RESTful Network API for <Functional Area>

Draft Version x.y – dd Mmm 2015

Specification Provider

PROVIDER-TS-REST_NetAPI_<FuncArea>-Vx_x-2015mmdd-D

<< In the flow text in this template, yellow marks are used for placeholders that need to be replaced by real-world text, and cyan marks are used for explanations that need to be deleted in the final document.

This is a template, intended to be used only for RESTful Network application programming interface (API) specifications.

Note regarding the **font style** to be used for Technical Specifications:

- for main body text use Times New Roman font size 10,,
- in tables, generally Arial font size 10 should be used, however if necessary to minimize wrapping other fonts of the Arial family such as Arial Narrow font size 10 or Arial font size 9 can be considered also.
- for XML/JSON examples use 'listing' paragraph style

Note regarding the **references** used in Technical specifications:

- references to external documents SHOULD NOT be in hyperlink style,
- for internal references to particular sections, cross-references SHOULD be used.

Delete this comment. >>

Contents

Figures

Tables

1. Scope

<< Alternative 1: This is a suggestion for the introduction if there is a baseline specification. Use either alternative 1 or alternative 2.>>

This specification defines a RESTful API for [Functional Area] using HTTP protocol bindings, based on [the baseline spec].

<< Alternative 2: This is a suggestion for the introduction if there is **no** baseline specification. Use either alternative 1 or alternative 2.>>

This specification defines a RESTful API for [Functional Area] using HTTP protocol bindings.

2. References

2.1 Normative References

[BASELINE_REF]	Baseline specification, if applicable, otherwise delete this reference.
[Autho4API_10]	"Authorization Framework for Network APIs", Open Mobile Alliance™, OMA-ER-Autho4API-V1_0, URL: http://www.openmobilealliance.org/
[REST_NetAPI_ACR]	"RESTful Network API for Anonymous Customer Reference Management ", Open Mobile Alliance™, OMA-TS-REST_NetAPI_ACR-V1_0, URL: http://www.openmobilealliance.org/
[REST_NetAPI_Common]	"Common definitions for RESTful Network APIs", Open Mobile Alliance™, OMA-TS-REST_NetAPI_Common-V1_0, URL: http://www.openmobilealliance.org/
[REST_NetAPI_NotificationChannel]	Include if the use of Notification Channel is supported, otherwise delete this reference. "RESTful Network API for Notification Channel", Open Mobile Alliance™, OMA-TS-REST_NetAPI_NotificationChannel-V1_0, URL: http://www.openmobilealliance.org/
[REST_SUP_FUNCAREA]	"XML schema for the RESTful Network API for [Functional Area]", Open Mobile Alliance™, OMA-SUP-XSD_rest_netapi_funcarea-V1_0, URL: http://www.openmobilealliance.org/
[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
[RFC3966]	"The tel URI for Telephone Numbers", H.Schulzrinne, December 2004, URL: http://www.ietf.org/rfc/rfc3966.txt
[RFC3986]	"Uniform Resource Identifier (URI): Generic Syntax", R. Fielding et. al, January 2005, URL:http://www.ietf.org/rfc/rfc3986.txt
[RFC7159]	"The JavaScript Object Notation (JSON) Data Interchange Format", T. Bray, Ed., March 2014, URL http://tools.ietf.org/rfc/rfc7159.txt
[XMLSchema1]	W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, URL: http://www.w3.org/TR/xmlschema11-1/
[XMLSchema2]	W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, URL: http://www.w3.org/TR/xmlschema11-2/
<< Add/Remove reference rows to this table as needed - DELETE This Row >>	

2.2 Informative References

[OMADICT]	"Dictionary for OMA Specifications", Version 2.9, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_9, URL:http://www.openmobilealliance.org/
[REST_WP]	"Guidelines for RESTful Network APIs", Open Mobile Alliance™, OMA-WP-Guidelines_for_RESTful_Network_APIs, URL:http://www.openmobilealliance.org/
<< Add/Remove reference rows to this table as needed - DELETE This Row >>	

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]. If the use of Notification Channel and/or Light-weight Resources are supported, include also the definitions below, otherwise delete those that are not applicable.

