Service Layer API for oneM2M

Draft

51 Pages

Text in Red is here to help you. Delete it when you have followed the instructions.
The <RFC Title> can be set from the File>Properties:User Defined menu. To update it onscreen, press F9. To update all of the fields in the document Select All (CTRL-A), then hit F9. Set the release level by selecting one from: Draft, Final Draft, Release. The date is set automatically when the document is saved.

Abstract

10 point Arial Centered.

oneM2M is standard organization and specifies middleware for IoT, called Common Services Entities (CSE). Application can access functionality in CSE with RESTful operations, which are Create, Retrieve, Update, Delete and Notify. oneM2M allows variety of communication methods, 4 protocol bindings (HTTP, MQTT, CoAP, Websocket) and 3 serializations (XML, JSON, CBOR). This RFC describes the way to provide high level API for oneM2M RESTful operations hiding the difference of variety of communication methods.

0 Document Information

0.1 License

0.2 Trademarks

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 10.1.

`Source code is shown in this typeface.`

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	*SEP 19 2017*	*Put information relating to the changes you have made here.* *<name>, <company> <e-mail>* Initial Contribution. Hiroyuki Maeomichi, NTT, maeomichi.hiroyuki@lab.ntt.co.jp

1 Introduction

Introduce the RFC. Discuss the origins and status of the RFC and list any open items to do.

oneM2M is standard organization and specifies middleware for Internet of Things (IoT), called Common Services Entities (CSE). Applications can access CSE's functionality with RESTful operations, which are Create, Retrieve, Update, Delete and Notify. TS-0001 [2] defines more than 40 resource types to expose CSE's functionalities. oneM2M allows variety of communication methods, combination of 4 protocol bindings (HTTP, MQTT, CoAP, Websocket) and 3 serializations (XML, JSON, CBOR).

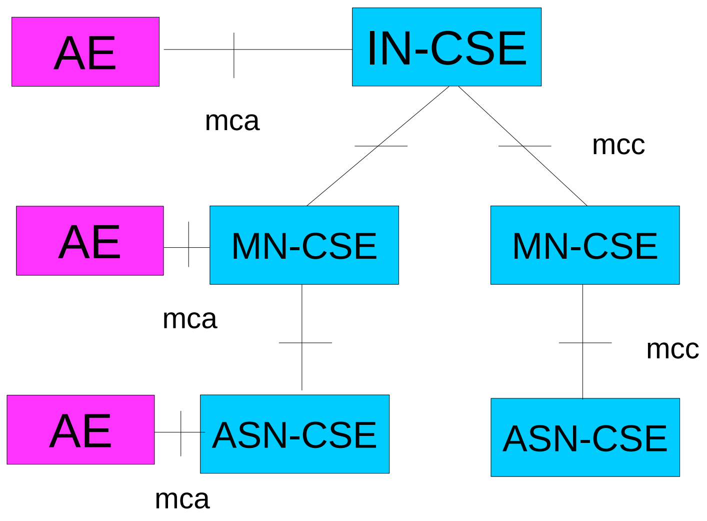
This RFP discuss the way to provide high level API (namely service layer API) for oneM2M RESTful operations hiding the difference of variety of communication methods.

2 Application Domain

This section should be copied from the appropriate RFP(s). It is repeated here so it can be extended while the RFC authors learn more subtle details.

2.1 IoT Application configuration using oneM2M

oneM2M's middleware, called CSE can be deployed in different locations and they are connected each other forming tree topology. Depending on deployed location, CSEs are categorized to 3 types, IN-CSE, MN-CSE and ASN-CSE. IN-CSE is located top of tree, ASN-CSE is located at leaf and MN-CSE is located and MN-CSE is located on middle.

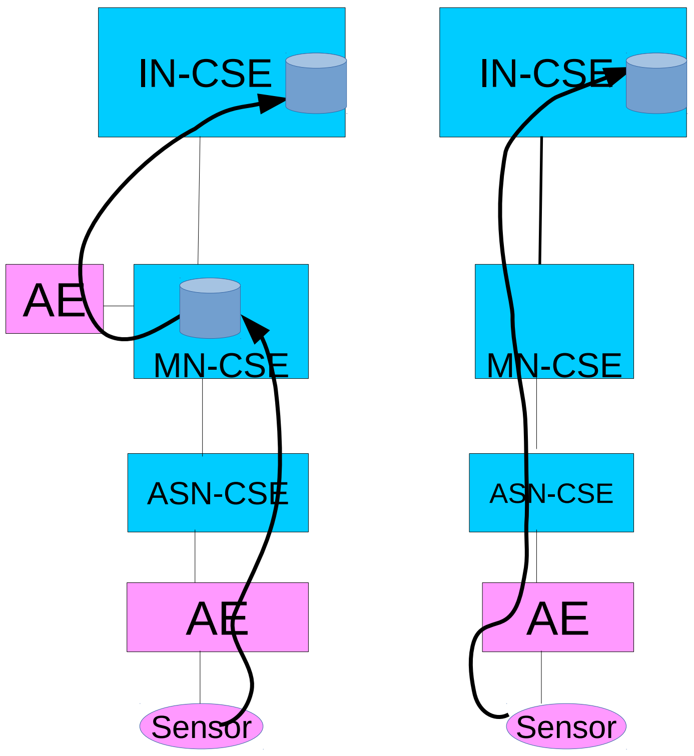


oneM2M’s application, called Application Entity (AE) connects to one of CSEs. After AE connecting to the CSE, AE can access to all of CSEs, by retargeting function of CSE.

AE accesses to CSE’s functionality through RESTful API, which consists of Create, Retrieve, Update, Delete and Notify in targeting more than 40 types of resources. For examples, typical resources are <*contentInstance*> that expresses IoT data and <*container*> that holds set of <*contentInstance*>s. AE can create or retrieve the <contentInstance> on any CSE by the retargeting functionality, as far as permission is allowed. Interface between CSEs is called *mcc* and interface between CSE and AE is called *mca,* both interfaces have almost same interface.

