Open Connection Manager WebAPI

Candidate Version 1.1 – 17 Feb 2015

Open Mobile Alliance

OMA-TS-OpenCMAPI_Web_V1_1-20150217-C

Contents

Figures

No table of figures entries found.

Tables

1. Scope

This specification of the OpenCMAPI defines interfaces (derived from [OpenCMAPI-TS]), through which connection management services are made available to Web applications.

The specification addresses the requirements enumerated in [OpenCMAPI-RD] and adheres to the architecture described in [OpenCMAPI-AD].

2. References

2.1 Normative References

[JSON-RPC] "JSON RPC (Remote Procedure Call) Specification 2.0", URL: http://www.jsonrpc.org/specification

[OpenCMAPI-AD] "Open Connection Manager API Architecture", Open Mobile Alliance™, OMA-AD-OpenCMAPI-V1_1, URL: http://www.openmobilealliance.org/

[OpenCMAPI-RD] "Open CM API Requirements", Open Mobile Alliance™, OMA-RD-OpenCMAPI-V1_1, URL: http://www.openmobilealliance.org/

[OpenCMAPI-SUP-JSD] "JSON schema for the Open Connection Manager API", Open Mobile Alliance™, OMA-SUP-JSD_ OpenCMAPI-V1_1, URL: http://www.openmobilealliance.org/

[OpenCMAPI-SUP-WIDL] "JSON schema for the Open Connection Manager API", Open Mobile Alliance™, OMA-SUP-WIDL_ OpenCMAPI-V1_1, URL: http://www.openmobilealliance.org/

[OpenCMAPI-TS] "Open Connection Manager API", Open Mobile Alliance™, OMA-TS-OpenCMAPI-V1_1, URL: http://www.openmobilealliance.org/

[RFC1034] "DOMAIN NAMES - CONCEPTS AND FACILITIES", P. Mockapetris, November 1987, URL:http://tools.ietf.org/html/rfc1034

[RFC1035] "DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION", P. Mockapetris, November 1987, URL:http://tools.ietf.org/html/rfc1035

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding et. al, January 1999, URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC6455] "The Web Socket Protocol", I. Fette and A. Melnikov, December 2011, URL: http://tools.ietf.org/html/rfc6455

[RFC7159] "The JavaScript Object Notation (JSON) Data Interchange Format",T. Bray, Ed., March 2014, URL:http://tools.ietf.org/html/rfc7159

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL: http://www.openmobilealliance.org/

[Wi-Fi Alliance HS2.0 TS] Hotspot 2.0 (Release 2) Technical Specification version 1.0.0, Wi-Fi Alliance Wi-Fi CERTIFIED Passpoint™ (Release 2) program, URL:https://www.wi-fi.org/Hotspot_2-0_(R2)_Technical_Specification_v1-0-0.pdf

2.2 Informative References

[JSON-Schema] "JSON Schema: core definitions and terminology", Francis Galiegue, Kris Zyp, Gary Court, URL:http://tools.ietf.org/html/draft-zyp-json-schema-04
Note: The referenced IETF draft is a work in progress, subject to change without notice.

[OMADICT] "Dictionary for OMA Specifications", Version 2.9, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_9, URL: http://www.openmobilealliance.org/

[RFC4122] "A Universally Unique IDentifier (UUID) URN Namespace", P. Leach, M. Mealling, R. Salz, July 2005, URL: http://www.ietf.org/rfc/rfc4122.txt

[RFC6202] "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", April 2011, URL:http://tools.ietf.org/rfc/rfc6202.txt

[W3C_WebSocket] "The WebSocket API", W3C Candidate Recommendation 20 September 2012, Ian Hickson, ed., URL:http://www.w3.org/TR/websockets/

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

For the purpose of this TS, all definitions from the OMA Dictionary apply [OMADICT].

Hotspot 2.0	Hotspot 2.0 [Wi-Fi Alliance HS2.0 TS] (also known as Passpoint) is a set of specifications from the Wi-Fi Alliance.
JSON	The JSON refers to the definition of [RFC7159].
Long Polling	A variation of the traditional polling technique, where the server does not reply immediatelly to a request unless a particular event, status or timeout has occurred. Once the server has sent a response, it closes the connection, and typically the client immediately sends a new request. This allows the emulation of a push mechanism from a server to a client.

3.3 Abbreviations

ANDSF	Access Network Discovery and Selection Function
API	Application Programming Interface
CM	Connection Manager
D2D	Device to Device
DNS	Domain Name System
GNSS	Global Navigation Satellite System
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
M2M	Machine to Machine
MIME	Multipurpose Internet Mail Extensions
OMA	Open Mobile Alliance
OpenCMAPI	Open Connection Manager (CM) Application Programming Interface (API)
PIN	Personal Identification Number
ProSe	Proximity Services (Also referred to as LTE D2D)
PUK	Personal Unlocking Key also called UNBLOCK PIN.
RFC	Request For Comments
RPC	Remote Procedure Call
SCR	Static Conformance Requirements
SMS	Short Message Service
TLS	Transport Layer Security
UICC	Universal Integrated Circuit card
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
USSD	Unstructured Supplementary Service Data
WLAN	Wireless Local Area Network

4. Introduction

With the multiplicity of networks available and the need for more connectivity, there is a market demand for a standardized WebAPI to provide connection management functionalities which would facilitate development and integration of Connection Manager Web Applications as well as to provide more status information about the connection to any Web application using mobile data services.

The goal of the OMA OpenCM WebAPI is to facilitate the development of Connection Manager Web Applications to the mobile environment and to provide additional services such as Information Status to Web applications relying on connectivity to mobile networks.

In this context, the Technical Specification for the Open Connection Management WebAPI defines a WebAPI binding for the [OpenCMAPI_TS] specification, i.e. it provides the CMAPI device discovery, a transport independent JSON-RPC payload data structure, return values, error codes and two transport bindings (native HTTP and WebSockets).

In the context of this specification a WebIDL [OpenCMAPI-SUP-WIDL] and a JSON schema [OpenCMAPI-SUP-JSD] is provided.

Appendix D. describes how a JavaScript Library implements the WebAPI based on WebSocket Transport binding.

4.1 Version 1.1

Version 1.1 is the first version which was produced of this document as WebAPI was not part of the scope of OpenCMAPI Enabler 1.0.

This version of the specification addresses the following aspects:

- Security and concurrency control function, e.g. access control and authorization
- Device Discovery & Device Handling
- Device Services
- Cellular Network Connection Management
- PIN/PUK Management
- Interaction with the UICC
- WLAN connection management including extensions to support of Hotspot 2.0, ANDSF & user and operator preferences
- Information Status handling
- Statistics handling
- GNSS handling
- SMS&USSD management
- Push Data service management
- Callbacks & Registration/Deregistration to receive callbacks
- Phone Book /Contacts management support
- Support of extended device services
- Support of P2P (or D2D or ProSe as known in 3GPP) Direct connection
- Router Management support
- Support of IP Multimedia Services functions
- Support of dedicated M2M/IoT functions

5. Detailed API specification

This section is organized to support a comprehensive understanding of the OpenCM WebAPI design. It specifies the device discovery, message payload data structures, transport bindings and error handling.

5.1 Device Discovery

All devices implementing the Web Binding of OpenCMAPI v1.1 SHALL register the name “cmapi.device” on its local network. Thus those devices SHALL be discovered by innate DNS ([RFC1034], [RFC1035]) resolution of “cmapi.device”.

5.2 Data Structures of CMAPI interface messages

This section defines the transport independent representation of CMAPI-1 and CMAPI-2 interfaces and explains how to translate CMAPI functions into the JSON messages.

The request and response messages are based on JSON-RPC 2.0 [JSON-RPC]. JSON-RPC is a stateless, light-weight remote procedure call based on JSON data format [RFC7158].

CMAPI functionality is implemented by using extended JSON-RPC data objects as the Application Data. A schema for each message is provided in [OpenCMAPI-SUP-JSD] that is based on [JSON-Schema].

Each function of CMAPI-1 and CMAPI-2 defined in [OpenCMAPI-TS] can be directly translated into JSON schema. To represent a function as JSON schema, the following steps are taken:

- Place the function name into appropriate places into the JSON schema
- Place the function’s input parameters into the JSON request schema using JSON data types
- Place the function’s output and return values into the JSON response schema using JSON data types
- Note: Hexdecimal values have to be translated into decimal values for JSON

All CMAPI JSON Schemas and CMAPI JSON examples are available in [OpenCMAPI-SUP].

5.2.1 JSON Data Types and Naming Conventions

This section defines the used data types and naming conventions used for CMAPI Messages in JSON schemas.

5.2.1.1 JSON Data Types

The JSON data types are:

- string
  Used for all UTF8* parameters

```
{
      " parameter": {
        "type": "string",
        "description": "Some Description."
      }

}
```

- integer
  Used for all byte, word, dword, and qword parameters

```
{
      " parameter": {
        "type": "integer",
        "description": "Some Description."
      }

}
```

- number
  Used for all floating-point parameters, if any

```
{
```

```
        " parameter": {
           "type": "integer",
           "description": "Some Description."
        }

}
```

- boolean
  Used for all boolean parameters

```
{
        " parameter": {
           "type": "boolean",
           "description": "Some Description."
        }

}
```

- array
  Used for all non-string arrays

```
{
  "parameter": {
     "type": "array",
     "description": "Some Description.",
     "items": {
        "type": "object",
        "properties": {
           "property_1": {
              "type": "JSON_Data_Type",
           },
           "property_n": {
              "type": "JSON_Data_Type
           }
        },
       "required": [ "property_1", "property_n " ]
     }
  },

}
```

- object
  Used for all data structure types

```
{
   "parameter": {
        "type": "object",
        "properties": {
           " property_1": {
              "type": " JSON_Data_Type ",
              "description": "Some Description."
           },
           "property_n": {
              "type": " JSON_Data_Type",
              "description": "Some Description."
           }
        },
       "required": ["property_1", "property_n " ]
   }
}
```

Note: There are no size restrictions on JSON data types.

5.2.1.2 Naming Conventions

This section describes the conventions used for the parameter names in the JSON schemas.

Names as defined in [OpenCMAPI-TS] are used with the additional considerations regarding case usage.

Two general cases are provided for, both using mixed case names; one with a leading capital letters, the other with a leading lowercase letters.

Names will start with a letter and be mixed case, with the leading letter of each but the first word capitalized. The conventions for the leading letter of the first differ depending on the context, as given below. Words will not be separated by white space, underscore, hyphen or other non-letter character.

All names will have a leading lowercase letter except if the name starts with an abbreviation.

If a name starts with or includes an abbreviation, all characters of that abbreviation are capitalized, e.g. “CMAPIMessage”.

For names consisting of concatenated words, all subsequent words start with a capital, for example, “concatenatedWord”.

5.2.2 CMAPI-1 Messages

CMAPI-1 request messages are originated from the client, response messages are originated from the CMAPI implementation.

5.2.2.1 CMAPI-1 Request Message

The CMAPI-1 request message conforms to the structure as defined in [JSON-RPC], extended by “cmapiversion”. The details are as follows:

Member	Type	Optional	Description
jsonrpc	String	No	It SHALL be the exact value of “**2.0**”.
cmapiversion	String	No	It SHALL be the exact value of “**1.1**”.
method	String	No	The name of the method to be invoked. It SHALL be the name of a function call as specified in [OpenCMAPI-TS], e.g. “CMAPI_Network_GetRFInfo”.
id	String	No	An identifier established by the Client, which is used to match the response with the request that it is replying to. It SHALL be the globally unique identifier to distinguish each CMAPI-1 function call originated by a Web application. Note: How such a globally unique identifier is generated is out of scope of this specification, however, it is pointed out that for example UUID [RFC4122] provides a way to implement such a scheme. The CMAPI-1 function call and corresponding response messages SHALL have the same “**id**” value Note: This specification does not use notifications as specified in [JSON-RPC], i.e. the “id” property SHALL NOT be omitted (or Null).
params	Object	No	Object, with member names that match the Server expected parameter names. It SHALL be a structure of specific parameters of a CMAPI-1 function call. The set of parameters is unique for each CMAPI-1 function call as specified in [OpenCMAPI-TS].

Table 1: CMAPI-1 Request Message Data Structure

5.2.2.1.1 JSON Request Schema Definition

The following is a general definition of the CMAPI-1 JSON request schema. A specific example of a CMAPI-1 conversion is shown in section 5.2.5.1.

The request schema contains dependent on the function requested a number of parameters 1..n each of which can contain some number of properties 1..n. The parts shown in italics and red represent variables which change depending on the CMAPI function.

```
{
  "title" : "CMAPI_FunctionName_Request_Schema",
```

```
"description" : "LEGAL DISCLAIMER  Use of this document and the content of…..",
"properties" : {
    "cmapiversion" : { "enum" : [ "1.1" ], "type" : "string" },
    "id" : { "type" : "string" },
    "jsonrpc" : {  "enum" : [ "2.0" ], "type" : "string" },
    "method" : {  "enum" : [ "CMAPI_FunctionName" ], "type" : "string" },
    "params" : {
        "properties" : {
          "param_1": {
            "type": "array",
            "description": "Some Description.",
            "items": {
              "type": "object",
              "properties": {
                "property_1": {
                  "type": "JSON_Data_Type",
                },
                "property_n": {
                  "type": "JSON_Data_Type
                }
              },
             "required": [ "property_1", "property_n " ]
            }
          },
          "param_n": {
            "type": "object",
            "properties": {
              " property_1": {
                "type": "string",
                "description": "Some Description."
              },
              "property_n": {
                "type": "integer",
                "description": "Some Description."
              }
            },
           "required": ["property_1", "property_n " ]
          }
        }          },
        "required" : ["param_1", "param_n "],
        "type" : "object"
    }
},
"required" : [ "jsonrpc", "cmapiversion", "method", "id", "params"]
}
```

5.2.2.2 CMAPI-1 Response Message

The CMAPI-1 response message conforms to the structure as defined in [JSON-RPC], extended by "cmapiversion". The details are as follows:

Member	Type	Optional	Description
jsonrpc	String	No	It SHALL be the exact value of "**2.0**".
cmapiversion	String	No	It SHALL be the exact value of "**1.1**".
id	String	No	It SHALL be the same value of "**id**" member in the prior request message of corresponding function call. Multiple sequential response messages are possible for the same prior CMAPI-1 function call. Their "**id**" SHALL be the same value. If there was an error in detecting the id in the request object (e.g. parse error/invalid request), it SHALL be Null.
error	Object	Yes	In case of an error it SHALL be the error structure according to [JSON-RPC] indicating the execution status of the corresponding CMAPI-1 function call. Please

			refer to section 5.3 for CMAPI specific error codes. On success this member SHALL be omitted. The error structure SHALL include the members “code” and “message”. For “code” the integer value of the error as listed in the table in section 5.3.1 SHALL be used. For “message” the text in the description column in the table in section 5.3.1 SHALL be used. The error structure MAY include the member “data”. It may be a used for additional information about the error.
result	Object	Yes	On success it SHALL be a structure of the information with regard to execution outcome resulting from the function call as specified in [OpenCMAPI-TS]. In case of an error this member SHALL be omitted. Note: The structure of the response is unique for each CMAPI-1 function call

Table 2: CMAPI-1 Response Message Data Structure

5.2.2.2.1 JSON Response Schema Definition

The following is a general definition of the CMAPI-1 JSOIN response schema. A specific example of a CMAPI-1 conversion is shown in section 5.2.5.2.

