Figure 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)

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

Contents

1 Scope

The present document specifies the binding of Mca and Mcc primitives onto the WebSocket binding.

It specifies:

- Procedures and message formats for operating and closing of WebSocket connections.
- How request and response primitives are mapped into the payload of the WebSocket protocol.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. For specific references, only the cited

version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

The following referenced documents are necessary for the application of the present document.

- [1] IETF RFC 6455 (December 2011): "The WebSocket Protocol".
- [2] oneM2M TS-0001: "Functional Architecture".
- [3] IETF RFC 7230 (June 2014): "Hypertext Transport Protocol (HTTP/1.1): Message Syntax and Routing".
- [4] oneM2M TS-0003: "Security solutions".
- [5] oneM2M TS-0004: "Service Layer Core Protocol Specification".
- [6] IETF RFC 7692 (December 2015): "Compression Extension for WebSocket".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or nonspecific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

- [i.1] oneM2M Drafting Rules. > NOTE: Available at http://www.onem2m.org/images/files/oneM2M-Drafting-Rules.pdf.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

oneM2M WebSocket Client (WS Client) : WebSocket Client associated with an AE or a CSE capable of establishing the WebSocket connections

oneM2M WebSocket Server (WS Server) : WebSocket Server associated with a CSE which accepts requests to establish WebSocket connections

3.2 Abbreviations

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

```
ADN       Application Dedicated Node
AE        Application Entity
ASN       Application Service Node
```

CBOR	Concise Binary Object Representation
CMDH	Communication Management and Delivery Handling
CRUDN	Create Retrieve Update Delete Notify
CSE	Common Services Entity
FQDN	Fully Qualified Domain Name
GUID	Globally Unique Identifier
HTTP	Hypertext Transport Protocol
IETF	Internet Engineering Task Force
IN-CSE	Infrastructure Node Common Services Entity
IP	Internet Protocol
JSON	JavaScript Object Notation
MN	Middle Node
MN-CSE	Middle Node Common Services Entity
NAT	Network Address Translator
RFC	Request for Comments
SAEF	Security Association Establishment Framework
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
WS	WebSocket
WSS	WebSocket Secure
XML	eXtensible Markup Language

4 Conventions

The key words “Shall”, “Shall not”, “May”, “Need not”, “Should”, “Should not” in the present document are to be interpreted as described in the oneM2M Drafting Rules [i.1].

5 Overview on WebSocket Binding

5.1 Use of WebSocket

This binding makes use of the WebSocket protocol IETF RFC 6455 [1] to transport serialized representations of oneM2M request and response primitives over the Mca or Mcc reference points.

Establishment of a WebSocket connection shall be initiated by a WebSocket client by sending a handshake to a WebSocket server as specified in section 4 of IETF RFC 6455 [1]. Once the WebSocket connection is established, both oneM2M request and response primitives can be exchanged bi-directionally between the two endpoints of the connection. Serialized representations of the request and response primitives shall be mapped in the Payload Data field of the WebSocket base framing protocol, as defined in section 5.2 of IETF RFC 6455 [1].

A WebSocket connection employs either a TCP/IP or a TLS over TCP/IP

connection. The underlying TCP and TLS connections are established prior to sending the WebSocket client handshake as the first step (see example in Annex A).

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.