It is possible to develop variety types of distributed applications using the architecture. For example for IoT data aggregation applications, it is possible to develop gradual aggregation type or direct aggregation type. In gradual aggregation type, AE connected to ASN-CSE creates <*conentInstance*>s in ASN-CSE, and intermediate applications calculate statistics and put the result on IN-CSE as a <*contentInstance*>, while, in direct aggregation type, AE connected to ASN-CSE creates <*contentInstance*>s in IN-CSE directly.

Under CSE layer, oneM2M specifies NSE(Network Services Entity), but this RFC doesn’t cover the NSE layer.



2.2 Communication methods used in oneM2M

oneM2M allows variety of communication methods, combination of 4 protocol bindings (HTTP, MQTT, CoAP, Websocket) and 3 serializations (XML, JSON, CBOR). It might be added in future. oneM2M specifies specification in different level.

Firstly TS-0001[2] specifies high level resource definitions, it defines more than 40 resource types, such as <contentInstance> for storing IoT data, <timeSeriesInstance> for periodic sensor measurement with leap detection mechanism.

Secondly TS-0004[3] specifies procedures and serializations in independent manner from protocol bindings. Resource type and protocol data unit are defined using XSD for XML serialization. Mapping between XML and other serializations are also specified.
Thirdly TS-0008, TS-0009, TS-0010, TS-0020 specify protocol specific details for CoAP, HTTP, MQTT and Web Socket respectively.

2.3 Long name and short name

oneM2M introduced two types of notation, called long name and short name for resource types, attribute and so on. Long name is human friendly string and specifications mainly use this notation, while short name is short string consist of typically 2 or 3 characters (but not limited and sometimes longer) and communication protocol use this notation. In most cases, the initial characters of long name are assigned as short name, for examples, ct for CreationTime and at for AnnounceTo.

3 Problem Description

This section should be copied from the appropriate RFP(s). It is repeated here so it can be extended while the RFC authors learn more subtle details.

oneM2M specifies protocol based interface, but doesn't specify programing level API. As previously mentioned oneM2M allows variety of communication methods which are the combinations of 4 protocol bindings (HTTP, MQTT, CoAP, Websocket) and 3 serializations (XML, JSON, CBOR).

First problem is application portability. Without standardized API, application program tends to depend on the communication method initially intend to use and it will became hard to run another environment in which uses another communication method. (For example, an application designed for XML/HTTP, tend to run on environment use JSON/Websocket)

Second problem is the latency of the communication between CSE and application. Even if CSE and application is located in the same box, current oneM2M specifications define methods through protocols which requires serialization/deserialization of data, context-switch of applications, validation of incoming data and resulted in large latency compared to the situation both CSE and Application resides in the same Java VM and communicate with Java interfaces. Large latency reduces applicable area of oneM2M based solution.

Third problem is the complexity of handling of long name and short name. Even if short name is defined by trying to use initial characters, it is not straight forward to translate them in head.

4 Requirements

This section should be copied from the appropriate RFP(s)

- R0010 – The solution MUST provide means to access outer CSE from application.
- R0011 – The solution MUST provide means to access outer CSE from client CSE.
- R0012 – The solution MUST provide means to select a communication method for application.
- R0013 – The solution MUST provide means to select a communication method for client CSE.
- R0020 – The solution MUST provide means for CSE to accept requests form outer CSE.
- R0020 – The solution MUST provide means for CSE to accept requests form outer application.
- R0030 – The solution MUST provide means to communicate through Java interface between CSE and application that are located in the same OSGi framework.
- R0040 – The solution SHOULD hide differences of communication methods, which are combinations of 4 protocol bindings and 3 serializations (XML, JSON, CBOR).
- R0050 – The solution SHOULD provide developer friendly way for handling short names.
- R0060 – The solution MUST provide asynchronous interface using 'call by value', such as DTO.

5 Technical Solution

First give an architectural overview of the solution so the reader is gently introduced in the solution (Javadoc is not considered gently). What are the different modules? How do the modules relate? How do they interact? Where do they come from? This section should contain a class diagram. Then describe the different modules in detail. This should contain descriptions, Java code, UML class diagrams, state diagrams and interaction diagrams. This section should be sufficient to implement the solution assuming a skilled person.