The response schema contains dependent on the function requested zero or a number of parameters 0..n each of which can contain some number of properties 1..n. The parts shown in italics and red represent variables which change depending on the CMAPI function.

```
{
  "title" : "CMAPI_FunctionName_Response_Schema",
  "description" : "LEGAL DISCLAIMER  Use of this document ….",
  "properties" : {
    "cmapiversion" : { "enum" : [ "1.1" ], "type" : "string" },
    "error" : {
      "properties" : {
        "code" : { "type" : "integer" },
        "data" : {},
        "message" : { "type" : "string" }
      },
      "required" : [ "code", "message" ],
      "type" : "object"
    },
    "id" : { "type" : "string" },
    "jsonrpc" : { "enum" : [ "2.0" ], "type" : "string" },
    "result" : {
            "properties" : {
              "param_1": {
                "type": "array",
                "description": "Some Description.",
                "items": {
                  "type": "object",
                  "properties": {
                    "property_1": {
                      "type": "JSON_Data_Type",
                    },
                    "property_n": {
                      "type": "JSON_Data_Type
                    }
                  },
                 "required": [ "property_1", "property_n " ]
                }
              },
              "param_n": {
                "type": "object",
                "properties": {
                  " property_1": {
```

```
                    "type": "string",
                    "description": "Some Description."
                  },
                  "property_n": {
                    "type": "integer",
                    "description": "Some Description."
                  }
                },
                "required": ["property_1", "property_n " ]
              }
            }          },
            "required" : ["param_1", "param_n "],
            "type" : "object"
          }
    },
    "required" : [ "jsonrpc", "cmapiversion", "id" ]
}
```

5.2.3 CMAPI-2 Messages

The CMAPI-2 interface is an asynchronous interface used to provide callbacks (i.e. notifications) and the registration/deregistration mechanisms to receive these callbacks.

The CMAPI-2 callback message conforms to the structure defined in [JSON-RPC], extended by "cmapiversion" and "callbackId". The details are as follows:

Member	Type	Optional	Description
jsonrpc	String	No	It SHALL be the exact value of "**2.0**".
cmapiversion	String	No	It SHALL be the exact value of "**1.1**".
id	String	No	It SHALL be a globally unique identifier to distinguish each CMAPI-2 callback function. Note: How such a globally unique identifier is generated is out of scope of this specification, however, it is pointed out that for example UUID [RFC4122] provides a way to implement such a scheme. Note: This specification does not use notifications as specified in [JSON-RPC], i.e. the "id" property SHALL NOT be omitted (or null).
error	Object	Yes	In case of an error it SHALL be the error structure according to [JSON-RPC] indicating the execution status of a CMAPI-2 callback function. Please refer to section 5.3 for CMAPI specific error codes. On success this member SHALL be omitted. The error structure SHALL include the members "code" and "message". For "code" the integer value of the error as listed in the table in section 5.3.1 SHALL be used. For "message" the text in the description column in the table in section 5.3.1 SHALL be used. The error structure MAY include the member "data". It may be a used for additional information about the error.
result	Object	Yes	On success it SHALL be a structure of the information that the application server intends to inform the Web application as specified in [OpenCMAPI-TS]. Note: The structure of callback is unique for each CMAPI-2 callback function In case of an error this member SHALL be omitted.
callbackId	String	No	It SHALL be present if and only if it is a CMAPI-2 callback message, and indicate the type of callback function as specified in [OpenCMAPI-TS].

Table 3: CMAPI-2 Callback Message Data Structure

5.2.3.1.1 JSON Callback Response Schema Definition

The following is a general definition of the CMAPI-2 JSON Callback response schema. A specific example of a CMAPI-2 conversion is shown in section 5.2.5.3.

The callback response schema contains dependent on the function requested zero or a number of parameters 0..n each of which can contain some number of properties 1..n. The parts shown in italics and red represent variables which change depending on the CMAPI callback function.

```
{
  "title" : "CMAPI_Callback_FunctionName_Response_Schema",
  "description" : "LEGAL DISCLAIMER  Use of this ….",
  "properties" : {
    "callbackId" : { "enum" : [ "CMAPI_Callback_FunctionName" ], "type" : "string" },
    "cmapiversion" : { "enum" : [ "1.1"  ], "type" : "string" },
    "error" : {
      "properties" : {
        "code" : { "type" : "integer" },
        "data" : {},
        "message" : { "type" : "string" }
      },
      "required" : [ "code", "message" ],
      "type" : "object"
    },
    "id" : { "type" : "string" },
    "jsonrpc" : { "enum" : [ "2.0" ], "type" : "string" },
    "result" : {
      "properties" : {
            "properties" : {
              "param_1": {
                 "type": "array",
                 "description": "Some Description.",
                 "items": {
                    "type": "object",
                    "properties": {
                       "property_1": {
                          "type": "JSON_Data_Type",
                       },
                       "property_n": {
                          "type": "JSON_Data_Type
                       }
                    },
                   "required": [ "property_1", "property_n " ]
                 }
              },
              "param_n": {
                 "type": "object",
                 "properties": {
                    " property_1": {
                       "type": "string",
                       "description": "Some Description."
                    },
                    "property_n": {
                       "type": "integer",
                       "description": "Some Description."
                    }
                 },
                "required": ["property_1", "property_n " ]
              }
            }            },
            "required" : ["param_1", "param_n "],
            "type" : "object"
        }
    },
    "required" : [ "jsonrpc", "cmapiversion", "id", "callbackId" ]
}
```

5.2.4 Binary Data Handling

Occasionally, binary data may be passed as a parameter in a request message, or returned in a response message after the function call is executed, or part of a callback message. BASE64 encoding SHALL be applied to binary data before it is constructed into the data structure of a request message, or a response message, or a callback message.

5.2.5 Message Examples (Informative)

5.2.5.1 CMAPI-1 Request Message Example

An example of a request message of CMAPI-1 function call “CMAPI_Network_GetRFInfo()” is as follows:

```
{
   "jsonrpc": "2.0",
   “cmapiversion”: “1.1”,
   "method": "CMAPI_Network_GetRFInfo",
   "id": "111",
   "params": {
      "deviceId": "1"
   }
}
```

5.2.5.2 CMAPI-1 Response Message Examples

An example of a successful response message of CMAPI-1 function call “CMAPI_Network_GetRFInfo()” is as follows:

```
{
   "jsonrpc": "2.0",
   “cmapiversion”: “1.1”,
   "id": "111",
   "result": {
      "RFInfoListElements": 1,
      "RFInfoList": [
         {
            "Radio": "WCDMA_UMTS",
            "maxDataRateUL": 1024,
            "maxDataRateDL": 1024,
            "frequencyBand": "1900 PCS",
            "channelNumberUL": "333,444",
            "channelNumberDL": "333,444"
         }
      ]
   }
}
```

An example of a error response message of CMAPI-1 function call “CMAPI_Network_GetRFInfo()” is as follows:

```
{
   "jsonrpc": "2.0",
   “cmapiversion”: “1.1”,
   "id": "111",
   " error": {
                "code": 1,
                "message": “A fatal error has occurred.”
              }
 }
```


5.2.5.3 CMAPI-2 Callback Message Example

An example of a CMAPI-2 Callback Message “CMAPI_Callback_DeviceChanged()” is as follows:


```
{
   "jsonrpc": "2.0",
   “cmapiversion”: “1.1”,
   "id": "511",
   "callbackId": "CMAPI_Callback_DeviceChanged",
   "result": {
      "deviceId": 1,
      "deviceState": 3,
      "radio": 64,
      "deviceCapability": 1,
      "connectionType": 32,
      "deviceType": 5,
      "description": "This is a wireless router",
      "uniqueIdentifier": "1234567890"
   }
}
```


5.3 Error Codes

This section defines the CMAPI specific error codes and UICC Status Words.

For error handling specific to the transport please refer to the respective transport binding sections 5.4.1.4 and 5.4.2.6. For error handling specific to JSON-RPC please refer to [JSON-RPC].

5.3.1 Error Codes

The error codes table is used to capture the warnings, error codes and information when the Open CMAPI is running. Some additional warnings and output information can be defined depending on the implementation.

Error Codes		
Integer Value	**Hex Value**	**Description**

General Error Codes		
1	0X00000001	A fatal error has occurred.
2	0X00000002	Invalid Parameter
4	0X00000004	Invalid Operation
5	0X00000005	No service
6	0X00000006	The requested operation cannot currently be completed because another application is currently performing the same operation.
7	0X00000007	This optional function is not supported by this implementation
16	0X00000010	The OpenCMAPI implementation cannot perform this operation since there is currently a connection which prevents the request. NOTE: The OpenCMAPI implementation may be able to apply the change in some conditions and may return success instead of this return code in some connected conditions.
17	0X00000011	The type of data requested is not present
19	0X00000013	QoS unsupported
20	0X00000014	Not connected
Device Error Codes		
256	0X00000100	The UniqueIdentifier is referencing a non-existing device
257	0X00000101	The deviceID references a non-existing device or a device which is not open
258	0X00000102	The device is already opened.
259	0X00000103	Maximum number of device that the API can handle per client is reached (can be 1), close another open device handle.
260	0X00000104	The device does not contain hardware which supports this operation.
261	0X00000105	The radio references a radio which the device does not support
262	0X00000106	The radio references a radio which the device does not support (exception, this error is not reported if the radio is set to 0xFF (all)).
263	0X00000107	System not supported by the device
264	0X00000108	The requested data is not meaningful for a 3GPP device.
265	0X00000109	The requested data is not meaningful for a 3GPP2 device.
272	0X00000110	The device cannot be activated while connected.
273	0X00000111	The device is not connected
274	0X00000112	The routerID references a non-existing router
288	0X00000120	Configuration not supported by the device
289	0X00000121	The device does not offer this capability
304	0X00000130	The device is not in a power state which allows this operation.
305	0X00000131	Requested power state is not supported by the device (ex power saving)
306	0X00000132	Radio off
307	0X00000133	The power state is invalid
308	0X00000134	The system ID is invalid
309	0X00000135	No IMSI available
320	0X00000140	The MACAddress references a non-existing Connected Device
336	0X00000150	The threshold value(s) is/are invalid

337	0X00000151	The location is invalid
352	0X00000160	The PDP context ID is invalid
353	0X00000161	The PDP Type is invalid
356	0X00000164	The back off time interval is invalid
528	0X00000210	Control Key not supported by this system (when an ID of a 3GPP2 only Control Key is sent to a 3GPP system device or when an ID of a 3GPP only Control Key is sent to a 3GPP2 system device).
529	0X00000211	The control key value is invalid
UICC Error Codes		
1281	0X00000501	There is no smart card support for this device
1282	0X00000502	Smart card not accessible
1361	0X00000551	ENVELOPE command was not sent to SIM/R-UIM/UICC as overlapping was detected.
1362	0X00000552	The envelope command is invalid
1363	0X00000553	The terminal profile is invalid
1364	0X00000554	The function succeeded except for the overlapping ToolKit functions with the device or another or other Connection Manager Application(s)
1365	0X00000555	The terminal response is invalid
Profile Error Codes		
8193	0X00002001	The Cellular profile name does not exist
8194	0X00002002	The Cellular profile name is not valid
8195	0X00002003	The Cellular profile name is already existing, only happen when creating a profile with a existing name
8196	0X00002004	The Cellular profile can not be updated while currently in use (connected)
8197	0X00002005	A default profile has not been set for this device.
8449	0X00002101	The user name is not valid
8450	0X00002102	The password is not valid
8452	0X00002104	The APN is not valid
8453	0X00002105	The IP Address is not valid
8454	0X00002106	The primary DNS address is not valid
8455	0X00002107	The secondary DNS address is not valid
8456	0X00002108	The Auth type is not valid
8457	0X00002109	The IPAddrType is not valid
8458	0X0000210A	The profile type is not valid
8459	0X0000210B	The timeout is not valid
8706	0X00002202	The type of IP address is not available.
Network Connection Error Codes		
12289	0X00003001	The requested bearer is not possible
12290	0X00003002	There is no connection to disconnect from
12292	0X00003004	There is no connecting session for cancellation
12293	0X00003005	The Connection is releasing