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

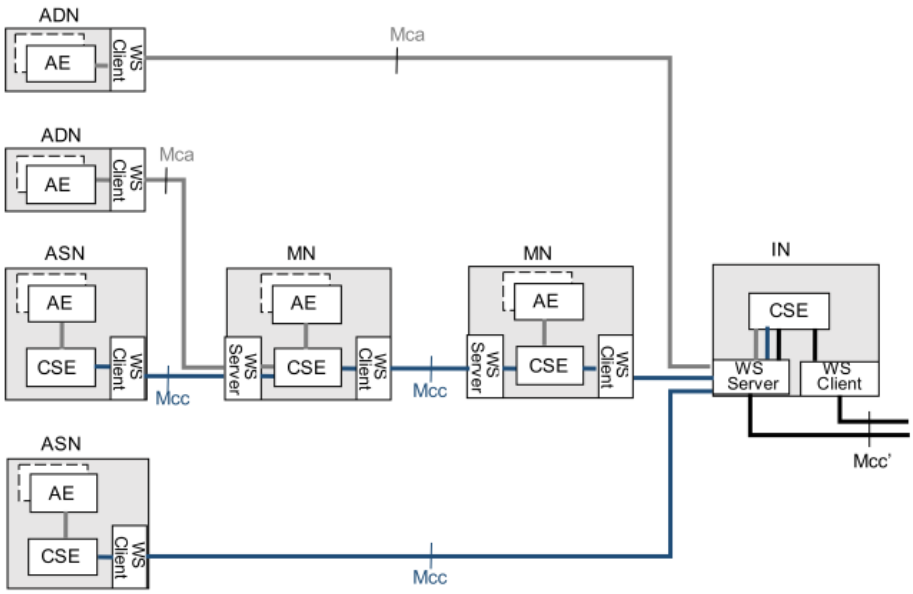

Figure 2: Figure 5.2-1: Example scenarios of WebSocket client and server configurations

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.

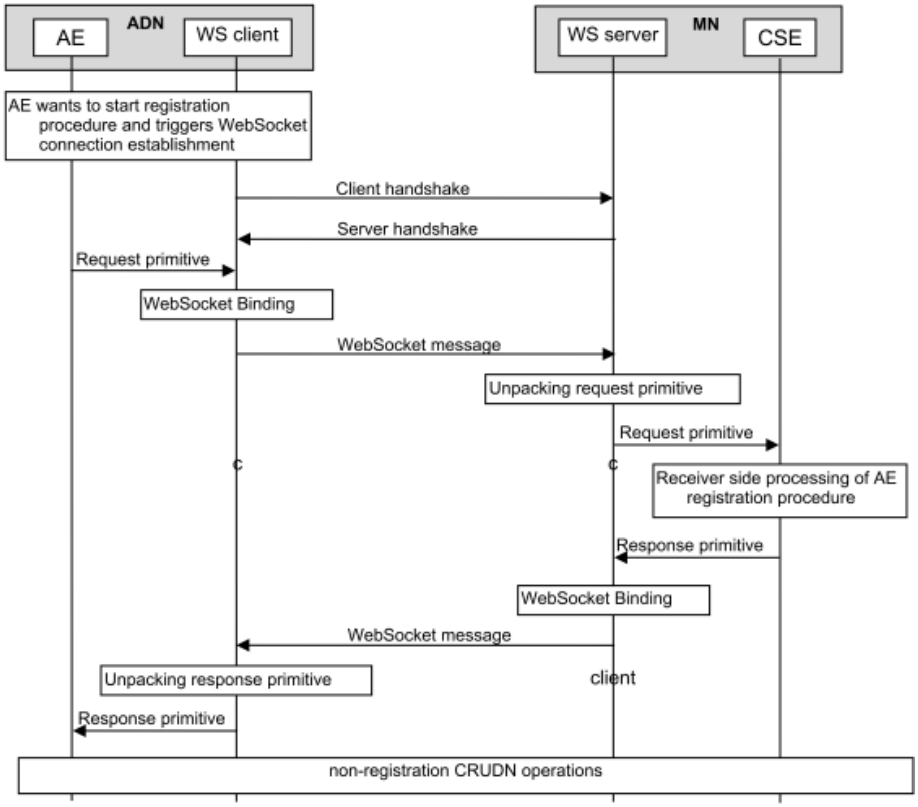

Figure 3: Figure 5.2-2: Example message flow with WebSocket binding

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.

6 Protocol Binding

6.1 Introduction

The WebSocket protocol enables two-way communication between client and server even when a firewall and/or NAT are present between them. This means, once a WebSocket connection is established, request (and response) primitives can be exchanged in both directions, from the client to the server and vice versa. However, AEs may be capable of handling Notification request primitives only, or no request primitives at all.