Client-side Notification URL	An HTTP URL exposed by a client, on which it is capable of receiving notifications and that can be used by the client when subscribing to notifications.
Heavy-weight Resource	A resource which is identified by a resource URL which is then used by HTTP methods to operate on the entire data structure representing the resource. Include this definition if Light-weight Resources are supported, otherwise delete it..
Light-weight Resource	A subordinate resource of a Heavy-weight Resource which is identified by its own resource URL which is then used by HTTP methods to operate on a part of the data structure representing the Heavy-weight Resource. The Light-weight Resource URL can be seen as an extension of the Heavy-weight Resource URL. There could be several levels of Light-weight Resources below the ancestor Heavy-weight Resource, depending on the data structure. Include this definition if Light-weight Resources are supported, otherwise delete it.
Long Polling	A variation of the traditional polling technique, where the server does not reply 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 an information push from a server to a client.
Notification Channel	A channel created on the request of the client and used to deliver notifications from a server to a client. The channel is represented as a resource and provides means for the server to post notifications and for the client to receive them via specified delivery mechanisms. For example in the case of Long Polling the channel resource is defined by a pair of URLs. One of the URLs is used by the client as a call-back URL when subscribing for notifications. The other URL is used by the client to retrieve notifications from the Notification Server.
Notification Server	A server that is capable of creating and maintaining Notification Channels.
Server-side Notification URL	An HTTP URL exposed by a Notification Server, that identifies a Notification Channel and that can be used by a client when subscribing to notifications.

3.3 Abbreviations

ACR	Anonymous Customer Reference
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OMA	Open Mobile Alliance
REST	REpresentational State Transfer

SCR	Static Conformance Requirements
SIP	Session Initiation Protocol
TS	Technical Specification
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WP	White Paper
XML	eXtensible Markup Language
XSD	XML Schema Definition
<< Add/Remove abbreviation rows to this table as needed - DELETE This Row>>	

4. Introduction

<< Alternative 1: This is a suggestion for the introduction if there is a baseline specification. Use either alternative 1 or alternative 2. >>

The Technical Specification of the RESTful Network API for [Functional Area] contains HTTP protocol bindings for the [Baseline specification] [BASELINE_REF] specification, using the REST architectural style. The specification provides resource definitions, the HTTP verbs applicable for each of these resources, and the element data structures, as well as support material including flow diagrams and examples using the various supported message body formats (i.e. XML, JSON, and application/x-www-form-urlencoded).

<< Alternative 2: This is a suggestion for the introduction if there is **no** baseline specification. Use either alternative 1 or alternative 2. >>

The Technical Specification of the RESTful Network API for [Functional Area] contains HTTP protocol bindings for [Functionality], using the REST architectural style. The specification provides resource definitions, the HTTP verbs applicable for each of these resources, and the element data structures, as well as support material including flow diagrams and examples using the various supported message body formats (i.e. XML, JSON, and application/x-www-form-urlencoded)..

4.1 Version 1.0

Version 1.0 of this specification supports the following operations:

- One
- Two

<< Include a list of supported operations >>

In addition this specification provides:

- Support for scope values used with authorization framework defined in [Autho4API_10]
- Support for Anonymous Customer Reference (ACR) as an end user identifier
- Support for “acr:auth” as a reserved keyword in an ACR

5. [Functional Area] API definition

This section is organized to support a comprehensive understanding of the [Functional Area] API design. It specifies the definition of all resources, definition of all data structures, and definitions of all operations permitted on the specified resources.

« Include a description that is specific for this Functional Area TS »

Common data types, naming conventions, fault definitions and namespaces are defined in [REST_NetAPI_Common].

The remainder of this document is structured as follows:

Section 5 starts with a diagram representing the resources hierarchy followed by a table listing all the resources (and their URL) used by this API, along with the data structure and the supported HTTP verbs (section 5.1). What follows are the data structures (section 5.2). A sample of typical use cases is included in section 5.3, described as high level flow diagrams.