12294	0X00003006	Remote system not present
12295	0X00003007	The supplied index identifies a record which does not exist.
12296	0X00003008	Current APN cannot be retrieved because there is no connection.
12297	0X00003009	The requested connection type is not valid
12298	0X0000300A	There is currently a connection which prevents this operation. It is necessary to disconnect before the requested operation can be completed.
12545	0X00003101	The requested mode is not valid
12546	0X00003102	The requested PLMNID is not valid
12547	0X00003103	The requested bearer or combination of bearers is not valid.
12801	0X00003201	No Primary context activated
12802	0X00003202	The secondary context doesn't exist
12803	0X00003203	The secondary context is already activated/created
12804	0X00003204	The secondary context activation is in progress
12805	0X00003205	The secondary context is already deactivated
12806	0X00003206	The secondary context deactivation is in progress
12807	0X00003207	The secondary context is already deactivating
CDMA 2000 Error Codes		
16385	0X00004001	Unrecognized session identifier.
16386	0X00004002	The SPC is valid.
16387	0X00004003	The SPC is invalid.
16388	0X00004004	The requested activation code is invalid.
16389	0X00004005	Activation failed (other than invalid activation code).
16390	0X00004006	The index is invalid
16391	0X00004007	File does not exist at the given path.
16392	0X00004008	An invalid PRL file is entered.
16395	0X0000400B	No record exists at the specified index.
16396	0X0000400C	The ACCOLC is invalid.
16397	0X0000400D	The requested ForceRev0 is invalid
16398	0X0000400E	The CustomSCP is invalid
16399	0X0000400F	The protocol is invalid
16400	0X00004010	The broadcast is invalid
16401	0X00004011	The application is invalid
16402	0X00004012	The roaming is invalid
16403	0X00004013	The SID is invalid
16404	0X00004014	The MDN is invalid
16405	0X00004015	The MIN is invalid
16406	0X00004016	The PRL is invalid
16407	0X00004017	The MNHA is invalid
16408	0X00004018	The MNAAA is invalid

16409	0X00004019	The session type is invalid
16410	0X0000401A	The session state is invalid
16411	0X0000401B	The failure reason is invalid
16412	0X0000401C	The retry count is invalid
16413	0X0000401D	The session pause is invalid
16414	0X0000401E	The selection is invalid
16415	0X0000401F	The session id is invalid
16416	0X00004020	The defer is invalid
16417	0X00004021	The feature state is invalid
16418	0X00004022	The update feature state is invalid.
16419	0X00004023	The firmware update feature state is invalid
16420	0X00004024	The reason is invalid
16421	0X00004025	The mode is invalid
16422	0X00004026	The enabled value is invalid
16423	0X00004027	The RevTunn value is invalid
16424	0X00004028	The NAI is invalid
16425	0X00004029	The HASPI is invalid
16426	0X0000402A	The AAASPI is invalid
16427	0X0000402B	The Address parameter was not formatted properly.
16428	0X0000402C	The Primary Home Agent parameter was not formatted properly.
16429	0X0000402D	The Secondary Home Agent parameter was not formatted properly.
16430	0X0000402E	The retry limit is invalid
16431	0X0000402F	The retry interval is invalid
16432	0X00004030	The Reregperiod is invalid
16433	0X00004031	The Reregtraffic is invalid
16434	0X00004032	The HAAuthenticator is invalid
16435	0X00004033	The HA2002bis is invalid
SMS Error Codes		
20481	0X00005001	Failure of communication with device
20482	0X00005002	Timer expired without receiving response from device
20483	0X00005003	Response with error indication from device
20484	0X00005004	Operation NOT supported
20485	0X00005005	SMS message NOT found
20486	0X00005006	The SMS record is invalid
20487	0X00005007	The ifrom value is invalid
20488	0X00005008	The SMSC value is invalid
20489	0X00005009	The PSI value is invalid
20490	0X0000500A	The delivery report switch is invalid
20491	0X0000500B	The SMS Class is invalid

20492	0X0000500C	The msgID is invalid
22785	0X00005901	The USSD Data is invalid
Contact Management Error Codes		
21761	0X00005501	The contact record is invalid
21762	0X00005502	Memory capacity exceeded.
21763	0X00005503	The index is invalid
21764	0X00005504	The contact location value is invalid
Information Status Error Codes		
24577	0X00006001	The type of data requested is not present
24578	0X00006002	The type is not valid
24579	0X00006003	Remote system not present
24580	0X00006004	The supplied index identifies a record which does not exist.
24581	0X00006005	Current APN cannot be retrieved because there is no connection.
24582	0X00006006	The type of IP address is not available.
24583	0X00006007	IP Address is not currently assigned (advisable to retry call)
24584	0X00006008	Authentication failure
GNSS Error Codes		
28673	0X00007001	The GNSS state is invalid
28674	0X00007002	The operation is invalid
28675	0X00007003	The accuracy threshold is not supported
28676	0X00007004	The server address is invalid.
28677	0X00007005	The server port is invalid.
28678	0X00007006	The server FQDN is invalid.
28679	0X00007007	The tracking value is invalid
P2P Direct Management Error Codes		
32769	0X00008001	The P2PTechnology is not supported
32770	0X00008002	The P2P Technology is invalid
32771	0X00008003	The Service Record is invalid
32772	0X00008004	The list of Remote Devices is invalid.
32773	0X00008005	The list of Service Identifiers is invalid.
32774	0X00008006	The ID of the Connection is invalid.
32775	0X00008007	The list of Device ID is invalid
32776	0X00008008	The ID of the group is invalid
32777	0X00008009	The ID of the Remote Device is invalid
32778	0X0000800A	The Invitation ID is invalid
Router Management Error Codes		
36865	0X00009001	The routerConfig value(s) are incorrect
36866	0X00009002	The policy value(s) are incorrect
36867	0X00009003	The restrict value(s) are incorrect

36868	0X00009004	The administrator password is incorrect
WLAN Error Codes		
65537	0X00010001	No network exists at the specified index.
65538	0X00010002	Predefined networks are not able to be modified.
65540	0X00010004	The SSID is invalid
65541	0X00010005	The BSSID is invalid
65542	0X00010006	The Friendly Name is invalid
65543	0X00010007	The security parameter is invalid
65544	0X00010008	The mode parameter is invalid
65545	0X00010009	The hidden parameter is invalid
65546	0X0001000A	The key is invalid
65547	0X0001000B	The EAP authentication method is invalid
65548	0X0001000C	The EAP configuration is invalid
65549	0X0001000D	The WLAN Encryption Type is invalid
69633	0X00011001	There is no existing WLAN connection
69634	0X00011002	Security mode does not allow connectivity to unknown networks.
69637	0X00011005	Operation is prohibited by security policy.
69638	0X00011006	No pending operation.
69639	0X00011007	The pin for WPS was malformed or incorrect size
69640	0X00011008	The device is not connected
69641	0X00011009	Device (i.e.: WLAN only device that does not support NAA on UICC for authentication) does not support the requested function.
73729	0X00012001	The SSID does not reference a valid known network.
73730	0X00012002	The BSSID does not reference a valid known network
73731	0X00012003	IP Address is not currently assigned (advisable to retry call)
73732	0X00012004	Authentication failure
77825	0X00013001	Invalid combination of AUTH and CIPHER
77826	0X00013002	Index NOT referring to a valid known network
77827	0X00013003	NO existing WLAN connection
77828	0X00013004	IP address NOT valid
77829	0X00013005	Subnet mask NOT valid
77830	0X00013006	Operation prohibited by security policy
77831	0X00013007	The specified index is to large and would leave a gap in the known networks list
77832	0X00013008	Index is not valid for user defined networks. Please try a higher index.
77833	0X00013009	The mode is invalid
77834	0X0001300A	The address is invalid
77835	0X0001300B	The subnet mask is invalid
77836	0X0001300C	The http proxy is invalid
77837	0X0001300D	The mac address is invalid

77838	0X0001300E	The default gateway is invalid
81921	0X00014001	The Advertisement Protocol Element is invalid
81922	0X00014002	The Query List ANQP element is invalid
81923	0X00014003	The HS Query List is invalid
81953	0X00014021	HS 2.0 MO is not supported by the device
81954	0X00014022	ANDSF MO is not supported by the device
PIN/PUK management Error Codes		
		SW1 and SW2 are the Status Words provided by the SIM/R-UIM/UICC (see next chapter). If no Status Word is provided, SW1SW2 will be replaced by “0000”.
2684SW1SW2	0X1001SW1SW2	Wrong PIN.
2685SW1SW2	0X1002SW1SW2	PIN is blocked. PUK (UNBLOCK PIN) needed.
2686SW1SW2	0X1003SW1SW2	Wrong Old PIN.
2687SW1SW2	0X1004SW1SW2	Old PIN is blocked. PUK (UNBLOCK PIN) needed.
2688SW1SW2	0X1005SW1SW2	Wrong PUK.
2689SW1SW2	0X1006SW1SW2	PUK (UNBLOCK PIN) blocked.
2690SW1SW2	0X1007SW1SW2	Invalid parameter(s)
285212673	0X11000001	The NAA Name is invalid
285212674	0X11000002	The PIN Type is invalid
285212675	0X11000003	The PUK Type is invalid
Reserved for other use		
805306368 to 1073741823	0X30000000 to 0X3FFFFFFF	Reserved for other purpose – do not use
M2M/IoT related Error Codes		
107381XXXX	0X4001XXXX	Operation cannot be done – Back off timer in place – time left is indicated by the 4 last digits (in seconds) XXXX is the time left in seconds – example: 0X40010360 in Hex or 1073810630 in decimal mean 360 seconds are left
10748MMCME	0X401MMCME	Error codes related to GSM Mobility Management where MM indicates the Mobility Management Cause code and CME the code for Mobile Equipment error.
10759GMCME	0X402GMCME	Error codes related to GPRS Mobility Management where GM indicates the GPRS Mobility Management Cause code and CME the code for Mobile Equipment error.
10769SMCME	0X403SMCME	Error codes related to Session Management where SM indicates the Session Management Cause code and CME the code for Mobile Equipment error.
10780XXCMS	0X404XXCMS	Error codes related to other reasons where XX indicates other Cause code and CMS the code for Mobile Equipment specific error.
Security Errors		
4026531841	0XF0000001	The security request supplied when the API was opened does not grant privilege to access this functionality. You may close and reopen the API with updated credentials to perform this operation.
4026531842	0XF0000002	The authentication failed
4026531843	0XF0000003	The authentication has been denied. Please seek proper credentials for your access level.

4026531844	0XF0000004	The security request was malformed. Please consult vendor materials and/or output log.
4026531845	0XF0000005	The requested access level is not supported
4026531846	0XF0000006	The WLAN Encryption Type used is not allowed. Please use proper Encryption type

Table 4: Return Values & Error Codes

5.3.2 UICC Status Words

The following table is listing possible Status Words (SW1 and SW2) provided by the SIM/R-UIM/UICC in accordance with the [ETSI TS 102 221] Status Words list.

Status Words		
	Status words (SW1 SW2)	**Description**
144 00	90 00	Normal ending of the command
145 00	91 XX	Normal ending of the command, with extra information from the proactive UICC containing a command for the terminal. Length 'XX' of the response data
098 00	62 00	No information given, state of non volatile memory unchanged
099 CX	63 CX	Command successful but after using an internal update retry routine 'X' times Verification failed, 'X' retries remaining (For the VERIFY PIN command, SW1SW2 indicates that the command was successful but the PIN was not correct and there are 'X' retries left. For all other commands it indicates the number of internal retries performed by the card to complete the command.)
100 00	64 00	No information given, state of non-volatile memory unchanged
101 00	65 00	No information given, state of non-volatile memory changed
101 81	65 81	Memory problem
103 XX	67 XX	The interpretation of this status word is command dependent, except for SW2 = '00' (Wrong length)
104 00	68 00	No information given
104 81	68 81	Logical channel not supported
104 82	68 82	Secure messaging not supported
105 00	69 00	No information given
105 83	69 83	Authentication/PIN method blocked
105 84	69 84	Referenced data invalidated
105 89	69 89	Command not allowed - secure channel - security not satisfied
106 81	6A 81	Function not supported
106 86	6A 86	Incorrect parameters P1 to P2
106 88	6A 88	Referenced data not found
107 00	6B 00	Wrong parameter(s) P1-P2
110 00	6E 00	Class not supported
111 XX	6F XX	The interpretation of this status word is command dependent, except for SW2 = '00' (Technical problem, no precise diagnosis)

Table 5: Status Words Codes

5.3.3 CMEE codes

The following tables are listing possible GSM Mobile Equipment error codes and GSM network error codes.