WebSocket binding applied by oneM2M entities/nodes shall be fully compliant with IETF RFC 6455 [1]. After establishment of a WebSocket connection between two nodes, at the transmitter side each individual request and response primitive is mapped into one or several WebSocket frames.

6.2 WebSocket connection establishment

6.2.1 General

A WebSocket connection is opened by the client side as specified in section 4 of IETF RFC 6455 [1] with sending of a client handshake. The server responds with a server handshake.

The client handshake consists of an HTTP upgrade request, along with a list of required and optional header fields.

The handshake shall be a valid HTTP request as specified by IETF RFC 7230 [3]. The server handshake consists of a HTTP status-line and a list of header fields.

The applicable format of the request-line, status-line and the applicable header fields are specified in the following sub-clauses.

HTTP headers fields have case-insensitive field names.

CSEs capable to support WebSocket shall indicate the schemes ws and/or wss together with the applicable host name and port numbers in the *pointOfAccess* attribute of their <CSEBase> and in the <remoteCSE> resources, i.e. as ws://host:port1 and wss://host:port2, where host refers to either an IP address or an FQDN.

By default, the WebSocket Protocol [1] uses port 80 for regular WebSocket connections and port 443 for WebSocket connections over Transport Layer Security (TLS). If a WebSocket URI does not include an explicit port number, the default port number shall apply. Possible example representations of the *pointOfAccess* attribute <CSEBase> or <remoteCSE> resources associated with entities supporting a WebSocket server are the following:

- ws://ws-server.example.com:80
- ws://ws-server.example.com
- wss://10.251.232.119:443

NOTE: ADN-AEs and ASN-CSEs do not need to support WebSocket servers and therefore do not require a WebSocket URI in the *pointOfAccess* attribute (see figure 5.2.1).

6.2.2 Client handshake

6.2.2.1 Format of request-line The request-line of a client handshake shall begin with the method token "GET", followed by the request target "/" and the HTTP version set to "HTTP/1.1" as follows:

```
GET / HTTP/1.1
```

If the client is configured to use a proxy when using the WebSocket Protocol, a connection to the proxy server shall be established prior to sending the above client handshake. This is described in clause 6.6.

6.2.2.2 Host header The Host header shall be present in each client handshake.

The Host header indicates the FQDN or IP address of the Receiver CSE of the next hop. If the originator of the client handshake is an oneM2M field entity, the host header represents the registrar CSE of the originator.

When no proxy is used, the Host header shall be set as one of the pointOfAccess attribute values associated with the Receiver. Selection of the appropriate Receiver is described in oneM2M TS-0004 [5].

If the client is configured to use a proxy when using the WebSocket Protocol, then the client should connect to that proxy and ask it to open a TCP connection to the host and port rather than to the next hop CSE.

6.2.2.3 Upgrade header The Upgrade header shall be present in each client handshake message with value WebSocket as follows:

```
Upgrade: WebSocket
```

6.2.2.4 Connection header The Connection header shall be present in each client handshake message with value Upgrade as follows:

```
Connection: Upgrade
```

6.2.2.5 Sec-WebSocket-Key header The Sec-WebSocket-Key header shall be present in each client handshake message. The header field includes a base64-encoded representation of a random 16 bytes pattern, for example:

```
Sec-WebSocket-Key: ud63env87LQLd4uIV20/oQ==
```

6.2.2.6 Sec-WebSocket-Version header The Sec-WebSocket-Version header shall be present in each client handshake message with value 13 as follows:

```
Sec-WebSocket-Version: 13
```

6.2.2.7 Sec-WebSocket-Protocol header The Sec-WebSocket-Protocol header shall be present in a client handshake message. It enables the client to indicate its supported application subprotocols on the server and be sure that the server agreed to support that subprotocol. It is used by the client to indicate the oneM2M Service Layer Protocol version and supported serialization formats to the server.