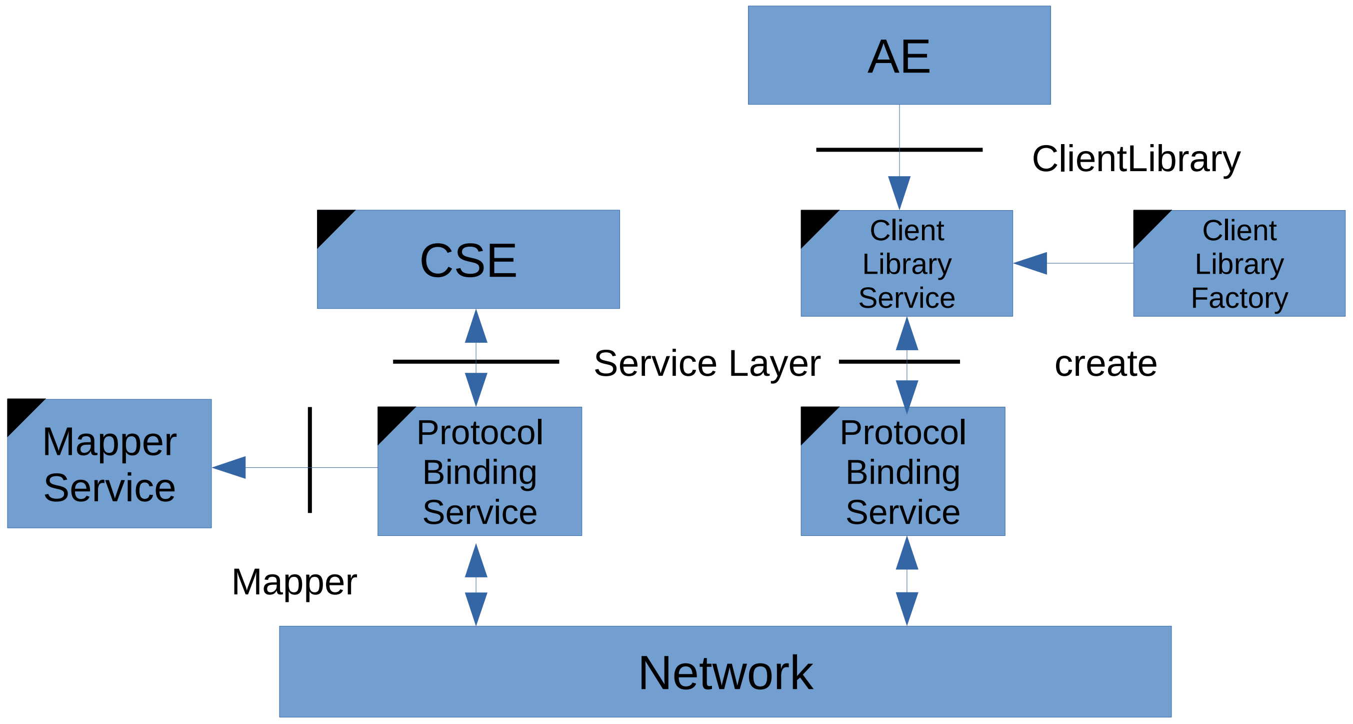
Strictly use the terminology a defined in the Problem Context.

On each level, list the limitations of the solutions and any rationales for design decisions. Almost every decision is a trade off so explain what those trade offs are and why a specific trade off is made.

Address what security mechanisms are implemented and how they should be used.

5.1 Overview for communication through network

Protocol binding service and Mapper service are introduced to handle different protocols and serializations, respectively. CSE communicates the protocol binding service through Service Layer Interface. The interface is protocol and serialization agnostic interface. Protocol binding service uses Mapper service to handle different serializations. For the Application entity, higher level abstraction is provided by Client Library Service through ClientLibrary interface. Since this service is a stateful service, so it will be generated by Client Library Factory. Following figure illustrates overall architecture.



Service Layer Interface is defined as follows. Only method request sends request message and return Promise for the response. Here, Promise enables asynchronous messaging.

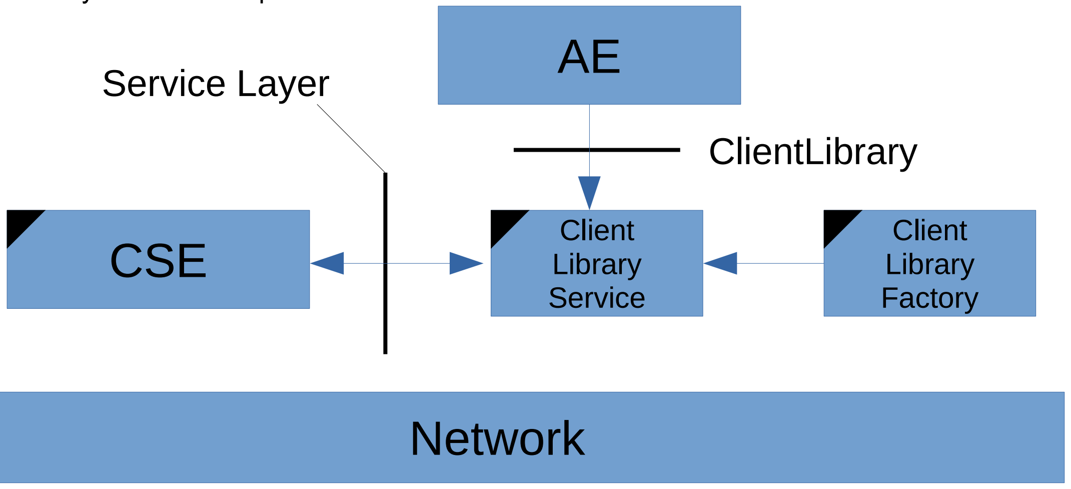
```
package org.osgi.onem2m.servicelayer;

import org.osgi.onem2m.dto.RequestDTO;
import org.osgi.onem2m.dto.ResponseDTO;
import org.osgi.util.promise.Promise;

/**
 * Service Layer Interface, which locates between CSE and Protocol Binding Service.
 */
public interface ServiceLayer {
	/**
	 * send a request.
	 *
	 * @param request request
	 * @return promise for ResponseDTO.
	 */
	Promise<ResponseDTO> request(RequestDTO request);
}
```

On the Service Layer interface, there are bidirectional invocations, that is, CSE sends request to protocol binding service and get response from it, protocol binding service sends request to CSE and get response from it. For the clarification purpose, Service Layer Interface is extended to ProtocolBinding Interface, Cse Interface, and Ae Interface without additional methods.

```
public interface ProtocolBinding extends ServiceLayer {}
public interface Cse extends ServiceLayer {}
public interface Ae extends ServiceLayer {}
```

5.2 Overview for internal communication within single OSGi framework.

For internal communication, Cse Service and ClientLibrary Service communicate directly without inter-mediating ProtocolBinding Service. Following figure depicts overall architecture. Though this type of communication is not clearly defined in oneM2M specification, communicating directly without serializing data between AE and CSE allows shorter latency and less computational resources.



5.3 Service Property for sub-interfaces of Service Layer Interface

Depending on the sub-interfaces of Service Layer Interface, they put different service properties. Table below summarizes the properties on the interfaces.