Causes Codes Related to GSM Mobility Management				
MM Code	**CME Code**	**Cause**	**Reason**	**Action proposed**
2		IMSI unknown in HLR	This cause is sent to the MS if the MS is not known (registered) in the HLR. This cause code does not affect operation of the GPRS service, although it may be used by a GMM procedure.	
3	103	Illegal MS	This cause is sent to the MS when the network refuses service to the MS either because an identity of the MS is not acceptable to the network or because the MS does not pass the authentication check, i.e. the SRES received from the MS is different from that generated by the network.	
4		IMSI unknown in VLR	This cause is sent to the MS when the given IMSI is not known at the VLR.	
5		IMEI not accepted	This cause is sent to the MS if the network does not accept emergency call establishment using an IMEI.	
6	106	Illegal ME	This cause is sent to the MS if the ME used is not acceptable to the network, e.g. blacklisted.	
11	111	PLMN not allowed	This cause is sent to the MS if it requests location updating in a PLMN where the MS, by subscription or due to operator determined barring is not allowed to operate.	
12	112	Location Area not allowed	This cause is sent to the MS if it requests location updating in a location area where the MS, by subscription, is not allowed to operate.	
13	113	Roaming not allowed in this location area	This cause is sent to an MS which requests location updating in a location area of a PLMN which restricts roaming to that MS in that Location Area, by subscription.	
17	615	Network failure	This cause is sent to the MS if the MSC cannot service an MS generated request because of PLMN failures, e.g. problems in MAP.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
22	42	Congestion	This cause is sent if the service request cannot be processed because of congestion (e.g. no channel, facility busy/congested etc.)	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
32	132	Service Option Not	This cause is sent when the MS requests a	Additionally, device should not

		Supported.	service/facility in the CM SERVICE REQUEST message which is not supported by the PLMN.	retry the attempt on the same PLMN unless prompted externally to do so (i.e. modem should not automatically retry).
33	133	Requested Service Option Not Subscribed	This cause is sent when the MS requests a service option for which it has no subscription.	Additionally, device should not retry the attempt unless prompted externally to do so (i.e. modem should not automatically retry).
34	134	Service option temporarily out of order	This cause is sent when the MSC cannot service the request because of temporary outage of one or more functions required for supporting the service.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
38		Call Cannot be identified	This cause is sent when the network cannot identify the call associated with a call re-establishment request.	
Causes Codes Related to GPRS Mobility Management				
GM Code	**CME Code**	**Cause**	**Reason**	**Action proposed**
7	107	GPRS Services Not Allowed	This cause is sent to the MS if it requests an IMSI attach for GPRS services, but is not allowed to operate GPRS services.	
8		GPRS services and non-GPRS services not allowed	This cause is sent to the MS if it requests a combined IMSI attach for GPRS and non-GPRS services, but is not allowed to operate either of them.	
9		MS identity cannot be derived by the network	This cause is sent to the MS when the network cannot derive the MS's identity from the P-TMSI in case of inter-SGSN routing area update.	
10		Implicitly detached	This cause is sent to the MS either if the network has implicitly detached the MS, e.g. some while after the Mobile reachable timer has expired, or if the GMM context data related to the subscription does not exist in the SGSN e.g. because of a SGSN restart.	
14	111	GPRS services not allowed in this PLMN	This cause is sent to the MS which requests GPRS service in a PLMN which does not offer roaming for GPRS services to that MS.	
16		MSC temporarily not reachable	This cause is sent to the MS if it requests a combined GPRS attach or routing are updating in a PLMN where the MSC is temporarily not reachable via the GPRS part of the GSM network.	
	148	unspecified GPRS error		Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
Causes Codes Related to Session Management				

SM Code	CME Code	Cause	Reason	Action proposed
25		LLC or SNDCP failure	This cause code is used by the MS indicate that a PDP Context is deactivated because of a LLC or SNDCP failure (e.g. if the SM receives a SNSM-STATUS.request message with cause "DM received " or " invalid XID response)	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
26		Insufficient resources	This cause code is used by the MS or by the network to indicate that a PDP Context activation request or PDP Context modification request cannot be accepted due to insufficient resources.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
27	134	Unknown or missing access point name	This cause code is used by the network to indicate that the requested service was rejected by the external packet data network because the access point name was not included although required or if the access point name could not be resolved.	Additionally, do not retry with same APN unless device is power cycled.
28		Unknown PDP address or PDP type	This cause code is used by the network to indicate that the requested service was rejected by the external packet data network because the PDP address or type could not be recognised.	Additionally, do not retry with same PDP address and/or type unless device is power cycled.
29	149	User authentication failed	This cause code is used by the network to indicate that the requested service was rejected by the external packet data network due to a failed user authentication (e.g. rejected by Radius)	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
30		Activation rejected by GGSN	This cause code is used by the network to indicate that the requested service was rejected by the GGSN.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
31		Activation rejected, unspecified	This cause code is used by the network to indicate that the requested service was rejected due to unspecified reasons.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
32	132	Service option not supported	This cause code is used by the network when the MS requests a service which is not supported by the PLMN.	Additionally, device should not retry the attempt on the same PLMN unless prompted externally to do so (i.e. modem should not automatically retry).
33	133	Requested service option not subscribed	This cause is sent when the MS requests a service option for which it has no subscription.	Additionally, device should not retry the attempt on the same PLMN unless prompted externally to do so (i.e. modem should not automatically retry).
34	134	Service option temporarily out of order	This cause is sent when the MSC cannot service the request because of temporary outage of one or more functions required for supporting the service.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
35		NSAPI already used	This cause code is used by the network to	Device may choose to use a

			indicate that the NSAPI requested by the MS in the PDP Context activation is already used by another active PDP Context of this MS.	different NSAPI, or retry after the context using the required NSAPI has been deactivated.
36		Regular PDP Context deactivation	This cause code is used to indicate a regular MS or network initiated PDP Context deactivation.	
37		QoS not accepted	This cause code is used by the MS if the new QoS cannot be accepted that were indicated by the network in the PDP Context Modification procedure.	N/A
38	615	Network Failure	This cause code is used by the network to indicate that the PDP Context deactivation is caused by an error situation in the network.	Additionally, retry retries may be attempted, but no more frequently than once every 60 seconds
39		Reactivation requested	This cause code is used by the network to request a PDP Context reactivation after a GGSN restart.	Additionally, the device may re-establish the PDP Context.
40		Feature not supported	This cause code is used by the MS to indicate that the PDP Context activation initiated by the network is not supported by the MS.	N/A
Causes Codes Related to other reasons				
XX Code	**CMS Code**	**Cause**	**Reason**	**Action proposed**
	8	Operator determined barring	This cause indicates that the device has tried to send a mobile originating short message when the device's network operator or service provider has forbidden such transactions.	SMS back-off, blocking immediately any new SMS TX request sent
	10	Call barred	This cause indicates that the outgoing call barred service applies to the short message service for the called destination.	SMS back-off, blocking immediately any new SMS TX request sent
	21	Short message transfer rejected	This cause indicates that the equipment sending this cause does not wish to accept this short message, although it could have accepted the short message since the equipment sending this cause is neither busy nor incompatible.	SMS back-off, blocking immediately any new SMS TX request sent
	27	Destination out of service	This cause indicates that the destination indicated by the Device cannot be reached because the interface to the destination is not functioning correctly. The term "not functioning correctly" indicates that a signalling message was unable to be delivered to the remote user; e.g., a physical layer or data link layer failure at the remote user, user equipment off-line, etc.	SMS back-off, blocking immediately any new SMS TX request sent
	28	Unidentified subscriber	This cause indicates that the subscriber is not registered in the PLMN (i.e. IMSI not known).	SMS back-off, blocking immediately any new SMS TX request sent

	29	Facility rejected	This cause indicates that the facility requested by the Device is not supported by the PLMN.	SMS back-off, blocking immediately any new SMS TX request sent
	30	Unknown subscriber	This cause indicates that the subscriber is not registered in the HLR (i.e. IMSI or directory number is not allocated to a subscriber).	SMS back-off, blocking immediately any new SMS TX request sent
	38	Network out of order	This cause indicates that the network is not functioning correctly and that the condition is likely to last a relatively long period of time; e.g., immediately reattempting the short message transfer is not likely to be successful.	SMS back-off, blocking immediately any new SMS TX request sent
	41	Temporary failure	This cause indicates that the network is not functioning correctly and that the condition is not likely to last a long period of time; e.g., the Device may wish to try another short message transfer attempt almost immediately.	SMS back-off, blocking immediately any new SMS TX request sent
	42	Congestion	This cause indicates that the short message service cannot be serviced because of high traffic.	SMS back-off, blocking immediately any new SMS TX request sent
	47	Resources unavailable, unspecified	This cause is used to report a resource unavailable event only when no other cause applies.	SMS back-off, blocking immediately any new SMS TX request sent
	50	Requested facility not subscribed	This cause indicates that the requested short message service could not be provided by the network because the user has not completed the necessary administrative arrangements with its supporting networks.	SMS back-off, blocking immediately any new SMS TX request sent
	69	Requested facility not implemented	This cause indicates that the network is unable to provide the requested short message service.	SMS back-off, blocking immediately any new SMS TX request sent
	81	Invalid short message transfer reference value	This cause indicates that the equipment sending this cause has received a message with a short message reference which is not currently in use on the MS-network interface.	SMS back-off, blocking immediately any new SMS TX request sent
	148	Unspecified GPRS error		
17		Network failure	This cause is sent to the MS if the MSC cannot service an MS generated request because of PLMN failures, e.g. Problems in MAP.	SMS back-off, blocking immediately any new SMS TX request sent
21		Congestion	This cause is sent if the service request cannot be processed because of congestion (e.g. no channel, facility busy/congested, etc).	SMS back-off, blocking immediately any new SMS TX request sent

Table 6: CMEE Codes

5.4 WebAPI Transport Bindings

The specification defines two transport bindings for the CMAPI messages, first based on the WebSocket Protocol and second is based on HTTP.

5.4.1 WebSocket Transport Binding

This section introduces a transport binding for the CMAPI messages defined in section 5.2 using the W3C WebSocket API [W3C_WebSocket] and the underlying Web Socket Protocol [RFC6455] for both request/response and callbacks.

WebSocket provides Web applications with a full-duplex communication channel over a persistent connection. It enables a stream of messages, which is a perfect fit for the message exchange of OpenCMAPI..

5.4.1.1 Design Principle

The nature of modern Web applications is asynchrony. The "WebSocket" interface designed in W3C Web Socket API enables the asynchrony of a Web application over a full-duplex communication channel. Once a "WebSocket" connection object is established with the application server:

- sending a message in a Web application: message from a Web application can be sent to the application server using the "send(data)" method, which is non-blocking and immediately returns to the Web application
- receiving a message in a Web application: a Web application can use a "EventHandler onmessage" event handler to receive and handle messages from the application server.

5.4.1.2 CMAPI-1 Transport Binding

CMAPI-1 defines normal function calls, which is normally synchronous in native API. The native application makes a function call and waits until the function finishes the communication with the application server and returns the result.

However, in WebSocket API Binding, because of the asynchrony nature of WebSocket interface and the asynchronous way for a Web application to handle sending a message and receiving a message, CMAPI-1 functions are all modelled as asynchronous function calls. It means that all function calls are effectively the same as "_Async()" calls in semantics. The binding details are as follows:

- All CMAPI-1 function signatures are defined in WebIDL in [OpenCMAPI-SUP-WIDL];
    - An extra parameter "ResultCallback cb" is added to every function signature so that the Web application can specify a callback function "cb" to receive and handle the response message of the function call from the application server
    - "ResultCallback" interface is defined in WebIDL for the callback function "cb" of an asynchronous function call to receive and handle the response message formatted as a JSON-RPC data object "CmapiResponse" defined in section 5.2.2.2 and WebIDL as well.
- When a CMAPI-1 function call is invoked by a Web application, the JavaScript Library that implements the WebSocket API Binding follows the steps in the table below.

Step 1	Assign a globally unique transaction "id" for this CMAPI-1 function call (see section 5.2.2.1)
Step 2	Construct the JSON-RPC request object, whose format is defined in section 5.2.2.1, based on the transaction "id", the method and parameters of this CMAPI-1 function call.
Step 3	Set up the transaction "id" and callback function "cb" with the event handler of "onmessage" of the "WebSocket" object so that the corresponding "CmapiResponse" data object can be routed to this callback function "cb" according to matching the transaction "id" appropriately (see Section 5.2.2)
Step 4	Send the request message of this CMAPI-1 function call to the application server using "send(data)" method of the "WebSocket" object
Step 5	Immediately return to the Web application without blocking on waiting for the response message from the application server, which will be received and handled asynchronously.

Table 7: Steps of Handling a CMAPI-1 Function Call

- When a response message from the application server is received by the "WebSocket" object of the JavaScript Library that implements the WebSocket API Binding, the event handler of "onmessage" of the "WebSocket" object is invoked to:

Step 1	Construct the "CmapiResponse" JSON-RPC object according to the response message (see section 5.2.2.2).
Step 2	Match the transaction "id" of "CmapiResponse" with the list of transaction "id"s of prior request messages.
Step 3	If there is a match of transaction "id", invoke the corresponding callback function "cb" and pass "CmapiResponse" as its parameter.
Step 4	If there is no match, handle it in the way defined in section 5.4.1.4.

Table 8: Steps of Handling a CMPI-1 Response Message

- It should be noted that there may be more than one response messages of a CMAPI-1 function call sequentially sent from the application server. Those response messages are in sequence, and may indicate different stages of serving the CMAPI-1 function call in the application server. For example, the stages of a function call request may include "received", "processing", "completed" etc in the application server, Those multiple response messages SHALL have the same transaction "id" as that of the original CMAPI-1 function call.

5.4.1.3 CMAPI-2 Transport Binding

CMAPI-2 defines callback functions, which are sent to the client device in the same way as delivering response messages of CMAPI-1 function calls through the "WebSocket" object. In addition, there are two application-initiated function calls to register and unregister callback functions:

- When a Web application registers a callback function, the JavaScript Library that implements the WebSocket API Binding handles this function call in the same way as that of CMAPI-1 function calls. In addition, there is one more step to follow:

Step 1	The JavaScript Library sets up the "callbackId" and the callback function "cb" with the event handler of "onmessage" of the "WebSocket" object so that the corresponding "CmapiResponse" data object can be routed to this callback function "cb" according to matching the "callbackId" appropriately (see Section 5.2.3).

Table 9: Extra Step of Handling a Callback Registration

- When a Web application unregisters a callback function, the JavaScript Library that implements the WebSocket API Binding SHALL handle this function call in the same way as that of CMAPI-1 function calls. In addition, there is one more step to follow:

Step 1	JavaScript library SHALL remove the prior setup of the "callbackId" and the callback function "cb" with the event handler of "onmessage" of the "WebSocket" object.

Table 10: Extra Step of Handling a Callback Unregistration

When a "WebSocket" object of the JavaScript Library that implements the WebSocket API Binding receives a message from the application server, the event handler of "onmessage" of the "WebSocket" object is invoked to:

Step 1	Construct the "CmapiResponse" JSON-RPC object according to the message (see Section 5.2.3).
Step 2	Match the transaction "id" of "CmapiResponse" with the list of transaction "id"s of prior CMAPI-1 request messages.
Step 3	If there is a match of transaction "id", handle it in the way defined in Section 5.4.1.2.
Step 4	If there is no match:

	- If there is a "callbackId" member in "CmapiResponse" with a valid value, match it with the list of registered "callbackIds". o If the "callbackId' is in the list of registered "callbackIds", invoke the corresponding callback function "cb" and pass "CmapiResponse" as its parameter. o If the "callbackId' is not in the list of registered "callbackIds", call general error handling functions defined in Section 5.4.1.4. - If there is not a "callbackId" member in "CmapiResponse", or if the "callbackId" member is empty or invalid value, call general error handling functions defined in Section 5.4.1.4.
Step 5	It should be noted that the same type of callback function may be initiated and sent from the server more than once for the changed situation of the same characteristics. Those multiple messages of the same "callbackId" SHALL NOT have the same transaction "id" in order to distinguish those changes.