The value of the Sec-WebSocket-Protocol header shall be one or more of the registered names defined in clause 6.2.2.9. It shall also be allowed to include multiple Sec-WebSocket-Protocol headers with a value that includes one registered name each as defined in IETF RFC 6455 [1], for example:

```
Sec-WebSocket-Protocol: oneM2M.json, oneM2M.xml
```

and

```
Sec-WebSocket-Protocol: oneM2M.xml
Sec-WebSocket-Protocol: oneM2M.json
```

are equivalent headers, expressing that the WebSocket client supports both application subprotocols, oneM2M.json and oneM2M.xml. The order of names indicated in the Sec-WebSocket-Protocol header specifies the client's preference.

6.2.2.8 Sec-WebSocket-Extensions header The Sec-WebSocket-Extensions header may be used to negotiate the use of per-message compression as specified in IETF RFC 7692 [6].

If the client handshake includes the header, e.g.

```
Sec-WebSocket-Extensions: permessage-deflate
```

it indicates to the server the client's preference to apply the compression mechanism defined in IETF RFC 7692 [6]. The header may include additional parameters as specified in IETF RFC 7692 [6].

When the server accepts use of message compression it responds with a Sec-WebSocket-Extensions header in the server handshake message as specified in clause 6.2.3.6, and in this case compression is applied in both transmission directions. If the server handshake message does not include a Sec-WebSocket-Extensions header, compression shall not be applied.

6.2.2.9 Subprotocol names and serialization formats The Sec-WebSocket-Protocol header in the opening handshake is used to negotiate the application protocol layered on top of WebSocket. The application protocol addressed in this specification is the Release-2 version of the oneM2M Service Layer.

The oneM2M Service Layer Protocol consists of the exchange of serialized representations of request and response primitives as defined in oneM2M TS-0001 [2] and oneM2M TS-0004 [5]. This version of the specification allows use of the serialization formats listed in table 6.2.2.9-1. Both, protocol version and serialization format are associated with a specific subprotocol name.

Table 6.2.2.9-1 lists the serialization formats, associated subprotocols names and opcode setting of the WebSocket Frame protocol applicable for the present version of this specification.

Table 2: Table 6.2.2.91: Applicable Subprotocol names

Serialization Format	Subprotocol Name	WS opcode	Notes
JSON	oneM2M.json	x1 ("text frame")	See clause 8.4 in oneM2M TS-0004 [5]
XML	oneM2M.xml	x1 ("text frame")	See clause 8.3 in oneM2M TS-0004 [5]
CBOR	oneM2M.cbor	x2 ("binary frame")	See clause 8.5 in oneM2M TS-0004 [5]

6.2.3 Server handshake format

6.2.3.1 Format of status-line The status-line of a server handshake shall begin with the HTTP version set to "HTTP/1.1", followed by the status code and reason phrase as defined in IETF RFC 6455 [1]. When the WebSocket connection is established successfully, the status-line may look as follows:

```
HTTP/1.1 101 Switching Protocols
```

For the unsuccessful connection establishment, any appropriate HTTP error status code shall be returned with optional addition of a corresponding reason phrase.

6.2.3.2 Upgrade header The Upgrade header shall be present in each server handshake message with value WebSocket as follows:

```
Upgrade: WebSocket
```

6.2.3.3 Connection header The Connection header shall be present in each server handshake message with value Upgrade as follows:

```
Connection: Upgrade
```

6.2.3.4 Sec-WebSocket-Accept header The Sec-WebSocket-Accept header shall be present in each server handshake message. The header field shall be constructed from the Sec-WebSocket-Key value and the GUID as specified in section 4.2.2 of IETF RFC 6455 [1]. It may look e.g. as follows:

```
Sec-WebSocket-Accept: FuSSKANnI7C/6/FrPMt70mfBY8E=
```

6.2.3.5 Sec-WebSocket-Protocol header The Sec-WebSocket-Protocol header shall be present in a server handshake message. It indicates to the client that the server accepts (one of) the subprotocol(s) indicated by the client.

The server compliant with this specification shall select one of the subprotocol names indicated in the Sec-WebSocket-Protocol header of the client handshake message and set the value of the Sec-WebSocket-Protocol header of the server handshake message accordingly.