Interface	property Name	type	explanation
ProtocolBinidng	protocol	String	Supporting protocol. Possible values are “HTTP”, “MQTT”, “CoAP”, or “WebSocket”
	serialization	String	Serialization. Possible values are “XML”, “JSON” or “CBOR”
	Secure	boolean	True, if secure protocol is supported, otherwise false.
	versions	String[]	Supported versions, possible values would be “R1”, “R1.1”, “R2”, “R2A”.
Cse	CSE-ID	String	CSE-ID: ID of CSE
	SP-ID	String	ID of Service Provider
	CSE-type	String	Type of CSE. Possible values are IN, MN, or ASN
	POA	String	URI for point of access
	versions	String[]	Supported versions

Ae	AE-ID-Stem	String	Head part of AE-ID
	APP-ID	String	Application ID
	POA	String	URI for point of access
	versions	String[]	Supported versions

5.4 ClientLibrary

For the Application entity, higher level abstraction is provided by Client Library Service through ClientLibrary interface. ClientLibrary manages Request-ID and AE-ID, so that it need to be stateful service. It will be generated by Client Library Factory depending on using AE bundles. ClientLibrary Interface provides more developer friendly interface that exposes CRUD+N operation on resources.

```
package org.osgi.onem2m.client;

public interface ClientLibrary {
	/**
	 * Create resource
	 *
	 * @param uri URI for the target Resource
	 * @param resource ResourceDTO for creating resource
	 * @return ResouceDTO for created resource
	 */
	public ResourceDTO create(String uri, ResourceDTO resource) throws OneM2MException;

	/**
	 * Retrieve resource
	 *
	 * @param uri URI for the target Resource
	 * @return ResouceDTO for retrieved resource
	 */
	public ResourceDTO retrieve(String uri);

	/**
	 * Retrieve of partial attributes.
	 *
	 * @param uri URI for the target Resource
	 * @param attributes attribute names for retrieving attribute.
	 * @return ResouceDTO for retrieved resource
	 */
	public ResourceDTO retrieve(String uri, List<String> attributes) throws
OneM2MException;

	/**
	 * Update resource
	 *
	 * @param uri URI for resource
	 * @param resource resource data
	 * @throws OneM2MException
```

```
         */
        public ResourceDTO update(String uri, ResourceDTO resource) throws OneM2MException;


        /**
         * Delete resource
         * @param uri URI for resource
         * @throws OneM2MException
         */
        public void delete(String uri) throws OneM2MException;

        /**
         * send notification
         *
         * @param poa
         * @param notification
         */
        public void notify(NotificationDTO notification ) throws OneM2MException;

        ...
}
```

There are some configuration methods. Application must set communication method by calling setProtocol(), setSerialization(), setSecureProtocol() and some communication information by calling setAeId(), setRequestIDHeader(). If the Application need to receive notifications, it must set notification listener by calling setNotificationListener().

```
public interface ClientLibrary {
        ...
        /**
         * Set using protocol. This method must be called once before using actual operation
methods.
         * @param protocol HTTP, CoAP, MQTT, WebSocket
         */
        public void setProtocol(String protocol);

        /**
         * Set using serialization. This method must be called once before using actual
operation methods.
         * @param serialization using serialization. allowed value is one of "XML", "JSON",
"CBOR"
         */
        public void setSerialization(String serialization);

        /**
         * Set security mode for the communication. This method must be called once before
using acutal operation methods.
         * @param protocol
         */
        public void setSecureProtocol(boolean isSecure);
        /**
         * set Application ID. This value is used for registration process of oneM2M.
         * "C" is used for the initial registration. After registration the assigned AEID.
         *
         * @param aeid
         */
        public void setAeId(String aeid);

        /**
         * set header for request ID
         *
         * @param header for request ID
         */
        public void setRequestIDHeader(String format);

        /**
         * set Notification Listener with client library.
         *
         * @param poa
         * @param listner
         */
        public void setNotificationListener(String poa, NotificationListener listener );

        ...
}
```

NOTE: need to add setAppID()?

5.5 Validator Interface

Validator Services implementing Validator interface, can be used for validating OneM2MDTO and its sub classes.

TBD

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

6.1 OneM2MDTO

OneM2MDTO is the base DTO for more concrete DTO related to oneM2M. The DTO holds a Map field and JSON like structure is stored in it.

```
package org.osgi.onem2m.dto;

import java.util.Map;
import org.osgi.dto.DTO;

/**
 * DTO containing oneM2M related data.
 *
 */
public class OneM2MDTO extends DTO {
	/**
	 * map holding JSON like data structure
	 */
	public Map<String,Object> map;
}
```

6.2 RequestDTO

RequestDTO holds a Request Information used for oneM2M communication. This DTO extends OneM2MDTO but has no additional fields. The extension is used for clarification purpose of type.

```
package org.osgi.onem2m.dto;

/**
 * DTO containing oneM2M request primitive.
 */
public class RequestDTO extends OneM2MDTO {
}
```

6.3 ResponseDTO

ResponseDTO holds a Response Information used for oneM2M communication. This DTO extends OneM2MDTO but has no additional fields. The extension is used for clarification purpose of type.

```
package org.osgi.onem2m.dto;

/**
 * DTO containing oneM2M response primitive.
 */
public class ResponseDTO extends OneM2MDTO {
}
```

6.4 ResourceDTO

ResourceDTO holds information for oneM2M resource. Extended attribute called resourceType holds resource type in short name, like “AE”.

```
package org.osgi.onem2m.dto;

/**
 * DTO containing oneM2M resource.
 */
public class ResourceDTO extends OneM2MDTO {
	/**
	 * resource type
	 */
	public String resourceType;
}
```

6.5 NotificationDTO

NotificationDTO holds information for Notification. This DTO extends OneM2MDTO but has no additional fields. The extension is used for clarification purpose of type.

```
package org.osgi.onem2m.dto;

/**
 * DTO for Notification.
 */
public class NotificationDTO extends OneM2MDTO {
}
```