Table 11: Steps of Handling a Callback

5.4.1.4 WebSocket Transport Error Handling

The error handling mechanism SHALL be able to handle those generic errors defined in [JSON-RPC] and CMAPI-specific errors defined in section 5.3

In addition, the following general error conditions will be handled according to operators' policy.

- In the message from application server, the transaction "id" doesn't match any transaction "id" of prior CMAPI-1 request messages, and the "callbackId" is either absent or empty or invalid value.
- In the message from application server, the "callbackId" is not in the list of registered CMAPI-2 "callbackIds".

5.4.2 HTTP Transport Binding

This section introduces the transport binding for the CMAPI messages defined in section 5.2 using HTTP for synchronous request /response and HTTP Long Polling used for callbacks.

5.4.2.1 General

CMAPI SHALL support HTTP1.1 [RFC2616] for CMAPI-1 and CMAPI-2 interfaces.

5.4.2.2 Content Type

CMAPI SHALL support messages formatted as entity-bodies with the following content type:

- application/json media type. The application/ json media type is used when a single CMAPI-1 or CMAPI-2 interface message is included in the HTTP request/response.

5.4.2.3 HTTP Method

CMAPI SHALL send all request messages on CMAPI-1 and CMAPI-2 interface as HTTP POST method requests.

5.4.2.4 CMAPI-1 HTTP Transport Binding

CMAPI-1 communication between Web applications and a CMAPI is carried out using HTTP POST requests and HTTP responses, with the JSON objects (as specified in section 5.2.2) as data.

5.4.2.5 CMAPI-2 HTTP Transport Binding

The method for a Web application to receive asynchronous notifications via CMAPI-2 interface about the callbacks the Web application has registered to is based on HTTP requests and often referred to as "HTTP Long Polling" [RFC6202].

When a callback fires a notification is sent to the Web application, i.e. a CMAPI-2 message included in the HTTP message body within the HTTP response to the pending HTTP Long Polling request.

5.4.2.6 HTTP Transport Error Handling

The error handling mechanism SHALL be able to handle those generic errors defined in [JSON-RPC] and CMAPI-specific errors defined in section 5.3.

When there is no CMAPI message to send in response to an request, CMAPI SHALL send a 204 No Content response. Other allowed status codes, reflecting the outcome of the HTTP POST request, are defined in [RFC2616].

5.5 Security Considerations

Management of connections is a sensitive operation which can involve secrets and confidential data (e. g. password), so it is required to perform CMAPI operations in a securely mutually authenticated , confidential and integrity protected context. This CMAPI release does leave the security mechanisms required up to implementation.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
n/a	n/a	No prior version

A.2 Draft/Candidate Version 1.1 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-TS-OpenCMAPI_Web_V1_1	04 Jun 2013	All	First baseline document
	04 Sep 2013	All	Incorporated the following CRs: OMA-CD-OpenCMAPI-2013-0074R02-CR_TS_WebAPI
	01 Jan 2014	All	Incorporated the following CRs: OMA-CD-OpenCMAPI-2013-0085-CR_JSON_API_Management OMA-CD-OpenCMAPI-2013-0103-CR_JSON_Device_Discovery OMA-CD-OpenCMAPI-2013-0104-CR_JSON_Cellular_Network_Management OMA-CD-OpenCMAPI-2013-0105-CR_JSON_Connection_Management OMA-CD- OMA-CD-OpenCMAPI-2013-0106-CR_JSON_Network_Management OMA-CD-OpenCMAPI-2013-0107-CR_JSON_CDMA2000 OMA-CD-OpenCMAPI-2013-0108-CR_JSON_Device_Service OMA-CD-OpenCMAPI-2013-0109-CR_JSON__Device_Extended_Service OMA-CD-OpenCMAPI-2013-0110-CR_JSON_PIN_PUK OMA-CD-OpenCMAPI-2013-0111-CR_JSON_UICC OMA-CD-OpenCMAPI-2013-0113-CR_JSON_Statistics OMA-CD-OpenCMAPI-2013-0114-CR_JSON_Information_Status OMA-CD-OpenCMAPI-2013-0115-CR_JSON_SMS OMA-CD-OpenCMAPI-2013-0116-CR_JSON_USSD OMA-CD-OpenCMAPI-2013-0117-CR_JSON_GNSS OMA-CD-OpenCMAPI-2013-0118-CR_JSON_Data_Push_Service OMA-CD-OpenCMAPI-2013-0119-CR_JSON_Contact_Management OMA-CD-OpenCMAPI-2013-0120-CR_JSON_P2P OMA-CD-OpenCMAPI-2013-0121-CR_JSON_Router_Management OMA-CD-OpenCMAPI-2013-0162-CR_CR_JSON_Callback OMA-CD-OpenCMAPI-2013-0163-CR_CMAPI_WebBinding_Reference OMA-CD-OpenCMAPI-2013-0166R01-CR_CMAPI_WebBinding_Introduction OMA-CD-OpenCMAPI-2013-0167-CR_WebBinding_WebSocket OMA-CD-OpenCMAPI-2013-0168R01-CR_WebBinding_JSONRPC OMA-CD-OpenCMAPI-2013-0169R01-CR_WebBinding_Appendix_D OMA-CD-OpenCMAPI-2013-0171-CR_JSON_CB_Registration
	30 Jan 2014	B	Incorporated CR: OMA-CD-OpenCMAPI-2014-0010-CR_WebTS_SCR Editorial changes
	31 Jan 2014	All	Editorial changes including changes in accordance with actions: OpenCMAPI-2014-A001 OpenCMAPI-2014-A002 OpenCMAPI-2014-A003 OpenCMAPI-2014-A005
	18 Feb 2014	All	Changes according to CONRR comments resolution in OMA-CONRR-OpenCMAPI-V1_1-20140221-D
	01 Apr 2014	All	Incorporated: OMA-CD-OpenCMAPI-2014-0017-CR_WebTS_errorcodes OMA-CD-OpenCMAPI-2014-0020-CR_WebTS_move_JSD_to_SUP OMA-CD-OpenCMAPI-2014-0021-CR_WebTS_References_Section
	22 Apr 2014	All	Incorporated: OMA-CD-OpenCMAPI-2014-0027R01-CR_Resolution_for_some_CONR_comments_to_WebTS

Document Identifier	Date	Sections	Description
	5 May 2014	5.2, 5.3,	Incorporated: OMA-CD-OpenCMAPI-2014-0032R01-CR_Next_WebTS_CONR_comments_resolutions
	29 Sep 2014	All	Incorporated: OMA-CD-OpenCMAPI-2014-0077-CR_Resolution_re_WebTS_Security OMA-CD-OpenCMAPI-2014-0092-CR_WebTS_Editorial_Update
	24 Nov 2014	5.2	Incorporated: OMA-CD-OpenCMAPI-2014-0099-CR_WebTS_Schema_description
Candidate Version OMA-TS-OpenCMAPI_Web_V1_1	17 Feb 2015	n/a	Status changed to Candidate by TP TP Ref # OMA-TP-2015-0059-INP_OpenCMAPI_V1_1_ERP_and_ETR_for_Candidate_approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

Every API function calls need to be supported by the implementation of the OpenCMAPI. It shall at least support the call of the function and the dedicated generic return value.

But if one the functions is listed as mandatory in one of the following tables the full feature needs to be implemented in the API for the targeted device type.

And if one the functions is listed as Optional in one of the following tables, when implemented then the full feature needs to be implemented in the API for the targeted device type.

B.1 SCR for Mobile Broadband Device

Item	Function	Reference	Requirement
OpenCMAPI-MBD-001-M	Support API Management	7.2	
OpenCMAPI-MBD-002-M	Support Device Discovery APIs	7.3	
OpenCMAPI-MBD-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-MBD-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-MBD-005-M	Support Network Management APIs	7.6	
OpenCMAPI-MBD-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-MBD-007-M	Support Device Service APIs	7.8	
OpenCMAPI-MBD-008-M	Support PINs/PUKs Management APIs	7.10	
OpenCMAPI-MBD-009-O	Support UICC Management APIs	7.11	
OpenCMAPI-MBD-010-O	Support WLAN APIs	7.12	
OpenCMAPI-MBD-011-M	Support Statistics APIs	7.13	
OpenCMAPI-MBD-012-M	Support Information Status APIs	7.14	
OpenCMAPI-MBD-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-MBD-014-M	Support USSD Management APIs	7.16	
OpenCMAPI-MBD-015-O	Support GNSS APIs	7.17	
OpenCMAPI-MBD-016-O	Support Data Push Service Management APIs	7.18	
OpenCMAPI-MBD-017-M	Support Callback APIs	8	
OpenCMAPI-MBD-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-MBD-019-M	Support Contact Management APIs	7.19	
OpenCMAPI-MBD-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-MBD-021-O	Support Wireless Router APIs	7.21	
OpenCMAPI-MBD-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-MBD-023-O	Support M2M/IoT APIs	7.23	

B.2 SCR for laptop

Item	Function	Reference	Requirement
OpenCMAPI-LAP-001-M	Support API Management	7.2	
OpenCMAPI-LAP-002-M	Support Device Discovery APIs	7.3	
OpenCMAPI-LAP-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-LAP-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-LAP-005-M	Support Network Management APIs	7.6	
OpenCMAPI-LAP-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-LAP-007-M	Support Device Service APIs	7.8	
OpenCMAPI-LAP-008-M	Support PINs/PUKs Management APIs	7.10	
OpenCMAPI-LAP-009-O	Support UICC Management APIs	7.11	

Item	Function	Reference	Requirement
OpenCMAPI-LAP-010-M	Support WLAN APIs	7.12	
OpenCMAPI-LAP-011-M	Support Statistics APIs	7.13	
OpenCMAPI-LAP-012-M	Support Information Status APIs	7.14	
OpenCMAPI-LAP-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-LAP-014-M	Support USSD Management APIs	7.16	
OpenCMAPI-LAP-015-O	Support GNSS APIs	7.17	
OpenCMAPI-LAP-016-O	Support Data Push Service Management APIs	7.18	
OpenCMAPI-LAP-017-M	Support Callback APIs	8	
OpenCMAPI-LAP-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-LAP-019-M	Support Contact Management APIs	7.19	
OpenCMAPI-LAP-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-LAP-021-O	Support Wireless Router APIs	7.21	
OpenCMAPI-LAP-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-LAP-023-O	Support M2M/IoT APIs	7.23	

B.3 SCR for wireless router

Item	Function	Reference	Requirement
OpenCMAPI-WIR-001-M	Support API Management	7.2	
OpenCMAPI-WIR-002-M	Support Device Discovery APIs	7.3	
OpenCMAPI-WIR-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-WIR-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-WIR-005-M	Support Network Management APIs	7.6	
OpenCMAPI-WIR-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-WIR-007-M	Support Device Service APIs	7.8	
OpenCMAPI-WIR-008-M	Support PINs/PUKs Management APIs	7.10	
OpenCMAPI-WIR-009-O	Support UICC Management APIs	7.11	
OpenCMAPI-WIR-010-O	Support WLAN APIs	7.12	
OpenCMAPI-WIR-011-M	Support Statistics APIs	7.13	
OpenCMAPI-WIR-012-M	Support Information Status APIs	7.14	
OpenCMAPI-WIR-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-WIR-014-O	Support USSD Management APIs	7.16	
OpenCMAPI-WIR-015-O	Support GNSS APIs	7.17	
OpenCMAPI-WIR-016-O	Support Data Push Service Management APIs	7.18	
OpenCMAPI-WIR-017-M	Support Callback APIs	8	
OpenCMAPI-WIR-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-WIR-019-M	Support Contact Management APIs	7.19	
OpenCMAPI-WIR-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-WIR-021-M	Support Wireless Router APIs	7.21	
OpenCMAPI-WIR-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-WIR-023-O	Support M2M/IoT APIs	7.23	

B.4 SCR for M2M device

B.4.1 General M2M device

Item	Function	Reference	Requirement
OpenCMAPI-M2M-001-M	Support API Management	7.2	

Item	Function	Reference	Requirement
OpenCMAPI-M2M-002-M	Support Device Discovery APIs	7.3	
OpenCMAPI-M2M-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-M2M-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-M2M-005-M	Support Network Management APIs	7.6	
OpenCMAPI-M2M-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-M2M-007-M	Support Device Service APIs	7.8	
OpenCMAPI-M2M-008-M	Support PINs/PUKs Management APIs	7.10	
OpenCMAPI-M2M-009-O	Support UICC Management APIs	7.11	
OpenCMAPI-M2M-010-O	Support WLAN APIs	7.12	
OpenCMAPI-M2M-011-M	Support Statistics APIs	7.13	
OpenCMAPI-M2M-012-M	Support Information Status APIs	7.14	
OpenCMAPI-M2M-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-M2M-014-O	Support USSD Management APIs	7.16	
OpenCMAPI-M2M-015-O	Support GNSS APIs	7.17	
OpenCMAPI-M2M-016-O	Support Data Push Service Management APIs	7.18	
OpenCMAPI-M2M-017-M	Support Callback APIs	8	
OpenCMAPI-M2M-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-M2M-019-O	Support Contact Management APIs	7.19	
OpenCMAPI-M2M-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-M2M-021-O	Support Wireless Router APIs	7.21	
OpenCMAPI-M2M-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-M2M-023-O	Support M2M/IoT APIs	7.23	

B.4.2 Basic M2M device

Basic M2M device is a subset of M2M device representing devices that are able to perform only basic functions such as a sensor or a meter. These basic M2M devices are also referred as IoT (Internet of Things) devices.

Therefore, for each group of requirements, only some functions will be supported by Basic M2M devices (only the Mandatory functions are listed here – Any function not mentioned below is considered as Optional for Basic M2M).