Section 6 contains detailed specification for each of the resources. Each such subsection defines the resource, the request URL variables that are common for all HTTP methods, and the supported HTTP verbs. For each supported HTTP verb, a description of the functionality is provided, along with an example of a request and an example of a response. For each unsupported HTTP verb, the returned HTTP error status is specified, as well as what should be returned in the Allow header.

All examples in section 6 use XML as the format for the message body. JSON examples are provided in Appendix B.

Section 7 contains fault definition details such as Service Exceptions and Policy Exceptions.

Appendix E lists the [Baseline specification] equivalent operation for each supported REST resource and method combination, where applicable. The wording if there is no baseline spec is as follows:

"Appendix E provides the operations mapping to a pre-existing baseline specification, where applicable.".

Appendix F provides a list of all Light-weight Resources, where applicable.

Appendix G defines authorization aspects to control access to the resources defined in this specification.

Note: Throughout this document client and application can be used interchangeably.

5.1 Resources Summary

This section summarizes all the resources used by the RESTful Network API for [Functional Area].

The "apiVersion" URL variable SHALL have the value [insert value, such as "v1"] to indicate that the API corresponds to this version of the specification. See [REST_NetAPI_Common] which specifies the semantics of this variable.

« Include a resource structure diagram

Use curly brackets around a path component if the text represents a placeholder that will be substituted by an actual value in a URL instance.

Do not use curly brackets around a path component if the text will appear verbatim in a URL instance.

Use text without mark-up for parts of the resource tree that **do not** have associated HTTP methods.

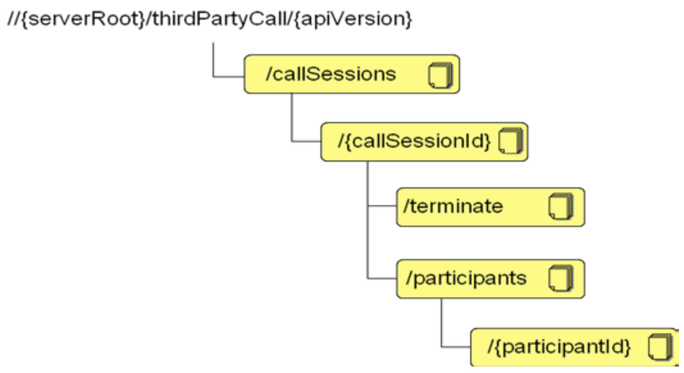
Use a box with a document icon for resources that **do** have associated HTTP methods.

Use a diamond-shaped box with a document icon as a placeholder for one or more Light-weight Resources

Editable PPT versions of the figures are provided below, as editing the embedded figure is problematic

example-structure.zip example-structure-with-LW.zip »

« Alternative 1: Resource structure diagram without Light-weight Resources. Use either alternative 1 or alternative 2. »



« Alternative 2: Resource structure diagram with Light-weight Resources. Use either alternative 1 or alternative 2. »

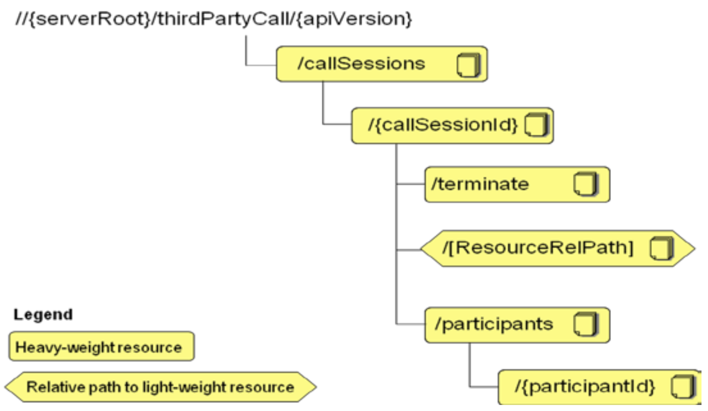


Figure 1 Resource structure defined by this specification

The following tables give a detailed overview of the resources defined in this specification, the data type of their representation and the allowed HTTP methods.

Note: this part of the TS uses a landscape layout, started and terminated by a section break.

