrieve the oneM2M response. If the Originator doesn’t receive the oneM2M ACK for the original response it can’t issue these subsequent requests (as the ACK contains the <request> reference)

The “subsequent requests” are non-blocking Retrieves, so they should be treated as in case 1.

The problem is what do to with initial oneM2M ACK. We can assume that any Originator using this mode with a non-Confirmable request is likely to be of type C, so we should make that ACK confirmable (or alternatively we could reject non-Confirmable requests with a CoAP Reset response). I have taken the former approach

Non-blocking Asynch

In this mode the Receiver expects the oneM2M response to be returned as a notification. If it has gone to the bother of asking for that, we could assume that it is likely to be of type C and so we should say that the notifications are sent as Confirmable

In this mode the Receiver is also permitted (but not required) to send a oneM2M ACK containing the <request> reference, as in case 2. For consistency with 2 we should make the ACK confirmable, but since the Originator can’t be sure that the Receiver will send it (even if its original request was confirmable) we could make it non-confirmable. I have gone with consistency with 2.

In summary – Blocking Synch is handled as case B the other two as case C.

### R03.

Improved the text to distinguish between oneM2M responses and CoAP responses.

Changed “shall” to “should” to allow a Blocking Synch response to be sent as Confirmable in the case where the request was sent as Non-confirmable.

Clarified 6.3.2 to say that the asynch non-blocking response is carried in the content of a Notification primitive

Added a change to 5.1 Required Features to include Non-confirmable.

Added a change to 6.2.1 to include Non-confirmable as well as Confirmable.

Added abbreviation in clause 3.

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start of change 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### 6.3.0 Introduction

This clause describes the behaviour of the CoAP layer depending on the ***Response Type*** parameter. Note that the CoAP messaging model defined in [1] applies to all message exchanges.

oneM2M Requests should be sent as CoAP Confirmable (CON) messages, although an Originator can send a request as a Non-confirmable (NON) message if there is a good reason for doing this. An Originator should not use Non-confirmable if it relies on getting a response to its request.

oneM2M Responses should be sent as CoAP CON messages, although there is one case where a NON message should be used. This is indicated in clause 6.3.1.

If the Originator or Receiver sends a CON message it shall retransmit that message if it does not receive a CoAP acknowledgement message, as required by [1]. The recipient (Receiver or Originator) shall take care to de-duplicate CON messages as described in [1].

The recipient of a CoAP message shall process the oneM2M request or response it contains, even if it was sent as Non-confirmable.

### 6.3.1 Blocking case

1. If ***Response Type*** parameter is configured as "blockingRequest" (blocking case), the Originator (CoAP client) shall send the oneM2M request to the Receiver (CoAP server). The oneM2M ***Operation*** parameter shall be mapped to a CoAP Method according to Table 6.2.1-1.
2. After processing the oneM2M request, the Receiver shall send the oneM2M response in a CoAP response with a CoAP response code as given by Table 6.2.4-1. If the request was sent as a CoAP Confirmable message, the Receiver may either piggyback this response to the request on the CoAP ACK message, or send the response as a separate CoAP Confirmable message after it has sent the CoAP ACK. If the oneM2M request was sent in a Non-confirmable message, the oneM2M response shall be returned as a separate CoAP message. This response should be sent as a Non-confirmable CoAP message but it may be sent as Confirmable.
3. The Originator’s CoAP binding may generate a response primitive containing a oneM2M ***Response Status Code*** of "REQUEST\_TIMEOUT" if it considers that it has taken too long for the CoAP response to come back from the Receiver. It shall ignore any response to the original request that it might receive after it has done this.

### 6.3.2 Non-Blocking Asynchronous case

1) If the ***Response Type*** parameter is configured as "nonBlockingRequestAsynch" (non-blocking asynchronous case), the Originator (CoAP client) should send the oneM2M request to the Receiver (CoAP server) as a CoAP Confirmable message. This request shall be sent using a CoAP POST method, and shall include the ***Operation*** parameter, mapped as described in clause 6.2.2.3.

2) The Receiver, after validating the request and before processing it fully, shall return a oneM2M response to the originator. It may either piggyback (2a) this response on the CoAP ACK message (if the request was sent as a CON message), or send the response as a separate CoAP CON message after it has sent the CoAP ACK (2b).

* If the Receiver supports the <request> resource type, it shall respond with a 2.01 (Created) CoAP response code and a oneM2M ***Response Status Code*** of "ACCEPTED for nonBlockingRequestAsynch". The response shall include the URI of the new <request> resource in a sequence of one or more Location-Path and/or Location-Query Options.
* If the Receiver does not support the <request> resource type, it shall respond with a 2.04 (Changed) CoAP response code and a oneM2M ***Response Status Code*** of "ACCEPTED for nonBlockingRequestAsynch".

3) The Receiver, upon successful processing of the request, shall send a new CoAP Confirmable request message using POST method. This message contains a oneM2M NOTIFY primitive whose content contains the response to the original request.

4) The Originator may either piggyback a response to this request (4a) or send it as a separate CoAP response after the acknowledgment message (4b). This response shall contain the appropriate CoAP response code as defined in table 6.2.4-1 and have an empty payload.



Figure 6.3.2-1: Non-Blocking Asynchronous Case

### 6.3.3 Non-Blocking Synchronous case

1) If the ***Response Type*** parameter is configured as "nonBlockingRequestSynch" (non-blocking synchronous case), the Originator (CoAP client) should send the oneM2M request to the Receiver (CoAP server) as a CoAP Confirmable message. This request shall be sent using a CoAP POST method, and shall include the ***Operation*** parameter, mapped as described in clause 6.2.2.3.