Item	Function	Reference	Requirement
OpenCMAPI-IoT-001-M	CMAPI_API_Open()	7.2	
OpenCMAPI-IoT-002-M	CMAPI_API_Close()	7.2	
OpenCMAPI-IoT-003-M	CMAPI_API_GetOpenCMAPIVersion()	7.2	
OpenCMAPI-IoT-004-M	CMAPI_API_GetFunctionsSupported()	7.2	
OpenCMAPI-IoT-005-M	CMAPI_Discovery_OpenDevice()	7.3	
OpenCMAPI-IoT-006-M	CMAPI_Discovery_CloseDevice()	7.3	
OpenCMAPI-IoT-007-M	CMAPI_Network_GetRFInfo()	7.4	
OpenCMAPI-IoT-008-M	CMAPI_NetCon_GetConnectionStatus()	7.6	
OpenCMAPI-IoT-009-M	CMAPI_NetCon_SetAutoConnectMode()	7.6	
OpenCMAPI-IoT-010-M	CMAPI_NetCon_GetAutoConnectMode()	7.6	
OpenCMAPI-IoT-011-M	CMAPI_NetCon_SetPermittedBearers()	7.6	
OpenCMAPI-IoT-012-M	CMAPI_NetCon_GetPermittedBearers()	7.6	
OpenCMAPI-IoT-013-M	CMAPI_DevSrv_GetIMSI()	7.8	
OpenCMAPI-IoT-014-M	CMAPI_DevSrv_GetDeviceStatus()	7.8	
OpenCMAPI-IoT-015-M	CMAPI_DevSrv_GetFirmwareVersion()	7.8	
OpenCMAPI-IoT-016-M	CMAPI_DevSrv_GetRFSwitch()	7.8	
OpenCMAPI-IoT-017-M	CMAPI_DevSrv_SetRadioState()	7.8	

Item	Function	Reference	Requirement
OpenCMAPI-IoT-018-M	CMAPI_Information_GetNetworkSelectionMode()	7.14	
OpenCMAPI-IoT-019-M	CMAPI_Information_GetSignalStrength()	7.14	
OpenCMAPI-IoT-020-M	CMAPI_Information_GetRoamingStatus()	7.14	
OpenCMAPI-IoT-021-M	CMAPI_Information_GetRATType()	7.14	
OpenCMAPI-IoT-022-M	CMAPI_Information_GetRadioState()	7.14	
OpenCMAPI-IoT-023-M	CMAPI_Information_GetBatteryStatus()	7.14	
OpenCMAPI-IoT-024-M	CMAPI_SMS_Send()	7.15	
OpenCMAPI-IoT-025-O	CMAPI_IoT_IMSI_Attach()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-026-O	CMAPI_IoT_GPRS_Register()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-027-O	CMAPI_IoT_Set_PDPContext()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-028-O	CMAPI_IoT_GetPDPContextList()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-029-O	CMAPI_IoT_GetPDPContextIPaddress()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-030-O	CMAPI_IoT_Activate_PDPContext()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-031-O	CMAPI_IoT_SetNFM()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-032-O	CMAPI_IoT_GetNFM()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-033-O	CMAPI_IoT_SetBack-OffBaseInterval()	7.23	Optional function but recommended for this type of device
OpenCMAPI-IoT-034-O	CMAPI_IoT_GetBack-OffTimer()	7.23	Optional function but recommended for this type of device

B.5 SCR for Smart Phone

Item	Function	Reference	Requirement
OpenCMAPI-SMA-001-M	Support API Management	7.2	
OpenCMAPI-SMA-002-M	Support Device Discovery APIs	7.3	
OpenCMAPI-SMA-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-SMA-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-SMA-005-M	Support Network Management APIs	7.6	
OpenCMAPI-SMA-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-SMA-007-M	Support Device Service APIs	7.8	
OpenCMAPI-SMA-008-M	Support PINs/PUKs Management APIs	7.10	

Item	Function	Reference	Requirement
OpenCMAPI-SMA-009-M	Support UICC Management APIs	7.11	
OpenCMAPI-SMA-010-M	Support WLAN APIs	7.12	
OpenCMAPI-SMA-011-M	Support Statistics APIs	7.13	
OpenCMAPI-SMA-012-M	Support Information Status APIs	7.14	
OpenCMAPI-SMA-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-SMA-014-M	Support USSD Management APIs	7.16	
OpenCMAPI-SMA-015-O	Support GNSS APIs	7.17	
OpenCMAPI-SMA-016-M	Support Data Push Service Management APIs	7.18	
OpenCMAPI-SMA-017-M	Support Callback APIs	8	
OpenCMAPI-SMA-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-SMA-019-M	Support Contact Management APIs	7.19	
OpenCMAPI-SMA-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-SMA-021-O	Support Wireless Router APIs	7.21	
OpenCMAPI-SMA-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-SMA-023-O	Support M2M/IoT APIs	7.23	

B.6 SCR for Tablets

Item	Function	Reference	Requirement
OpenCMAPI-TAB-001-M	Support API Management	7.2	
OpenCMAPI-TAB-002-M	Support Device Discovery APIs	7.3	
OpenCMAPI-TAB-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-TAB-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-TAB-005-M	Support Network Management APIs	7.6	
OpenCMAPI-TAB-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-TAB-007-M	Support Device Service APIs	7.8	
OpenCMAPI-TAB-008-M	Support PINs/PUKs Management APIs	7.10	
OpenCMAPI-TAB-009-M	Support UICC Management APIs	7.11	
OpenCMAPI-TAB-010-M	Support WLAN APIs	7.12	
OpenCMAPI-TAB-011-M	Support Statistics APIs	7.13	
OpenCMAPI-TAB-012-M	Support Information Status APIs	7.14	
OpenCMAPI-TAB-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-TAB-014-M	Support USSD Management APIs	7.16	
OpenCMAPI-TAB-015-O	Support GNSS APIs	7.17	
OpenCMAPI-TAB-016-M	Support Data Push Service Management APIs	7.18	
OpenCMAPI-TAB-017-M	Support Callback APIs	8	
OpenCMAPI-TAB-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-TAB-019-M	Support Contact Management APIs	7.19	
OpenCMAPI-TAB-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-TAB-021-O	Support Wireless Router APIs	7.21	
OpenCMAPI-TAB-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-TAB-023-O	Support M2M/IoT APIs	7.23	

B.7 SCR for Cloud Devices

Item	Function	Reference	Requirement
OpenCMAPI-CLD-001-M	Support API Management	7.2	
OpenCMAPI-CLD-002-M	Support Device Discovery APIs	7.3	

Item	Function	Reference	Requirement
OpenCMAPI-CLD-003-M	Support Cellular Network Management APIs	7.4	
OpenCMAPI-CLD-004-M	Support Connection Management APIs	7.5	
OpenCMAPI-CLD-005-M	Support Network Management APIs	7.6	
OpenCMAPI-CLD-006-O	Support CDMA2000 APIs	7.7	
OpenCMAPI-CLD-007-M	Support Device Service APIs	7.8	
OpenCMAPI-CLD-008-M	Support PINs/PUKs Management APIs	7.10	
OpenCMAPI-CLD-009-M	Support UICC Management APIs	7.11	
OpenCMAPI-CLD-010-M	Support WLAN APIs	7.12	
OpenCMAPI-CLD-011-M	Support Statistics APIs	7.13	
OpenCMAPI-CLD-012-M	Support Information Status APIs	7.14	
OpenCMAPI-CLD-013-M	Support SMS Management APIs	7.15	
OpenCMAPI-CLD-014-M	Support USSD Management APIs	7.16	
OpenCMAPI-CLD-015-O	Support GNSS APIs	7.17	
OpenCMAPI-CLD-016-O	Support Data Push Service Management APIs	7.18	
OpenCMAPI-CLD-017-M	Support Callback APIs	8	
OpenCMAPI-CLD-018-O	Support Device Extended Service APIs	7.9	
OpenCMAPI-CLD-019-M	Support Contact Management APIs	7.19	
OpenCMAPI-CLD-020-O	Support P2P Direct Management APIs	7.20	
OpenCMAPI-CLD-021-O	Support Wireless Router APIs	7.21	
OpenCMAPI-CLD-022-O	Support IP Multimedia Services APIs	7.22	
OpenCMAPI-CLD-023-O	Support M2M/IoT APIs	7.23	

Appendix C. Description of OpenCMAPI functions (Informative)

This appendix provides a list of all OpenCMAPI Functions as well as a short description and in which version (from [OpenCMAPI-TS]) they have been created.

C.1 CMAPI-1 Functions

CMAPI-1		
Function	**Description**	**Vers.**
API MANAGEMENT		
CMAPI_API_Open()	initialize the OpenCMAPI	1.0
CMAPI_API_Close()	deallocate any internal API structures including the security context	1.0
CMAPI_API_GetOpenCMAPIVersion()	retrieve the version number of the OpenCMAPI used	1.0
CMAPI_API_GetFunctionsSupported()	retrieve the OpenCMAPI groups of functions supported by the enabler	1.1
DEVICE DISCOVERY APIs		
CMAPI_Discovery_DetectDevices()	search for devices	1.0
CMAPI_Discovery_GetDevice()	discover information about the devices within the system	1.0
CMAPI_Discovery_OpenDevice()	“open” a device within the system	1.0
CMAPI_Discovery_CloseDevice()	“close” a device within the system	1.0
CELLULAR NETWORK MANAGEMENT APIs		
CMAPI_Network_GetRFInfo()	get information about RF (Radio access technology, band class, data rate supported and channel)	1.0
CMAPI_Network_GetHomeInformation()	get information about home network of the subscriber for a dedicated System	1.0
CMAPI_Network_GetServingInformation()	get information about serving network of the subscriber	1.0
CONNECTION MANAGEMENT APIs		
CMAPI_NetConnectSrv_MgrCellularProfile()	manage cellular profiles, including add/delete/update a profile information	1.0
CMAPI_NetConnectSrv_GetCellularProfile()	get the details of a specific Cellular Profile	1.0
CMAPI_NetConnectSrv_GetCellularProfileList()	get a list of all Cellular Profile names	1.0
CMAPI_NetConnectSrv_SelectNetwork()	select the current network mode and PLMN for a given System	1.0
CMAPI_NetConnectSrv_GetNetworkList_Sync()	search and compile a list of available Networks	1.0
CMAPI_NetConnectSrv_GetNetworkList_Async()	initiate the search of the Network list	1.0
CMAPI_NetConnectSrv_GetCurrentConnType()	get the current connection type	1.0
CMAPI_NetConnectSrv_Connect_Async()	connect to a network	1.0
CMAPI_NetConnectSrv_Disconnect_Async()	disconnect from the network	1.0
CMAPI_NetConnectSrv_CancelConnect_Async()	cancel of connect operation (as a result of a call to **CMAPI_NetConnectSrv_Connect_Async**)	1.0
CMAPI_NetConnectSrv_SecondaryPDPContext_Connect_Async()	connect to a network	1.0
CMAPI_NetConnectSrv_SecondaryPDPContext_Disconnect_Async()	disconnect from the network	1.0
CMAPI_NetConnectSrv_SecondaryPDPContext_CancelConnect_Async()	cancel of connect operation (as a result of a call to **CMAPI_NetConnectSrv_SecondaryPDPContext_Connect_Async**)	1.0
NETWORK MANAGEMENT APIs		
CMAPI_NetCon_GetConnectionStatus()	obtain information about the connection status	1.0
CMAPI_NetCon_SetAutoConnectMode()	set/disable “autoconnect” mode	1.0
CMAPI_NetCon_GetAutoConnectMode()	return the current “autoconnect” mode	1.0
CMAPI_NetCon_SetDefaultProfile()	identify the profile that shall be used when the device is in auto connect mode	1.0
CMAPI_NetCon_SetPermittedBearers()	restrict the permitted mobile bearer when connecting to the selected network	1.0
CMAPI_NetCon_GetPermittedBearers()	get the current permitted bearers	1.0
CMAPI_NetCon_SetNoDataProfile()	set up (enable or disable) the nodataprofile	1.0