6.6 AttributeDTO

AttributeDTO holds information for attribute. Extended fields, type and attributeName has resource type and name of attribute respectively.

```
package org.osgi.onem2m.dto;

/**
 * DTO holding attribute information.
 */
public class AttributeDTO extends OneM2MDTO {
	/**
	 * ResourceType which this attribute belongs to.
	 */
	public int type;
	/**
	 * Attribute name in short name for this attribute.
	 */
	public String attributeName;
}
```

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

Demo Documentation

17/09/14 19:26

Package Summary		Page
org.osgi.onem2m.client	Package containing interfaces of Client Library for oneM2M.	17
org.osgi.onem2m.dto	Package containing Data Transfer Objects (DTOs).	26
org.osgi.onem2m.mapper	Package contains Mapper interface that convert binary data to OneM2MDTO, vice versa.	34
org.osgi.onem2m.servicelayer	The key Package containing Service Layer API.	36
org.osgi.onem2m.util	Package containing Utility Classes for oneM2M.	41
org.osgi.onem2m.validation	Package containing validation interface for oneM2M data structures.	45

Package org.osgi.onem2m.client

Package containing interfaces of Client Library for oneM2M.

See:

Description

Interface Summary		*Page*
ClientLibrary		19
CllientLibraryFactory		23
NotificationListener	Interface for Notification Listener White Board Pattern にする場合。	23

Exception Summary		*Page*
OneM2MException		25

Package org.osgi.onem2m.client Description

Package containing interfaces of Client Library for oneM2M.

Interface ClientLibrary

org.osgi.onem2m.client

```
public interface ClientLibrary
```

Method Summary		Page
ResourceDTO	**create**(String uri, ResourceDTO resource) Create resource	21
void	**delete**(String uri) Delete resource	22
String	**getAeId**() get AEID.	20
long	**getRequestIDCount**() get Counter for the RequestID.	21
void	**notify**(String poa, NotificationDTO notification) send notification	22
ResourceDTO	**retrieve**(String uri) Retrieve resource	21
ResourceDTO	**retrieve**(String uri, List<String> attributes) Retrieve of partial attributes.	21
void	**setAeId**(String aeid) set Application ID.	20
void	**setNotificationListener**(String poa, NotificationListener listener) set Notification Listener with client library.	22
void	**setProtocol**(String protocol) Set using protocol.	19
void	**setRequestIDCount**(long count) set Counter for the RequestID.	20
void	**setRequestIDHeader**(String format) set header for request ID	20
void	**setSecureProtocol**(boolean isSecure) Set security mode for the communication.	20
void	**setSerialization**(String serialization) Set using serialization.	20
ResourceDTO	**update**(String uri, ResourceDTO resource) Update resource	22

Method Detail

setProtocol

```
void setProtocol(String protocol)
```

Set using protocol. This method must be called once before using actual operation methods.

Parameters:

`protocol` - HTTP, CoAP, MQTT, WebSocket

---

setSerialization

`void` **`setSerialization`**`(String serialization)`

Set using serialization. This method must be called once before using actual operation methods.

Parameters:

`serialization` - using serialization. allowed value is one of "XML", "JSON", "CBOR"

---

setSecureProtocol

`void` **`setSecureProtocol`**`(boolean isSecure)`

Set security mode for the communication. This method must be called once before using acutal operation methods.

---

setAeId

`void` **`setAeId`**`(String aeid)`

set Application ID. This value is used for registration process of oneM2M. "C" is used for the initial registration. After registration the assigned AEID.

---

getAeId

`String` **`getAeId`**`()`

get AEID. This may not differ the parameter used with setAeId() methods. Client Library replace the AEID after oneM2M CSE assigned AEID.

Returns:

Application ID

---

setRequestIDHeader

`void` **`setRequestIDHeader`**`(String format)`

set header for request ID

---

setRequestIDCount

`void` **`setRequestIDCount`**`(long count)` set Counter for the RequestID. Usually it is managed by the client library and no need to touch. In case of the special use case, the counter can be manipulated.

---

getRequestIDCount

```
long getRequestIDCount()
```

get Counter for the RequestID. Usually it is managed by the client library and no need to touch. In case of the special use case, the counter can be manipulated.

Returns:
count

---

create

```
ResourceDTO create(String uri,
                   ResourceDTO resource)
            throws OneM2MException
```

Create resource

Parameters:
`uri` - URI for the target Resource
`resource` - ResourceDTO for creating resource
Returns:
ResourceDTO for created resource
Throws:
`OneM2MException`

---

retrieve

```
ResourceDTO retrieve(String uri)
```

Retrieve resource

Parameters:
`uri` - URI for the target Resource
Returns:
ResourceDTO for retrieved resource

---

retrieve

```
ResourceDTO retrieve(String uri,
                     List<String> attributes)
              throws OneM2MException
```

Retrieve of partial attributes.

Parameters:
`uri` - URI for the target Resource
`attributes` - attribute names for retrieving attribute.

Returns:
ResourceDTO for retrieved resource

Throws:
`OneM2MException`

---

update

```
ResourceDTO update(String uri,
                   ResourceDTO resource)
            throws OneM2MException
```

Update resource

Parameters:
`uri` - URI for resource
`resource` - resource data

Throws:
`OneM2MException`

---

delete

```
void delete(String uri)
     throws OneM2MException
```

Delete resource

Parameters:
`uri` - URI for resource

Throws:
`OneM2MException`

---

notify

```
void notify(String poa,
            NotificationDTO notification)
     throws OneM2MException
```

send notification

Throws:
`OneM2MException`

---

setNotificationListener

```
void setNotificationListener(String poa,
                             NotificationListener listener)
```

set Notification Listener with client library.