2) The Receiver, after validating the request and before processing it fully, shall return a oneM2M response to the originator. It may either piggyback this response (2a) on the CoAP ACK message (if the request was sent as a CON message) or send the response as a separate CoAP CON message after it has sent the CoAP ACK (2b).

* If the Receiver supports the <request> resource type, it shall respond with a 2.01 (Created) CoAP response code and a oneM2M ***Response Status Code*** of "ACCEPTED for nonBlockingRequestSynch". The response shall include the URI of the new <request> resource in a sequence of one or more Location-Path and/or Location-Query Options.
* If the Receiver does not support the <request> resource type, it shall respond with a 5.01 (Not implemented) CoAP response code and a oneM2M ***Response Status Code*** of "NON\_BLOCKING\_REQUEST\_NOT\_SUPPORTED".

3) The Originator can use the <request> resource reference to synchronously retrieve the <request> resource that contains the response to the original request.

4) The Receiver, upon receipt of this retrieve request, shall handle it as in clause 6.3.1 since it is a non-blocking request.

NOTE: If the Receiver is a Transit CSE, the Receiver acts as CoAP client and CoAP server.



Figure 6.3.3-1: Non-Blocking Synchronous Case

### 6.3.4 Flex Blocking case

1) If the ***Response Type*** parameter is configured as "flex blocking", the Originator (CoAP client) should send the oneM2M request to the Receiver (CoAP server) as a CoAP Confirmable message. This request shall be sent using a CoAP POST method, and shall include the ***Operation*** parameter, mapped as described in clause 6.2.2.3.

2) The Receiver shall determine whether to handle the request using "nonBlockingRequestSynch" or "nonBlockingRequestAsynch" mode:

* If the Receiver chooses "nonBlockingRequestAsynch" processing proceeds as described in clause 6.2.2, starting from step 2).
* If the Receiver chooses "nonBlockingRequestSynch" processing proceeds as described in clause 6.2.3, starting from step 2).

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Change 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start of Change 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 5.1 Required Features

This clause explicitly specifies the required features of the CoAP layer for oneM2M to properly bind oneM2M primitives into CoAP messages:

* The 4-byte binary CoAP message header is defined in section 3 of IETF RFC 7252 [1].
* Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK) and Reset (RST) messages shall be supported. The Reset message is used to send an error message in response to a malformed Confirmable message in CoAP layer.
* GET, PUT, POST and DELETE methods shall be supported. oneM2M primitives map to these methods.
* The CoAP Response Codes specified in clause 6.2.4 shall be supported for oneM2M ***Response Status Code*** parameter mapping.
* The Uri-Host, Uri-Port, Uri-Path, and Uri-Query shall be supported.
* The Content-Type Option shall be used to indicate the media types of the payload.
* Block-wise transfers feature may be supported to carry large payloads.
* The Caching feature may be supported.

## 5.2 Introduction to CoAP

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Change 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start of Change 3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### 6.2.1 Header

This clause specifies how to configure CoAP header information:

* The Version field shall be configured as 1.
* The Type field shall be configured according to clause 6.3. The Reset message is used to indicate an error in response to a malformed message in CoAP layer.
* In case of a request, the Code field indicates the CoAP Method. If the oneM2M operation is sent as a Blocking request the oneM2M ***Operation*** parameter shall be mapped to a CoAP Method according to the table 6.2.1-1. In non-blocking and flex blocking cases, the request shall use the CoAP POST method, and the Operation parameter shall be mapped as described in clause 6.2.2.3.
* In case of a response, the Code field indicates the CoAP Response Code. The oneM2M ***Response Status Code*** parameter shall be mapped to a CoAP Response Code as specified in clause 6.2.4.
* The Originator and Receiver shall set the 16 bit MessageId in accordance with the CoAP specification [1] and shall retry transmission of all unacknowledged Confirmable messages, as required by that specification.

Table 6.2.1-1: oneM2M Operation Parameter Mapping

|  |  |  |
| --- | --- | --- |
| oneM2M Operation Parameter | CoAP Method | CoAP Method Code |
| CREATE | POST | 0.02 |
| RETRIEVE | GET | 0.01 |
| UPDATE | PUT | 0.03 |
| DELETE | DELETE | 0.04 |
| NOTIFY | POST | 0.02 |

At the Receiver, a CoAP request message with a POST method that does not carry an ***Operation*** parameter shall be mapped to a oneM2M CREATE or NOTIFY operation in accordance with the existence of the ***Resource Type*** parameter. If a ***Resource Type*** parameter exists then the value of the ***Operation*** parameter is CREATE and if the ***Resource Type*** parameter does not exist, the value of the ***Operation*** parameter is NOTIFY.

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Change 3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Start of Change 4 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 3 Abbreviations and acronyms

For the purposes of the present document, the following abbreviations and acronyms apply:

ACK CoAP Acknowledgement message

AE Application Entity

CON CoAP Confirmable message

CSE Common Service Entity

DTLS Datagram Transport Layer Security

HTTP Hyper Text Transfer Protocol

IANA Internet Assigned Numbers Authority

IP Internet Protocol

NON CoAP Non-confirmable message

RST CoAP ReSeT message

TCP Transport Control Protocol

TLS Transport Layer Security

TLV Tag - Length - Value (data structure)

UDP User Datagram Protocol

URI Uniform Resource Identifier

XML eXtensible Markup Language

### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Change 4 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*