CMAPI_NetCon_GetNoDataProfile()	return the current state of the nodata profile (enabled or disabled)	1.0
CDMA2000 APIs		
CMAPI_CDMA2000_SetACCOLC()	set the Access Overload Class (ACCOLC) for CDMA2000 devices	1.0
CMAPI_CDMA2000_GetACCOLC()	retrieve the current value of the Access Overload Class (ACCOLC) for CDMA2000 devices	1.0
CMAPI_CDMA2000_SetCDMANetworkParameters()	set the values of certain CDMA2000-specific network parameters	1.0
CMAPI_CDMA2000_GetCDMANetworkParameters()	retrieve the values of certain CDMA2000-specific network parameters	1.0
CMAPI_CDMA2000_GetANAAAAAuthenticationStatus()	retrieve the value of the most recent ANA AAA authentication attempt status for CDMA2000 devices	1.0
CMAPI_CDMA2000_GetPRLVersion()	retrieve the value of the Preferred Roaming List (PRL) version in use for CDMA2000 devices	1.0
CMAPI_CDMA2000_GetERIFile()	retrieve the contents of the Enhanced Roaming Indicator (ERI) file in use for CDMA2000 devices	1.0
CMAPI_CDMA2000_ActivateAutomatic()	command the device to perform automatic activation using a specified activation code	1.0
CMAPI_CDMA2000_ActivateManual()	command the device to perform manual activation using the specified parameters	1.0
CMAPI_CDMA2000_ValidateSPC()	command the device to validate a Service Programming Code (SPC)	1.0
CMAPI_OMADM_StartSession()	start an OMA DM session to configure the values of various CDMA2000 network information as specified by the session type in its input parameter	1.0
CMAPI_OMADM_CancelSession()	cancel an ongoing OMA DM session	1.0
CMAPI_OMADM_GetSessionInfo()	return information about the currently active OMA DM session (or the most recent session if none is active)	1.0
CMAPI_OMADM_GetPendingNIA()	return information about a Network-Initiated Alert (NIA) that is commanding the device to establish a DM session with a DM server to perform the requested configuration operation	1.0
CMAPI_OMADM_SendSelection()	return the response from the device to a Network-Initiated Alert (NIA) that is commanding the device to establish a DM session	1.0
CMAPI_OMADM_GetFeatureSettings()	return information about the settings of OMA DM features, indicating for each one whether OMA DM can be currently used for the specified configuration operation	1.0
CMAPI_OMADM_SetProvisioningFeature()	enable and disable the OMA DM device service provisioning update feature	1.0
CMAPI_OMADM_SetPRLUpdateFeature()	enable and disable the OMA DM PRL update feature	1.0
CMAPI_OMADM_SetFirmwareUpdateFeature() (Optional)	enable and disable the OMA DM Firmware update feature	1.0
CMAPI_OMADM_ResetToFactoryDefaults()	reset the device to factory default	1.0
CMAPI_OMADM_InitiateOTASP()	activate the device using OTA activation	1.0
CMAPI_OMADM_SetPRL()	update PRL/PLMN by uploading a PRL file	1.0
CMAPI_MobileIP_SetState()	set the current Mobile IP state of the device	1.0
CMAPI_MobileIP_GetState()	retrieve the current Mobile IP state of the device	1.0
CMAPI_MobileIP_SetActiveProfile()	set the index of the Mobile IP profile that the device will use	1.0
CMAPI_MobileIP_GetActiveProfile()	retrieve the index of the Mobile IP profile that the device is currently using	1.0
CMAPI_MobileIP_SetProfile()	configure the contents of a Mobile IP profile on the device	1.0
CMAPI_MobileIP_GetProfile()	retrieve the contents of a Mobile IP profile on the device	1.0
CMAPI_MobileIP_SetParameters()	set various parameters that configure the behaviour of the device's Mobile IP client	1.0
CMAPI_MobileIP_GetParameters()	retrieve the current values of the parameters that configure the behaviour of the device's Mobile IP client	1.0
CMAPI_MobileIP_GetLastError()	retrieve the last Mobile IP error that occurred (refer to RFC3344 for a list of error codes)	1.0
DEVICE SERVICE APIs		
CMAPI_DevSrv_GetManufacturerName()	retrieve the name of the manufacturer of the device	1.0
CMAPI_DevSrv_GetManufacturerModel()	retrieve the product model ID of the device	1.0
CMAPI_DevSrv_GetDeviceName()	retrieve the commercial name of the device	1.0
CMAPI_DevSrv_GetHardwareVersion()	retrieve the hardware version of the device	1.0
CMAPI_DevSrv_GetProductType()	retrieve the product type of the device	1.0

CMAPI_DevSrv_GetIMSI()	retrieve the active IMSI(s) from SIM/R-UIM/NAA on UICC	1.0
CMAPI_DevSrv_GetMDN()	retrieve the MDN (only applicable to 3GPP2 systems)	1.0
CMAPI_DevSrv_GetIMEI()	retrieve the IMEI (only applicable to 3GPP systems)	1.0
CMAPI_DevSrv_GetESN()	retrieve the ESN (only applicable to 3GPP2 systems)	1.0
CMAPI_DevSrv_GetMEID()	retrieve the MEID (only applicable to 3GPP2 systems)	1.0
CMAPI_DevSrv_GetMSISDN()	retrieve the MSISDN from the active NAA in the SIM/UICC (only applicable to 3GPP systems)	1.0
CMAPI_DevSrv_GetDeviceStatus()	retrieve the device status	1.0
CMAPI_DevSrv_GetFirmwareVersion()	retrieve the firmware version of the device	1.0
CMAPI_DevSrv_GetRFSwitch()	retrieve the radio switch status (Radio On / Off)	1.0
CMAPI_DevSrv_SetRadioState()	set the radio power state of the device	1.0
CMAPI_DevSrv_SetRadioState_Async()	set the power state of a radio within a device	1.0
CMAPI_DevSrv_GetControlKeyStatus()	get the specified Mobile Equipment (device) de-personalization control key status	1.0
CMAPI_DevSrv_DeactivateControlKey()	deactivate the specified Mobile Equipment (device) de-personalization control key	1.0
CMAPI_DevSrv_UnblockControlKey() (Optional)	unblock the specified Mobile Equipment (device) de-personalization control key	1.0
CMAPI_DevSrv_DevAttributes()	provide to application information regarding device attributes (e.g. screen, keypad, camera, microphone, loudspeaker)	1.1
DEVICE EXTENDED SERVICE APIs		
CMAPI_ExtDevSrv_NFC()	provide to application information regarding NFC functionalities available in the device	1.1
CMAPI_ExtDevSrv_SE()	provide to application information regarding SE (Secure Element) functionalities and services available in the device	1.1
PINS/PUKS MANAGEMENT APIs		
CMAPI_DevSrv_GetNAAavailable()	get all the available NAAs and the corresponding Application labels	1.0
CMAPI_DevSrv_EnablePIN()	enable PIN protection	1.0
CMAPI_DevSrv_DisablePIN()	disable PIN protection	1.0
CMAPI_DevSrv_VerifyPIN()	verify a PIN	1.0
CMAPI_DevSrv_UnblockPIN()	unblock a PIN	1.0
CMAPI_DevSrv_ChangePIN()	change a PIN	1.0
UICC MANAGEMENT APIs		
CMAPI_UICC_GetTerminalProfile()	get the last TERMINAL PROFILE sent by the device to the SIM/R-UIM/UICC	1.0
CMAPI_UICC_SetTerminalProfile()	transmit to the SIM/R-UIM/UICC via the device the ToolKit functions (i.e.: the TERMINAL PROFILE) that are supported by the Connection Manager Applications	1.0
CMAPI_UICC_SendToolKitEnvelopeCommand()	transmit to the SIM/R-UIM/UICC via the device any ToolKit ENVELOPE command that is supported by the Connection Manager Application and for which no overlapping was identified	1.0
CMAPI_UICC_SendTerminalResponse()	send a TERMINAL RESPONSE to the SIM/R-UIM/UICC via the device answering to any ToolKit Proactive Command received via the Callback **CMAPI_UICC_ToolKitProactiveCommand()**	1.0
WLAN APIs		
CMAPI_WLAN_IsSupported()	determine if WLAN functionality is supported	1.0
CMAPI_WLAN_AddKnownNetwork()	add a network to the known network list	1.0
CMAPI_WLAN_UpdateKnownNetwork()	update an existing known network record	1.0
CMAPI_WLAN_DeleteKnownNetwork()	remove the entry from the known networks list at the specified index	1.0
CMAPI_WLAN_GetKnownNetwork()	retrieve the known network record information	1.0
CMAPI_WLAN_GetScanResults()	retrieve the list of available WLAN networks	1.0
CMAPI_WLAN_Scan_Async()	initiate a scan for WLAN networks	1.0
CMAPI_WLAN_Connect()	connect to a WLAN network	1.0
CMAPI_WLAN_ConnectKnownNetwork()	connect to a WLAN network in the known networks list	1.0
CMAPI_WLAN_Disconnect()	disconnect any connected WLAN network	1.0

CMAPI_WLAN_GetConnectionMode()	determine if connectivity is being actively sought by the enabler or if manual connection requests are required	1.0
CMAPI_WLAN_SetConnectionMode()	change the connectivity mode	1.0
CMAPI_WLAN_ResetDevice()	reset the device	1.0
CMAPI_WLAN_GetConnectedParameters()	retrieve values related to the associated network.	1.0
CMAPI_WLAN_SetConnectedParameters()	set various attributes of an existing connection	1.0
CMAPI_WLAN_CancelOperation()	cancel any pending operation like connect or scan	1.0
CMAPI_WLAN_ConnectWPS()	initiate a connection with the WPS button push method.	1.0
CMAPI_WLAN_ConnectPinWPS()	initiate a connection with the WPS pin method	1.0
CMAPI_WLAN_ConnectionState()	determine if WLAN is connected	1.0
CMAPI_WLAN_SearchNetwork_Async()	check the availability of a specific WLAN network	1.0
CMAPI_WLAN_EnableCapability()	enable or disable the WLAN feature in the device	1.1
CMAPI_WLAN_AuthenticationSupported()	determine if HS2.0 is supported by the device and what are the authentications methods supported	1.1
CMAPI_WLAN_ManageKnownNetwork()	add/delete or update a network to or in the known network list.	1.1
CMAPI_WLAN_Get_WSIDL()	retrieve the user preferred list and operator preferred list of WLAN specific identifier (WSID) from the SIM/R-UIM/NAA on UICC	1.1
CMAPI_WLAN_Get_HS2MOSubcription()	retrieve the elements related to HS2.0 subscriptions	1.1
CMAPI_WLAN_Get_ANDSFMOSubcription()	retrieve the elements related to ANDSF subscription	1.1
CMAPI_WLAN_GetANQP()	get the ANQP information including HS2.0 ANQP	1.1
CMAPI_WLAN_Get_WLANSettings()	retrieve the user and operator settings for WLAN.	1.1
CMAPI_WLAN_Set_WLANUserSettings()	set the user settings for WLAN.	1.1
STATISTICS APIs		
CMAPI_NetStatistic_GetConnectionStatistics()	obtain network traffic statistics info	1.0
CMAPI_NetStatistic_GetAllConnectionRecords()	retrieve all connection records.	1.1
CMAPI_NetStatistic_DeleteConnectionRecord()	delete a connection record.	1.1
INFORMATION STATUS APIs		
CMAPI_Information_GetPINStatus()	return the status of the PINs and PUKs of all active SIM/R-UIM/NAA on UICC for a dedicated device	1.0
CMAPI_Information_GetNetworkSelectionMode()	determine the network selection mode	1.0
CMAPI_Information_GetSignalStrength()	obtain the current signal strength value, the percentage of signal present and the signal quality	1.0
CMAPI_Information_GetCSNetworkRegistration()	determine if a circuit switched registration is present	1.0
CMAPI_Information_GetPSNetworkRegistration()	determine if a packet switched attachment is present	1.0
CMAPI_Information_GetAPN()	obtain the APN identifier	1.0
CMAPI_Information_GetIPAddress()	retrieve the current IP address assigned to the device and the type of the address assigned	1.0
CMAPI_Information_GetRoamingStatus()	retrieve the current roaming status	1.0
CMAPI_Information_GetDriverVersion()	retrieve the driver version	1.0
CMAPI_Information_GetRATType()	retrieve the radio access technology	1.0
CMAPI_Information_GetQoS()	retrieve the QoS parameters related to the network	1.0
CMAPI_Information_GetWLANConnection()	retrieve identifying data of the currently connected network.	1.0
CMAPI_Information_GetRadioState()	return the power state of a radio within a device	1.0
CMAPI_Information_GetICCID()	get the ICCID	1.0
CMAPI_Information_GetBatteryStatus()	retrieve the current status of the battery of device if applicable	1.1
CMAPI_Information_SetBatteryThreshold()	set thresholds for the battery status	1.1
CMAPI_Information_GetMobilityState()	retrieve the current mobility state of the device	1.1
CMAPI_Information_GetMobilitytoLocation()	evaluate if the device is moving compared to a specific location (for example, an Access Point). The result is reported in callback **CMAPI_Callback_ GetMobilitytoLocation_Complete()**	1.1
SMS MANAGEMENT APIs		
CMAPI_SMS_Send()	send SMS	1.0

CMAPI_SMS_Get()	retrieve the message	1.0
CMAPI_SMS_Delete()	delete SMS	1.0
CMAPI_SMS_GetIDList()	get the list of SMS stored on local device or SIM or the terminal device like PC	1.0
CMAPI_SMS_Update()	update the status of the SMS	1.0
CMAPI_SMS_GetSMSCAddress()	get the address of SMSC	1.0
CMAPI_SMS_SetSMSCAddress()	set the address of SMSC	1.0
CMAPI_SMS_GetValidityPeriod()	get the validity period setting	1.0
CMAPI_SMS_SetValidityPeriod()	set the period of validity of a SMS	1.0
CMAPI_SMS_GetDeliveryReport()	get the delivery report setting	1.0
CMAPI_SMS_SetDeliveryReport()	set the delivery report “On” or “Off”	1.0
CMAPI_SMS_GetRecordCount()	retrieve the number of SMS segments	1.0
CMAPI_SMS_GetUnreadRecordCount()	retrieve the number of unread SMS records	1.0
CMAPI_SMS_Create()	create a draft SMS	1.1
USSD MANAGEMENT APIs		
CMAPI_USSD_Request()	build up a USSD request to the network	1.0
CMAPI_USSD_Release()	release the USSD session	1.0
GNSS APIs		
CMAPI_GNSS_SetState()	set the state of the GNSS functionality on the device	1.0
CMAPI_GNSS_GetState()	retrieve the state of the GNSS functionality on the device	1.0
CMAPI_GNSS_SetTrackingParameters()	set the values of parameters that control the operation of GNSS tracking on the device	1.0
CMAPI_GNSS_GetTrackingParameters()	retrieve the values of parameters that control the operation of GNSS tracking on the device	1.0
CMAPI_GNSS_SetAGPSConfig()	configure the Assisted GPS (AGPS) server IP address, port number and/or FQDN	1.0
CMAPI_GNSS_GetAGPSConfig()	retrieve the values of the Assisted GPS (AGPS) server IP address, port number and FQDN	1.0
CMAPI_GNSS_SetAutomaticTracking()	enable and disable automatic GNSS tracking on the device	1.0
CMAPI_GNSS_GetAutomaticTracking()	retrieve the state of automatic GNSS tracking on the device	1.0
CMAPI_GNSS_GetDevicePosition()	retrieve the current position of the device	1.0
CMAPI_GNSS_SetSystemTime()	set the value of the system time	1.0
DATA PUSH SERVICE MANAGEMENT APIs		
CMAPI_Push_Enable()	turn on PUSH option	1.0
CMAPI_Push_Disable()	turn off PUSH option	1.0
CMAPI_Push_GetRadioType()	get the current bearer type over which the PUSH session is established for an application	1.0
CONTACT MANAGEMENT APIs		
CMAPI_Contact_Create()	create a contact	1.1
CMAPI_Contact_Get()	retrieve the details of a contact	1.1
CMAPI_Contact_Delete()	delete a contact	1.1
CMAPI_Contact_GetContactList()	get the list of contacts stored on local device or SIM or the terminal device like PC	1.1
CMAPI_Contact_Update()	update an existing contact	1.1
CMAPI_Contact_Search()	search for a specific contact name in the list of contacts	1.1
P2P DIRECT MANAGEMENT APIs		
CMAPI_P2P_GetP2PInfo()	detect which P2P direct connection technology(ies) is/are supported if any	1.1
CMAPI_P2P_EnableDirectDiscovery()	activate the P2P Direct Discovery Feature in a P2P Direct enabled device	1.1
CMAPI_P2P_DisableDirectDiscovery()	deactivate the P2P Direct Discovery feature in a P2P Direct enabled device	1.1
CMAPI_P2P_EnableDirectConnection()	activate the P2P Direct Connection feature in a P2P Direct enabled device	1.1
CMAPI_P2P_DisableDirectConnection()	deactivate the P2P Direct Connection feature in a P2P Direct	1.1