Interface CllientLibraryFactory

org.osgi.onem2m.client

All Superinterfaces:

org.osgi.framework.ServiceFactory<ClientLibrary>

```
public interface CllientLibraryFactory
extends org.osgi.framework.ServiceFactory<ClientLibrary>
```

Methods inherited from interface org.osgi.framework.ServiceFactory
`getService, ungetService`

Interface NotificationListener

org.osgi.onem2m.client

```
public interface NotificationListener
```

Interface for Notification Listener White Board Pattern にする場合。 Application wanting to receive notification, MUST register a service implementing this interface, with POA URI in "org.osgi.onem2m.POA" service property. NOTE: This is like White Board Pattern, but there is some limitation. Only the ClientLibrary instance call the method. Client Library must check the registering bundle. (ここが複雑！)

Method Summary		***Page***
void	**notify**(NotificationDTO notification)	24

Method Detail

notify

```
void notify(NotificationDTO notification)
```

Class OneM2MException

org.osgi.onem2m.client

```
java.lang.Object
 └java.lang.Throwable
   └java.lang.Exception
     └org.osgi.onem2m.client.OneM2MException
```

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
ValidationException

```
public class OneM2MException
extends Exception
```

Constructor Summary	Page
OneM2MException(String string)	25

Constructor Detail

OneM2MException

```
public OneM2MException(String string)
```

Package org.osgi.onem2m.dto

Package containing Data Transfer Objects (DTOs).

See:
Description

Class Summary		*Page*
AttributeDTO	DTO holding attribute information.	27
NotificationDTO	DTO for Notification.	29
OneM2MDTO	DTO containing oneM2M related data.	30
RequestDTO	DTO containing oneM2M request primitive.	31
ResourceDTO	DTO containing oneM2M resource.	32
ResponseDTO	DTO containing oneM2M response primitive.	33

Package org.osgi.onem2m.dto Description

Package containing Data Transfer Objects (DTOs).

Class AttributeDTO

org.osgi.onem2m.dto

```
java.lang.Object
 └org.osgi.dto.DTO
   └org.osgi.onem2m.dto.OneM2MDTO
     └org.osgi.onem2m.dto.AttributeDTO
```

```
public class AttributeDTO
extends OneM2MDTO
```

DTO holding attribute information.

Field Summary		Page
String	**attributeName** Attribute name in short name for this attribute.	27
int	**type** ResourceType which this attribute belongs to.	27

Fields inherited from class org.osgi.onem2m.dto.OneM2MDTO
map

Constructor Summary	Page
AttributeDTO()	28

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

type

`public int type`

ResourceType which this attribute belongs to.

attributeName

`public String attributeName`

Attribute name in short name for this attribute.

Constructor Detail

AttributeDTO

```
public AttributeDTO()
```

Class NotificationDTO

org.osgi.onem2m.dto

```
java.lang.Object
 └org.osgi.dto.DTO
   └org.osgi.onem2m.dto.OneM2MDTO
     └org.osgi.onem2m.dto.NotificationDTO
```

```
public class NotificationDTO
extends OneM2MDTO
```

DTO for Notification.

Fields inherited from class org.osgi.onem2m.dto.OneM2MDTO
map

Constructor Summary	Page
`NotificationDTO()`	29

Methods inherited from class org.osgi.dto.DTO
`toString`

Constructor Detail

NotificationDTO

```
public NotificationDTO()
```

Class OneM2MDTO

org.osgi.onem2m.dto

```
java.lang.Object
 └org.osgi.dto.DTO
   └org.osgi.onem2m.dto.OneM2MDTO
```

Direct Known Subclasses:

AttributeDTO, NotificationDTO, RequestDTO, ResourceDTO, ResponseDTO

```
public class OneM2MDTO
extends org.osgi.dto.DTO
```

DTO containing oneM2M related data.

Field Summary		Page
`Map<String,Object>`	**map** map holding JSON like data structure	30

Constructor Summary	Page
OneM2MDTO()	30

Methods inherited from class org.osgi.dto.DTO
`toString`

Field Detail

map

```
public Map<String,Object> map
```

map holding JSON like data structure

Constructor Detail

OneM2MDTO

```
public OneM2MDTO()
```

Class RequestDTO

org.osgi.onem2m.dto

```
java.lang.Object
 └org.osgi.dto.DTO
   └org.osgi.onem2m.dto.OneM2MDTO
     └org.osgi.onem2m.dto.RequestDTO
```

```
public class RequestDTO
extends OneM2MDTO
```

DTO containing oneM2M request primitive.

Fields inherited from class org.osgi.onem2m.dto.OneM2MDTO
map

Constructor Summary	***Page***
`RequestDTO()`	31

Methods inherited from class org.osgi.dto.DTO
`toString`

Constructor Detail

RequestDTO

```
public RequestDTO()
```

Class ResourceDTO

org.osgi.onem2m.dto

```
java.lang.Object
 └org.osgi.dto.DTO
   └org.osgi.onem2m.dto.OneM2MDTO
     └org.osgi.onem2m.dto.ResourceDTO
```

```
public class ResourceDTO
extends OneM2MDTO
```

DTO containing oneM2M resource.

Field Summary		Page
String	**resourceType** resource type	32

Fields inherited from class org.osgi.onem2m.dto.OneM2MDTO
map

Constructor Summary	Page
ResourceDTO()	32

Methods inherited from class org.osgi.dto.DTO
toString

Field Detail

resourceType

```
public String resourceType
```

resource type

Constructor Detail

ResourceDTO

```
public ResourceDTO()
```

Class ResponseDTO

org.osgi.onem2m.dto

```
java.lang.Object
 └org.osgi.dto.DTO
   └org.osgi.onem2m.dto.OneM2MDTO
     └org.osgi.onem2m.dto.ResponseDTO
```

```
public class ResponseDTO
extends OneM2MDTO
```

DTO containing oneM2M response primitive.