<<**Naming conventions for resources**

Names will start with a letter and be mixed case, with the leading letter of each but the first word capitalized. Words will not be separated by white space, underscore, hyphen or other non-letter character.

For names consisting of concatenated words, all subsequent words start with a capital. For example, "concatenatedWord". If a lowercase name starts with an abbreviation, all characters of the abbreviation are de-capitalized, e.g. "smsService".

Path components of resource names are mixed case, with the leading letter lowercase. The leading path component which identifies the API (e.g. thirdpartycall) is all lowercase, and is aligned with the namespace name of the related XML schema.

Note that deviations from this the naming convention SHOULD be the exception and thoroughly justified, e.g. in case of re-use of existing resource structures.>>

Purpose: [Description of the purpose of this (set of) resource(s)]

Resource	**URL** **Base URL: http://{serverRoot}/Functional Area/{apiVersion}**	**Data Structures**	**HTTP verbs**			
			GET	**PUT**	**POST**	**DELETE**
[Description of the resource. Will be repeated in the headlines in section 6.x]	[URL for the resource]	[Data structure(s)] The above is used if there is only one applicable data structure for all HTTP methods. In case there are different data structures applicable, use the following schema instead. If applicable, also mention ResourceReference. Data structure (used for METHOD) common:ResourceReference (OPTIONAL alternative for	[Description of the operation or "no"]	[Description of the operation or "no"]	[Description of the operation or "no"]	[Description of the operation or "no"]

Resource	**URL** **Base URL: http://{serverRoot}/Functional Area/{apiVersion}**	**Data Structures**	**HTTP verbs**			
			GET	**PUT**	**POST**	**DELETE**
		POST response)				
<< Example below - DELETE this and following Row>>						
All Participants of a Call Session	callSessions/{callSessionId}/participants	CallParticipantList (used for GET) CallParticipantInformation (used for POST) common:ResourceReference (OPTIONAL alternative for POST response)	Get a list of participants of a call session	no	Add participant to call session	no
<< Add/Remove rows to this table as needed - DELETE This Row>>						

<< Include separate tables for each purpose>>

5.2 Data Types

5.2.1 XML Namespaces

The XML namespace for the [Functional Area] data types is:

urn: [specificationProvider]:xml:rest:[funcarea]:1

The 'xsd' namespace prefix is used in the present document to refer to the XML Schema data types defined in XML Schema [XMLSchema1, XMLSchema2]. The 'common' namespace prefix is used in the present document to refer to the data types defined in [REST_NetAPI_Common] ***(delete if not used)***. The use of namespace prefixes such as 'xsd' is not semantically significant.

The XML schema for the data structures defined in the section below is given in [REST_SUP_FUNCAREA].

5.2.2 Structures

<<Naming conventions for structures

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:

- Type names start with an uppercase letter
- Element and attribute names in types start with a lowercase letter

Words will not be separated by white space, underscore, hyphen or other non-letter character. For names consisting of concatenated words, all subsequent words start with a capital. For example, "concatenatedWord". If a lowercase name starts with an abbreviation, all characters of the abbreviation are de-capitalized, e.g. "smsService".

In all RESTful API TSs, sections 5.2.x where it is an optional element in a data structure:

resourceURL	xsd:anyURI	Ye s	Self referring URL. The resourceURL SHALL NOT be included in POST requests by the client, but MUST be included in POST requests representing notifications by the server to the client, when a complete representation of the resource is embedded in the notification. The resourceURL MUST also be included in responses to any HTTP method that returns an entity body, and in PUT requests.

Note that deviations from this the naming convention SHOULD be the exception and thoroughly justified, e.g. in case of re-use of existing data types.>>

<< Intro in case the document ***does not*** use the concept of Light-weight Resources. Pick one alternative and DELETE this comment >>

The subsections of this section define the data structures used in the [Functional Area] API.

Some of the structures can be instantiated as so-called root elements.

For structures that contain elements which describe a user identifier, the statements in section 6 regarding 'tel', 'sip' and 'acr' URI schemes apply.

<< Intro in case the document ***does*** use the concept of Light-weight Resources. Pick one alternative and DELETE this comment >>

The subsections of this section define the data structures used in the [Functional Area] API.