	enabled device	
CMAPI_P2P_DiscoveryResolve()	resolve a ServiceRecord for metadata and/or connection info	1.1
CMAPI_P2P_DiscoveryMonitor()	request discovery of Remote Device(s) and the services offered	1.1
CMAPI_P2P_DiscoveryAnnounce()	announce its presence and P2P Direct services supported to Remote Device(s)	1.1
CMAPI_P2P_EstablishConnection()	request the Local Device to establish a connection to a Remote Device	1.1
CMAPI_P2P_RejectConnection()	reject an incoming connection request	1.1
CMAPI_P2P_AcceptConnection()	accept an incoming connection request from a Remote Device	1.1
CMAPI_P2P_CloseConnection()	request the Local Device to close an existing connection to a Remote Device	1.1
CMAPI_P2P_GetConnectionStatus()	retrieve the status of the P2P Direct connection	1.1
CMAPI_P2P_EnableRelay()	request the Local Device to act as a relay to share its data connection with Remote Device members of the group (i.e. enable concurrent operations)	1.1
CMAPI_P2P_DisableRelay()	request the Local Device to stop acting as a relay to share its data connection with Remote Device members of the group	1.1
CMAPI_P2P_CreateGroup()	create a new P2P Direct group with one or several Remote Device (s)	1.1
CMAPI_P2P_RemoveGroup()	remove a P2P group, previously created by the Local Device	1.1
CMAPI_P2P_EnableMembershipInSeveralGroups()	enable a Local Device to be a member of several groups simultaneously	1.1
CMAPI_P2P_DisableMembershipInSeveralGroups()	disable a Local Device to be a member of several groups simultaneously	1.1
CMAPI_P2P_RemoveDeviceFromGroup()	remove a Remote Device from an existing group the Local Device owns	1.1
CMAPI_P2P_AcceptInvitationToGroup()	accept an group join invitation on the receiver side	1.1
CMAPI_P2P_JoinGroup()	invite a Remote Device to join an existing group	1.1
CMAPI_P2P_RejectInvitationToGroup()	reject an invitation to join an existing group	1.1
CMAPI_P2P_RejectJoiningGroup()	reject a Remote Device from joining to an existing group	1.1
CMAPI_P2P_RequestToJoinGroup()	send a request for joining an existing group to the group owner	1.1
CMAPI_P2P_RestrictFromGroup()	instruct the Local Device to be restricted from an existing group owned by a Remote Device	1.1
CMAPI_P2P_GetGroupInfo()	retrieve from the Local Device which P2P Direct enabled device(s) are in an existing group to which the Local Device is a member of	1.1
CMAPI_P2P_AllowSimultaneousConnection()	allow the device to have a P2P connection simultaneously to a normal data connection using the same radio technology.	1.1
CMAPI_P2P_DisallowSimultaneousConnection()	disallow the device to have a P2P connection simultaneously to a normal data connection using the same radio technology.	1.1
WIRELESS ROUTER APIs		
CMAPI_Router_GetConfigurations()	read the configuration values of a router (ssid, users, security, etc) of all defined routers of a physical router device	1.1
CMAPI_Router_SetConfiguration()	write the configuration values of a router (ssid, users, security, etc)	1.1
CMAPI_Router_DeleteConfiguration()	delete a router and its configuration	1.1
CMAPI_Router_GetConnectedDevices()	retrieve a list of Connected Devices connected to a router	1.1
CMAPI_Router_GetPolicies()	retrieve a list of policies within a router	1.1
CMAPI_Router_SetPolicy()	add or update a policy to a router's policies	1.1
CMAPI_Router_DeletePolicy()	delete a Connected Device policy from a router's policies	1.1
CMAPI_Router_GetRestrictions()	retrieve a list of Connected Device restrictions within a route	1.1
CMAPI_Router_SetRestriction()	add or update a Connected Device restriction to a router	1.1
CMAPI_Router_DeleteRestriction()	remove a Connected Device restriction	1.1
CMAPI_Router_SetAdminPassword()	update a router administrator password	1.1
CMAPI_Router_VerifyAdminPassword()	verify a router administrator password and to report the number of failed access attempts	1.1
CMAPI_Router_ResetToDefaults()	return a router to factory default settings	1.1
IP Multimedia Services APIs		
CMAPI_IMS_GetISIMinfo()	retrieve if there is an ISIM in the UICC for a specific radio system (either 3GPP or 3GPP2) and provide the IMPU, IMPI & Home	1.1

	Domain Name (relevant for IMS context) related.	
CMAPI_IMS_GetIARIinfo()	retrieve the IARI information from the ISIM for a dedicated radio system (either 3GPP or 3GPP2) on the UICC.	1.1
M2M/IoT APIs		
CMAPI_IoT_IMSI_Attach()	request an IMSI attach or detach.	1.1
CMAPI_IoT_GPRS_Register()	request an GPRS attach or detach.	1.1
CMAPI_IoT_Set_PDPContext()	define a PDP context.	1.1
CMAPI_IoT_GetPDPContextList()	get the list of currently defined PDP Contexts.	1.1
CMAPI_IoT_GetPDPContextIPaddress()	retrieve the IP address of the PDP context concerned.	1.1
CMAPI_IoT_Activate_PDPContext()	activate or deactivate a PDP context.	1.1
CMAPI_IoT_SetNFM()	set up (enable or disable) the Network Friendly Mode of the modem if supported	1.1
CMAPI_IoT_GetNFM()	return the current state of the Network Friendly Mode of the modem (enabled or disabled)	1.1
CMAPI_IoT_SetBack-OffBaseInterval()	configure the Back-off Base Intervals of the modem (time between re-attempts of whatever action previously failed)	1.1
CMAPI_IoT_GetBack-OffTimer()	retrieve the time left of the back-off Timer.	1.1

Table 12: List CMAPI-1 Functions

C.2 CMAPI-2 Functions

CMAPI-2		
Function	**Description**	**Vers.**
REGISTRATION APIs		
CMAPI_Callback_Register()	register for the callbacks which are expected to be received	1.0
CMAPI_Callback_Unregister()	turn off all callbacks or just some	1.0
CALLBACK APIs		
CMAPI_Callback_DetectDevicesComplete()	communicate that a search and validation of the devices in the system is complete	1.0
CMAPI_Callback_DeviceChanged()	communicate whenever there is a change in a given device state in particular indicate that a device has become present or been removed	1.0
CMAPI_Callback_GetNetworkList_Async_Complete()	result of a previous call made to **CMAPI_NetConnectSrv_GetNetworkList_Async()**.	1.0
CMAPI_Callback_Connect_Async_Complete()	result of a previous call to **CMAPI_NetConnectSrv_Connect_Async()**	1.0
CMAPI_Callback_Disconnect_Async_Complete()	result of a previous call to **CMAPI_NetConnectSrv_Disconnect()**	1.0
CMAPI_Callback_CancelConnect_Async_Complete()	result of a previous call to **CMAPI_NetConnectSrv_CancelConnect_Async()**	1.0
CMAPI_Callback_SessionStateChange()	communicate the session state change	1.0
CMAPI_Callback_BearerStatusChange()	communicate a bearer status change	1.0
CMAPI_Callback_TrafficChannelDormancy()	communicate the changes in the traffic level	1.0
CMAPI_Callback_CDMA2000ActivationState()	communicate the changes in the CDMA 2000 Activation state	1.0
CMAPI_Callback_SearchWLANNetworkComplete()	result of a previous call to **CMAPI_WLAN_SearchNetwork_Async()**	1.0
CMAPI_Callback_RadioState()	communicate changes in the radio power state	1.0
CMAPI_Callback_SetRadioState_Async_Complete()	result of a previous call to **CMAPI_DevSrv_SetRadioState_Async()**	1.0
CMAPI_Callback_Roaming()	indicate changes in Roaming status	1.0
CMAPI_Callback_SignalStrength()	return the current signal strength value, the percentage of signal present and the signal quality	1.0
CMAPI_Callback_GNSS()	indicate a change in the GNSS state	1.0
CMAPI_Callback_SMS()	indicate that a new SMS message has been received and the number of segments in the mailbox	1.0
CMAPI_Callback_SMS_Message()	provide to application the new received message while not only a notice that a new message is received	1.0
CMAPI_Callback_ByteCount	indicate the current byte count	1.0
CMAPI_Callback_USSD()	communicate a USSD message	1.0

CMAPI_Callback_QoSChange()	communicate a change in QoS	1.0
CMAPI_Callback_RFInformationChange()	communicate a change related to RF	1.0
CMAPI_Callback_PINPUKStatus()	return the status of the PINs/PUKs for all active NAAs	1.0
CMAPI_Callback_ScanWLANComplete()	notify that a scan for WLAN networks has been completed. result of a previous call to **CMAPI_WLAN_Scan_Async()**	1.0
CMAPI_Callback_WLANNewAvailableNetwork()	notify that a new network has been discovered	1.0
CMAPI_Callback_WLANConnectionStatus()	receive WLAN connection Status	1.0
CMAPI_Callback_PUSHReceived()	notify an application when a new PUSH message has been received	1.0
CMAPI_Callback_OMADMStatus()	indicate any OMA-DM operation Progress or Status in-between	1.0
CMAPI_Callback_UICC_ToolKitProactiveCommand()	receive the ToolKit Proactive Commands sent by the SIM/R-UIM/UICC	1.0
CMAPI_Callback_UICC_DeviceTerminalProfile()	receive the TERMINAL PROFILE sent by the device to the SIM/R-UIM/UICC	1.0
CMAPI_Callback_VerifyPIN()	signal that a PIN should be collected from the user and supplied to the API through the **CMAPI_DevSrv_VerifyPIN()** method	1.0
CMAPI_Callback_PermittedBearersChange()	notify that a change occurred in the PermittedBearers for the device	1.0
CMAPI_Callback_NetConnectSrv_SecondaryPDPContext_Connect_Async_Complete()	result of a previous call to **CMAPI_NetConnectSrv_SecondaryPDPContext_Connect_Async()**	1.0
CMAPI_Callback_NetConnectSrv_SecondaryPDPContext_Disconnect_Async_Complete()	result of a previous call to **CMAPI_NetConnectSrv_SecondaryPDPContext_Disconnect_Async()**	1.0
CMAPI_Callback_NetConnectSrv_SecondaryPDPContext_CancelConnect_Async_Complete()	result of a previous call to **CMAPI_NetConnectSrv_SecondaryPDPContext_CancelConnect_Async()**	1.0
CMAPI_Callback_WLANSettingsChanged()	notify that a WLAN operator setting has been changed.	1.1
CMAPI_Callback_WLANNewMO()	notify that a new or an updated WLAN MO has been provided to the Terminal.	1.1
CMAPI_Callback_Incoming_Voice_Call()	provide to application information regarding a voice call state (incoming, established...)	1.1
CMAPI_Callback_SEServicesChange()	communicate changes regarding the availability of services	1.1
CMAPI_Callback_BatteryStatusChanged()	communicate whenever there is a change in the battery status	1.1
CMAPI_Callback_BatteryThresholdReached()	communicate whenever the battery level of the device is reaching a threshold set by the function **CMAPI_Information_SetBatteryThreshold()**	1.1
CMAPI_Callback_P2P_DiscoveryMatch()	alert the Local Device of a DeviceID/ServiceID/ServiceRecord discovery match as indicated in **CMAPI_P2P_Monitor()**	1.1
CMAPI_Callback_P2P_Connection()	result of call made to **CMAPI _P2P_EstablishConnection()**	1.1
CMAPI_Callback_P2P_GroupNotification()	result of a previous call to **CMAPI_P2P_JoinGroup()**	1.1
CMAPI_Callback_ GetMobilitytoLocation_Complete()	result of a previous call to **CMAPI_Information_GetMobilitytoLocation()**	1.1

Table 13: List CMAPI-2 Functions

Appendix D. Web IDL Definitions (Informative)

For the definitions of the WebIDL for the CMAPI WebAPI please refer to [OpenCMAPI-SUP-WIDL].

Appendix E. JavaScript Library of WebSocket API Binding (Informative)

This appendix describes how the JavaScript Library implements the Web API Binding using the methods and event handlers of corresponding Web API, and the details of the JSON-RPC data structure of request and response messages as the Application Data within the underlying Web protocol.

A JavaScript Library implements the Web Socket API Binding using the following algorithm:

- Establishing one and only one persistent “WebSocket” object for a Web application
- Keep the Web Socket connection open unless error happens
- Maintain the list of registered CMAPI-2 “callbackIds” and corresponding callback function “cbs”
- Maintain the list of transaction “ids” and corresponding callback functions “cbs” for outstanding CMAPI-1 function calls
    - The validity period of an outstanding CMAPI-1 function call is defined according to operator’s policy in order to handle the possible situation of multiple sequential response messages of the same CMAPI-1 function call. For example, 30 minutes. If it is expired, the transaction “id” and corresponding callback function “cb” shall be removed from the list.
- Handle errors according to section 5.4.1.4.