Fields inherited from class org.osgi.onem2m.dto.OneM2MDTO
map

Constructor Summary	*Page*
`ResponseDTO()`	33

Methods inherited from class org.osgi.dto.DTO
`toString`

Constructor Detail

ResponseDTO

```
public ResponseDTO()
```

Package org.osgi.onem2m.mapper

Package contains Mapper interface that convert binary data to OneM2MDTO, vice versa.

See:
Description

Interface Summary		*Page*
Mapper	Mapper interface, which convert OneM2MDTO to binary data and vice versa.	35

Package org.osgi.onem2m.mapper Description

Package contains Mapper interface that convert binary data to OneM2MDTO, vice versa.

Interface Mapper

org.osgi.onem2m.mapper

```
public interface Mapper
```

Mapper interface, which convert OneM2MDTO to binary data and vice versa.

Method Summary		Page
OneM2MDTO	**deserialize**(byte[] bb)	35
OneM2MDTO	**deserialize**(ByteBuffer bb)	35
String	**getContentInfo**()	35
ByteBuffer	**serializeToByteBuffer**(OneM2MDTO dto)	35
byte[]	**serializeToBytes**(OneM2MDTO dto)	35

Method Detail

serializeToByteBuffer

```
ByteBuffer serializeToByteBuffer(OneM2MDTO dto)
```

---

serializeToBytes

```
byte[] serializeToBytes(OneM2MDTO dto)
```

---

deserialize

```
OneM2MDTO deserialize(ByteBuffer bb)
```

---

deserialize

```
OneM2MDTO deserialize(byte[] bb)
```

---

getContentInfo

```
String getContentInfo()
```

Package org.osgi.onem2m.servicelayer

The key Package containing Service Layer API.

See:

Description

Interface Summary		*Page*
Ae	Ae, which represents oneM2M AE.	37
Cse	CSEService, which represents oneM2M CSE.	37
ProtocolBinding	Protocol Binding Service, which represent Protocol Binding	38
ServiceLayer	Service Layer Interface, which locates between CSE and Protocol Binding Service.	39

Package org.osgi.onem2m.servicelayer Description

The key Package containing Service Layer API.

Interface Ae

org.osgi.onem2m.servicelayer

All Superinterfaces:
ServiceLayer

```
public interface Ae
extends ServiceLayer
```

Ae, which represents oneM2M AE.

Methods inherited from interface org.osgi.onem2m.servicelayer.ServiceLayer
request

Interface Cse

org.osgi.onem2m.servicelayer

All Superinterfaces:

ServiceLayer

```
public interface Cse
extends ServiceLayer
```

CSEService, which represents oneM2M CSE.

Methods inherited from interface org.osgi.onem2m.servicelayer.ServiceLayer
request

Interface ProtocolBinding

org.osgi.onem2m.servicelayer

All Superinterfaces:

ServiceLayer

```
public interface ProtocolBinding
extends ServiceLayer
```

Protocol Binding Service, which represent Protocol Binding

Methods inherited from interface org.osgi.onem2m.servicelayer.ServiceLayer
request

Interface ServiceLayer

org.osgi.onem2m.servicelayer

All Known Subinterfaces:

Ae, Cse, ProtocolBinding

```
public interface ServiceLayer
```

Service Layer Interface, which locates between CSE and Protocol Binding Service.

Method Summary		*Page*
org.osgi.util.promise.Promise<ResponseDTO>	**request**(RequestDTO request) send a request.	40

Method Detail

request

```
org.osgi.util.promise.Promise<ResponseDTO> request(RequestDTO request)
```

send a request.

Parameters:

`request` - request

Returns:

promise for ResoponseDTO.

Package org.osgi.onem2m.util

Package containing Utility Classes for oneM2M.

See:
Description

Class Summary		***Page***
Util	Utility Class for oneM2M.	42

Package org.osgi.onem2m.util Description

Package containing Utility Classes for oneM2M.

Class Util

org.osgi.onem2m.util

```
java.lang.Object
 └ org.osgi.onem2m.util.Util
```

```
public class Util
extends Object
```

Utility Class for oneM2M.

Constructor Summary	Page
Util()	42

Method Summary		Page
AttributeDTO	**getAttributeS**(ResourceDTO resource, String sname) retrieve AttributeDTO from ResourceDTO using short named Attribute	43
String	**l2s**(String lname) Convert long name to short name.	43
static void	**main**(String[] args) Test purpose main() function.	44
OneM2MDTO	**put**(OneM2MDTO dto, String key, int value) Put data into a map managed in OneM2MDTO.	43
OneM2MDTO	**put**(OneM2MDTO dto, String key, Object value) Put data into a map managed in OneM2MDTO.	43
OneM2MDTO	**put**(OneM2MDTO dto, String key, long value) Put data into a map managed in OneM2MDTO.	44
String	**s2l**(String sname) Convert short name to long name.	42

Constructor Detail

Util

```
public Util()
```

Method Detail

s2l

```
public String s2l(String sname)
```

Convert short name to long name.

Parameters:

`sname` - short name

Returns:
long name

---

l2s

```
public String l2s(String lname)
```

Convert long name to short name.

Parameters:
`lname` - long name
Returns:
short name

---

getAttributeS

```
public AttributeDTO getAttributeS(ResourceDTO resource,
                                  String sname)
```

retrieve AttributeDTO from ResourceDTO using short named Attribute

Parameters:
`resource` - ResourceDTO
`sname` - Attribute name in short name
Returns:
AttributeDTO

---

put

```
public OneM2MDTO put(OneM2MDTO dto,
                     String key,
                     Object value)
```

Put data into a map managed in OneM2MDTO.

Parameters:
`dto` - OneM2MDTO
`key` - key
`value` - value
Returns:
OneMDMDTO specified as the first argument, allowing method chaining.

---

put

```
public OneM2MDTO put(OneM2MDTO dto,
                     String key,
                     int value)
```

Put data into a map managed in OneM2MDTO.

