|  |
| --- |
| CHANGE REQUEST |
| Meeting ID:\* |  |
| Source:\* |  |
| Date:\* |  |
| Reason for Change/s:\* |  |
| CR against: Release\* |  |
| CR against: WI\* | [x]  Active <> [ ]  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

Conversion to Markdown

<https://git.onem2m.org/specifications/ts-0020/-/merge_requests/2>

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



oneM2M logo

**oneM2M Technical Specification**

|  |  |
| --- | --- |
| Document Number | TS-0020-V4.0.0 |
| Document Name: | WebSocket Protocol Binding |
| Date: | 2022-12-01 |
| Abstract: | WebSocket Protocol Binding TS |
| Template Version:23 February 2015 (Do not modify) | Template Version:23 February 2015 (Do not modify) |

This Specification is provided for future development work within oneM2M only. The Partners accept no liability for any use of this Specification.

The present document has not been subject to any approval process by the oneM2M Partners Type 1. Published oneM2M specifications and reports for implementation should be obtained via the oneM2M Partners’ Publications Offices.

About oneM2M

The purpose and goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide.

More information about oneM2M may be found at: http//www.oneM2M.org

Copyright Notification

1. 2022, oneM2M Partners Type 1 (ARIB, ATIS, CCSA, ETSI, TIA, TSDSI, TTA, TTC).

All rights reserved.

The copyright extends to reproduction in all media.

Notice of Disclaimer & Limitation of Liability

The information provided in this document is directed solely to professionals who have the appropriate degree of experience to understand and interpret its contents in accordance with generally accepted engineering or other professional standards and applicable regulations. No recommendation as to products or vendors is made or should be implied.

NO REPRESENTATION OR WARRANTY IS MADE THAT THE INFORMATION IS TECHNICALLY ACCURATE OR SUFFICIENT OR CONFORMS TO ANY STATUTE, GOVERNMENTAL RULE OR REGULATION, AND FURTHER, NO REPRESENTATION OR WARRANTY IS MADE OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR AGAINST INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. NO oneM2M PARTNER TYPE 1 SHALL BE LIABLE, BEYOND THE AMOUNT OF ANY SUM RECEIVED IN PAYMENT BY THAT PARTNER FOR THIS DOCUMENT, WITH RESPECT TO ANY CLAIM, AND IN NO EVENT SHALL oneM2M BE LIABLE FOR LOST PROFITS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES. oneM2M EXPRESSLY ADVISES ANY AND ALL USE OF OR RELIANCE UPON THIS INFORMATION PROVIDED IN THIS DOCUMENT IS AT THE RISK OF THE USER.

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

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

## 5.2 Binding Overview

WebSocket binding may be employed for communication between any two endpoints which can be connected over the Mca, Mcc or Mcc’ interface reference points supported by the oneM2M Architecture as shown in figure 6.1-1 of oneM2M TS-0001 [2].

When using the WebSocket protocol, one communication endpoint shall act as the WebSocket server. The WebSocket server listens for inbound handshake messages arriving from any WebSocket client to which a WebSocket connection is not yet established. Whether a communication endpoint takes the role of the client or the server shall depend on the registration relationship between the communicating entities as follows: the registree shall always use a WebSocket client, while the associated registrar shall always use a WebSocket server on the respective reference point.

This implies that ADN and ASN always take the role of a WebSocket client when WebSocket binding is employed. An MN-CSE uses a WebSocket server to communicate with its registrees and a WebSocket client to communicate with its own registrar (which can be another MN-CSE or an IN-CSE).

The IN-CSE provides a WebSocket server functionality to communicate with all its registrees, i.e. within a service provider’s domain. On the Mcc’ reference points, i.e. for communication between IN-CSEs of different Service Provider domains, the IN-CSE shall provide both WebSocket client and server functionality. This enables any IN-CSE to open a WebSocket connection to any IN-CSE of another Service Provider’s domain.

Figure 5.2-1 shows some applicable example system configuration.