6.2.3.6 Sec-WebSocket-Extensions header If the optional Sec-WebSocket-Extensions header with value "permessage-deflate" was included in the client handshake message, the Sec-WebSocket-Extensions header with same value shall also be included into the server handshake message, if the server accepts usage of message compression, and apply message compression in the transmit direction and message decompression in the receive direction as defined in IETF RFC 7692 [6].

If the server does not accept message compression, it shall not include the Sec-WebSocket-Extensions header.

6.3 Closing WebSocket connection

Compliant with section 7 of IETF RFC 6455 [1] a WebSocket connection shall be closed by sending a Connection Close Frame (opcode x8). Both, client and server may initiate a closing handshake of an existing WebSocket connection at any time.

WebSocket connections should be kept open for as long as possible considering any given constraints due to communication policies and power saving requirements. Unless communication policies enforce the closing of network access, it is left to implementation to decide when exactly the closing of a WebSocket shall be triggered.

6.4 Registration procedure

A oneM2M entity (AE or CSE) not yet registered to its registrar CSE needs to be preconfigured with various parameters as specified in oneM2M TS-0001 [2] and oneM2M TS-0003 [4] in order to be able to send the registration request primitive (i.e. create *<AE>* or create *<remoteCSE>* request primitive). To establish a WebSocket connection, the WebSocket client shall be configured with an applicable point of access of its registrar CSE which includes FQDN or IP address and the port number.

After the Registration procedure has been successfully completed, the WebSocket Server (e.g. Registrar CSE for WebSocket Client) shall enable routing of any incoming oneM2M primitives to this registree.

Before the Registration procedure is successfully completed, any incoming oneM2M primitives to the WebClient shall be rejected by the Receiver (e.g. registrar CSE).

Closing of the WebSocket connection after registration does not impact the registration status of an AE or CSE to its registrar, unless an explicit deregistration procedure is performed by deletion of the respective *<AE>* or *<remoteCSE>* resource instance.

6.5 Handling of Non-Registration Request

Registered entities (AE and CSE) are allowed to send and receive non-registration request primitives. A WebSocket connection should support any of the transfer modes defined in clause 8.2 of oneM2M TS-0001 [2], i.e. blocking requests, and non-blocking requests for both synchronous and asynchronous cases.

When sending blocking requests, the WebSocket connection shall not be closed before the response is received, or before any configured timeout period has expired.

When sending non-blocking requests, the WebSocket connection shall not be closed before the acknowledgment response is received, or before any configured

timeout period has expired. If the entities' communication policies and power saving requirements allow, the connection should be kept open at least until an ongoing procedure has fully completed, i.e. requesting of the result in synchronous mode or completion of Notify procedure in asynchronous mode.

If no WebSocket connection with a client exists when a Notify request primitive for this client becomes available at the server side, it should be stored and sent when the WebSocket connection is opened again by the client.

6.6 Use of proxy servers

The connection to a proxy shall be requested by sending a request-line with the method token "CONNECT", followed by the request target host and port of the WebSocket server and the HTTP version set to "HTTP/1.1" for example as follows:

```
CONNECT WSserver.example.com:80 HTTP/1.1
```

7 Security Aspects

Authentication and Transport Layer Security can be established when the oneM2M entity which hosts the WebSocket Server can be addressed with the wss URI scheme. When using the wss URI scheme, one of the Security Association Establishment Frameworks (SAEF) as defined in oneM2M TS-0003 [4] shall be applied to provide mutually authenticated Transport Layer Security between the communicating entities prior to sending the WebSocket client handshake.

The SAEF is accomplished by successful completion of a TLS handshake procedure before the client sends its opening handshake message. The details of SAEF and possibly required Remote Security Provisioning Frameworks are specified in oneM2M TS-0003 [4].

In special deployment scenarios, e.g. when the communicating oneM2M entities using WebSocket binding are located in a secure environment and/or implemented on the same device, Transport Layer Security may not be required. In such scenarios unsecured WebSocket communication addressed with the ws URI scheme may be adequate.

Annex A (informative): Example Procedures

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.