Parameters:
`dto` - OneM2MDTO

`key` - key
`value` - value

Returns:
OneMDMDTO specified as the first argument, allowing method chaining.

---

put

```
public OneM2MDTO put(OneM2MDTO dto,
                     String key,
                     long value)
```

Put data into a map managed in OneM2MDTO.

Parameters:
`dto` - OneM2MDTO
`key` - key
`value` - value

Returns:
OneMDMDTO specified as the first argument, allowing method chaining.

---

main

```
public static void main(String[] args)
```

Test purpose main() function.

Package org.osgi.onem2m.validation

Package containing validation interface for oneM2M data structures.

See:

Description

Interface Summary		Page
RequestValidator	Resource Validator interface.	46
ResourceValidator	Resource Validator interface.	47
ResponseValidator	Response Validator interface.	48

Exception Summary		Page
ValidationException	Validation Exception, which will be thrown in Validation related processing.	49

Package org.osgi.onem2m.validation Description

Package containing validation interface for oneM2M data structures.

Interface RequestValidator

org.osgi.onem2m.validation

```
public interface RequestValidator
```

Resource Validator interface.

Method Summary		*Page*
void	**validate**(RequestDTO req) validate RequestDTO	46

Method Detail

validate

```
void validate(RequestDTO req)
       throws ValidationException
```

validate RequestDTO

Throws:

ValidationException

Interface ResourceValidator

org.osgi.onem2m.validation

`public interface` **`ResourceValidator`**

Resource Validator interface.

Method Summary		***Page***
`void`	**`validate`**`(ResourceDTO resource)` Validate ResourceDTO	47

Method Detail

validate

```
void validate(ResourceDTO resource)
       throws ValidationException
```

Validate ResourceDTO

Parameters:
`resource` - resourceDTO under validation.

Throws:
`ValidationException`

Interface ResponseValidator

org.osgi.onem2m.validation

```
public interface ResponseValidator
```

Response Validator interface.

Method Summary		***Page***
void	**validate**(ResponseDTO response) Response Validator.	48

Method Detail

validate

```
void validate(ResponseDTO response)
       throws ValidationException
```

Response Validator.

Parameters:
`response` - ResoposeDTO under validation

Throws:
ValidationException

Class ValidationException

org.osgi.onem2m.validation

```
java.lang.Object
 └java.lang.Throwable
   └java.lang.Exception
     └org.osgi.onem2m.client.OneM2MException
       └org.osgi.onem2m.validation.ValidationException
```

All Implemented Interfaces:
Serializable

```
public class ValidationException
extends OneM2MException
```

Validation Exception, which will be thrown in Validation related processing.

Constructor Summary	Page
ValidationException(String string)	49

Constructor Detail

ValidationException

```
public ValidationException(String string)
```

8 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

8.1 Representation of DTO

As alternative solution, utilization of generated Java classes by JAXB has been considered, since oneM2M provides well defined XSD for defining data format. With the following aspects, this approach is not applied.

Many classes: Currently 65 XSD files are defined in oneM2M specification and JAXB tool (xjc) generates more than 140 Java classes. Using many classes as interface could make specification more complicated than its nature.

No Uniqueness: Generated classes by xjc are not unique, because it is possible to customize generation processes.

Changeability: Depending on the version of oneM2M, XSD files differ. It is preferable to choose version independent API, as much as possible. oneM2M ensures any data can be converted to JSON and CBOR, so proposed approach can be used with out modification, even if XSD file would be changed.

8.2 White Board pattern for receiving notification by AE

For setNotificationListener() methods on ClientLibrary interface, the alternative using White Board pattern is considered, in which the listener is registered on to the OSGi Service Registry, instead of calling setter method. But the listener should be called from only corresponding ClientLibrary Service instance, so the current design was chosen.

8.3 Non blocking API for ClientLibrary

Currently ClientLibrary is desinged as blocking concept. There was an option to design as non blocking API. Because developers want to create AE using non blocking API, she or he can create directly on top of ServiceLayer API. So Non blocking API for ClientLibrary is not provided.

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

9.1 ProtocolBinding Service with secure protocols

In case that ProtocolBInding Service uses secure protocols, it is expected to handle pre-shared key or certificate, in order to get authorized by the communication peer. Once Ae gets the service, it can use it to communicate. The ProtocolBinding service should be protected, in the environment that accommodates many different bundles from different vendors/providers.

9.2 Using Multiple Certificates with in a single ProtocolBinidng Service

With higher degree of aggregation of bundles from different vendos/providers on a single OSGi framework, it would be beneficial for ProtocolBinding service would handle multiple certificates through API. But it is not addressed yet, so far. It might be out of scope of this RFC.

10 Document Support

10.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. oneM2M TS-0001 Functional Architecture, http://onem2m.org/images/files/deliverables/Release2/TS-

0001-%20Functional_Architecture-V2_10_0.pdf

[3]. oneM2M TS-0004 Service Layer Core Protocol, http://onem2m.org/images/files/deliverables/Release2/TS-0004_Service_Layer_Core_Protocol_V2_7_1.zip

[4]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0 (NOTE:Is this needed?)

Add references simply by adding new items. You can then cross-refer to them by chosing <Insert><Cross Reference><Numbered Item> and then selecting the paragraph. ***STATIC REFERENCES (I.E. BODGED) ARE NOT ACCEPTABLE, SOMEONE WILL HAVE TO UPDATE THEM LATER, SO DO IT PROPERLY NOW.***

10.2 Author's Address

Name	Hiroyuki Maeomichi
Company	NTT
Address	Midorimachi 3-9-11, Musashino, Tokyo, Japan
Voice	+81 422 59 4072
e-mail	maeomichi.hiroyuki@lab.ntt.co.jp

10.3 Acronyms and Abbreviations

CSE: Common Services Entity

AE: Application Entity

CBOR: Concise Binary Object Representation

10.4 End of Document