Figure 5.2-1: Example scenarios of WebSocket client and server configurations

There exists a maximum of one WebSocket connection between two nodes. A WebSocket connection is established for the first time when the initial registration procedure of an entity to its registrar is performed. On an established WebSocket connection, request and response primitives can be exchanged in both directions. Any connection may be closed by either the WebSocket client or the server, depending on the communication schedule of either entity. However, the connection can be reopened from the client side only.

If the connection is closed temporarily, it shall be reopened when the next request primitive is sent from the client to the server side, or when the time to become reachable configured at <schedule> resource. If the WebSocket connection with the next-hop entity is not opened, and the WebSocket connection cannot be established due to lack of *pointOfAccess* address for the entity, a sending CSE may enable buffering of primitives which should be sent to the entity until the connection is reopened or their expiration time is reached. See Annex H of oneM2M TS-0004 [5] about buffering of primitives by CMDH functionality.

Figure 5.2-2 shows an example message flow for a scenario where an ADN-AE registers to its registrar MN-CSE using an unsecured TCP connection without proxy and then continues exchanging non-registration request and response primitives.





Figure 5.2-2: Example message flow with WebSocket binding

1. The ADN-AE wants to register to its registrar MN-CSE. If a WebSocket connection does not exist, it is established by the following steps 2) and 3). It is assumed that the ADN-AE knows the point of access (i.e. WebSocket URI specified in IETF RFC 6455 [1]) under which the registrar CSE can be reached with WebSocket binding.
2. The WebSocket client opens handshake to the server with subprotocol name oneM2M.json following IETF RFC 6455 [1]. If the server can be reached under the WebSocket URI ws://example.net:9000/, the client handshake may look as follows:

GET / HTTP/1.1
Host: mncse1234.net:9000
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key: ud63env87LQLd4uIV20/oQ==
Sec-WebSocket-Protocol: oneM2M.json
Sec-WebSocket-Version: 13

1. The WebSocket server replies with a handshake to the client. In the successful case, the status-line of this HTTP response may look as follow (note that text shown in brackets […] is not sent explicitely):

[Request-Version:] HTTP/1.1
[Status-Code:] 101
[Response-Phrase:] Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Protocol: oneM2M.json
Sec-WebSocket-Accept: FuSSKANnI7C/6/FrPMt70mfBY8E=

1. The ADN-AE issue a registration request primitive. The request primitive may e.g. look as follows as JSON-serialized representation (note that only mandatory parameters of the request primitive are shown in this example; the message may include any optional primitive parameters in addition, e.g. “fr”):

{
"op": 1,
"to": "//example.net/mncse1234",
"rqi": "A1234",
"pc": {
"m2m:ae": {
"api": "a56",
"apn": "app1234"
}
},
"ty": 2
}

1. WebSocket Binding process, which transforms a single oneM2M primitive into one or more data frames of the WebSocket Framing protocol, as specified in IETF RFC 6455 [1]. When transmitting a JSON-serialized primitive in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment will be set to x1 (“text frame”).
2. The WebSocket message (consisting of one or more frames) shall be sent to the WS server.
3. The original request primitive shall be unpacked from the WebSocket message by the WS server.
4. The request primitive is delivered to the MN-CSE.
5. The MN-CSE performs the receiver side operations of AE registration as specified in oneM2M TS-0001 [2].
6. The response primitive is issued to the WebSocket server.
7. WebSocket binding process for the response primitive is performed.
8. The WebSocket message (consisting of one or more frames) is sent to the client.
9. The response primitive is unpacked.
10. The response primitive is to the ADN-AE.
11. After successful completion of AE registration any other CRUDN requests and response primitives can be exchanged over the existing WebSocket connection in both directions. If the ADN-AE has no other requests to send, the WebSocket connection may be closed temporarily. When the WebSocket connection is closed after registration and reopened later again, the registration procedure as outlined in steps 4 to 14 is omitted. In this case any non-registration request primitives can be sent directly.

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

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

## A.1 AE Registration and creation of a container child resource

Figure A.1-1 illustrates a message flow for registration of an ADN-AE to an IN-CSE as described in clause 7.3.5.2.1 of oneM2M TS-0004 [5] with WebSocket mapping and subsequent creation of a <container> child resource.