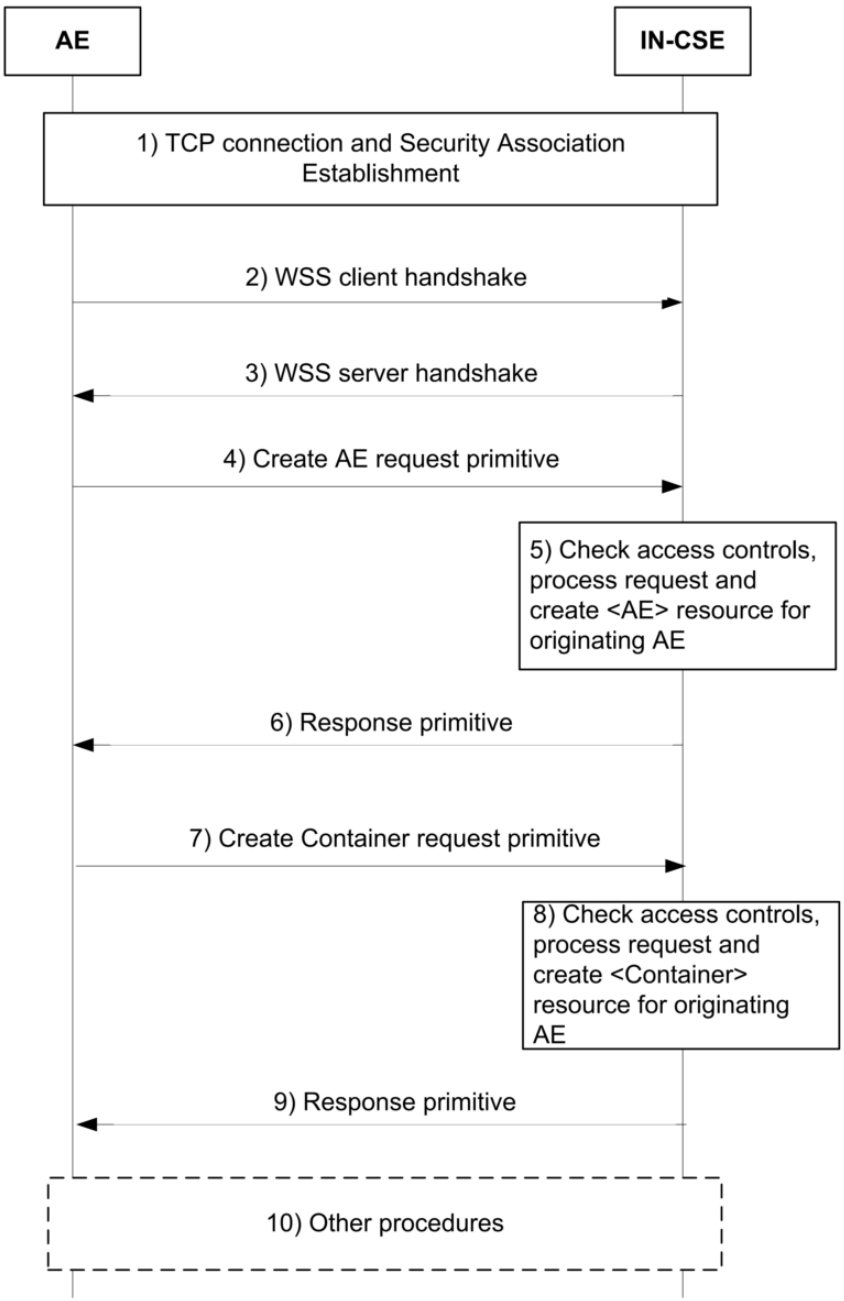

Figure 4: Figure A.1-1: Message flow for registration of an ADN-AE to an IN-CSE

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
```

3. 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=
```

4. 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”).

5. The IN-CSE validates the privilege of the originator to create an <AE> resource, and accepts the request to create the resource.

6. 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”).

7. 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").

8. 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.

9. 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”).

10. Primitives of further subsequent CRUDN procedures may be transferred on the existing WebSocket connection.

History

Publication history	Publication history	Publication history
V4.0.0	Dec 2022	Release 4 - Publication