Figure A.1-1: Message flow for registration of an ADN-AE to an IN-CSE

In the considered example, the WebSocket protocol is used to send JSON serialized request and response primitives in text format.

The message flow may look as follows:

1. TCP connection establishment and Security Association Establishment as defined in oneM2M TS-0003 [4] based on TLS handshake procedure is accomplished.
2. The WSS client sends e.g. the following opening handshake message, offering to use either JSON or XML serialization of primitives:

GET / HTTP/1.1
Host: mncse1234.net:9000
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key: ud63env87LQLd4uIV20/oQ==
Sec-WebSocket-Protocol: oneM2M.json, oneM2M.xml
Sec-WebSocket-Version: 13

1. The WSS server selects use of JSON serialization and responds the following handshake message:

Request-Version: HTTP/1.1
Status-Code: 101
Response-Phrase: Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Protocol: oneM2M.json
Sec-WebSocket-Accept: FuSSKANnI7C/6/FrPMt70mfBY8E=

1. The AE sends the following request primitive in textual JSON serialized format:

{
"op": 1,
"to": "//example.net/mncse1234",
"rqi": "A1000",
"rcn": 7,
"pc": {
"m2m:ae": {
"rn": "SmartHomeApplication",
"api": "Na56",
"apn": "app1234"
}
},
"ty": 2
}

The above JSON object is mapped by the WS client into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 (“text frame”).

1. The IN-CSE validates the privilege of the originator to create an <AE> resource, and accepts the request to create the resource.
2. The IN-CSE acknowledges the success of the request by responding the following JSON serialized response primitive. The response primitive includes all attributes of <AE> instance created in Step 5.

{
"rsc": 2001,
"rqi": "A1000",
"pc": {
"m2m:ae": {
"rn": "SmartHomeApplication",
"ty": 2,
"ri": "ae1",
"api": "Na56",
"apn": "app1234",
"pi": "cb1",
"ct": "20160506T153208",
"lt": "20160506T153208",
"acpi": [
"acp1",
"acp2"
],
"et": "20180506T153208",
"aei": "S\_SAH25"
}
}
}

NOTE: JSON serialized primitives are not encapsulated under member names “m2m:rqp” and “m2m:rsp” as in XML serialized representations, which allows differentiation between request and response primitives (see clause 8.4 of TS-0004 [5]). JSON serialized primitives can be differentiated by the presence of mandatory members such as “op” in request primitives (see step 4) above), and “rsc” in response primitives.

The above JSON object is mapped by the WS server into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 (“text frame”). 1. The AE sends in textual JSON serialized format the following request primitive to create a <container> resource as child resource of the <AE> created in Step 5:

{
"op": 1,
"to": "//example.net/mncse1234/SmartHomeApplication",
"fr":"S\\_SAH25",
"rqi": "A1001",
"rcn": 7,
"pc": {
"m2m:cnt": {
"rn": "SmartHomeContainer",
"mbs": 100000,
"mni": 500
}
},
"ty": 3
}

The above JSON object is mapped by the WS client into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 (“text frame”).

1. The IN-CSE validates the privilege of the originator to create an <container> resource under the <AE> resource created in step 5, and accepts the request to create the resource.
2. The IN-CSE acknowledges the success of the request by responding the following JSON serialized response primitive:

{
"rsc": 2001,
"rqi": "A1001",
"pc": {
"m2m:cnt": {
"rn": "SmartHomeContainer",
"ty": 3,
"ri": "cnt1",
"pi": "ae1",
"ct": "20160506T154048",
"lt": "20160506T154048",
"acpi": [
"acp1"
],
"et": "20180506T154048",
"cr": " S\_SAH25",
"st": 0,
"mni": 500,
"mbs": 100000,
"cni": 0,
"cbs": 0,
"mia": 3600
}
}
}

The above JSON object is mapped by the WS server into a data frame of the WebSocket Framing protocol in utf-8 text format, the 4-bit opcode in the WebSocket Base Framing Protocol of the first message fragment is set to x1 (“text frame”). 1. Primitives of further subsequent CRUDN procedures may be transferred on the existing WebSocket connection.

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