Some of the structures can be instantiated as so-called root elements, i.e. they define the type of a representation of a so-called Heavy-weight Resource.

The column [ResourceRelPath] in the tables below, if used, includes relative resource paths for Light-weight Resource URLs that are used to access individual elements in the data structure (so-called Light-weight Resources). A string from this column needs to be appended to the corresponding Heavy-weight Resource URL in order to create Light-weight Resource URL for that particular element in the data structure. “Not applicable” means that individual access to that element is not supported. The root element and data type of the resource associated with the [ResourceRelPath] are defined by the Element and Type columns in the row that defines the [ResourceRelPath].

5.2.2.1 Type: [Type Name (without Light-weight Resources)]

<< This defines the format of the subsections in case the type ***does not*** use the concept of Light-weight Resources.

In case the type includes attribute definition(s), then the name of the first column in the table below should reflect “Element/Attribute”. In addition, in the “Description” column for an attribute it should be stated that it is defined in XML as an attribute. DELETE this comment >>

[Brief description of the type]

Element	**Type**	**Optional**	**Description**
[Element Name]	[Type]	[Yes/No/ Choice]	[Description of the Element]
<< Add/Remove rows to this table as needed - DELETE This Row>>			

<< In case of a root element, include the following text below the table:

A root element named [typeName] of type [TypeName] is allowed in request and/or response bodies.

Or

A root element named [typeName] of type [TypeName] is allowed in notification request bodies.

5.2.2.2 Type: [Type Name (with Light-weight Resources)]

<< This defines the format of the subsections in case the type ***does*** use the concept of Light-weight Resources.

In case the type includes attribute definition(s), then the name of the first column in the table below should reflect "Element/Attribute". In addition, in the "Description" column for an attribute it should be stated that it is defined in XML as an attribute. DELETE this comment >>

[Brief description of the type]

Element	Type	Optional	[ResourceRelPath]	Description
[Element Name]	[Type]	[Yes/No/ Choice]	[relative path of Light-weight Resource or "Not applicable"]	[Description of the Element]
<< Add/Remove rows to this table as needed - DELETE This Row>>				

<< In case of a root element, include the following text below the table:

A root element named [typeName] of type [TypeName] is allowed in request and/or response bodies. If the element is used only in response bodies then the statement should reflect "allowed in response bodies"

Or

A root element named [typeName] of type [TypeName] is allowed in notification request bodies.

Please refer to section 5.2.2 for an explanation of the column [ResourceRelPath].

5.2.3 Enumerations

<<**Naming conventions for enumerations**

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:

- Enumeration type names start with an uppercase letter
- Enumeration value names in types start with an uppercase letter

Words will not be separated by white space, underscore, hyphen or other non-letter character. For names consisting of concatenated words, all subsequent words start with a capital. For example, "ConcatenatedWord". If an uppercase name includes an abbreviation, all characters of the abbreviation are capitalized, e.g. "SMSService", "VoiceXML".

Note that deviations from this the naming convention SHOULD be the exception and thoroughly justified, e.g. in case of re-use of existing data types.>>

The subsections of this section define the enumerations used in the [Functional Area] API.

5.2.3.1 Enumeration: [Enumeration Name]

Enumeration	Description

[Enumeration Value Name]	[Description of the enumeration value]
<< Add/Remove rows to this table as needed - DELETE This Row>>	

5.2.4 Values of the Link "rel" attribute

The "rel" attribute of the Link element is a free string set by the server implementation, to indicate a relationship between the current resource and an external resource. The following are possible strings (list is non-exhaustive, and can be extended):

- One
- Two

<< Include a bullet list with possible "rel" string values >>

These values indicate the kind of resource that the link points to.

5.3 Sequence Diagrams

The following subsections describe the resources, methods and steps involved in typical scenarios.

If the flows include notifications to the client that could be delivered either by POST or through the use of Notification Channel then include this paragraph, otherwise delete it.

In a sequence diagram, a step which involves delivering a notification is labeled with "POST or NOTIFY", where "POST" refers to delivery via the HTTP POST method, and "NOTIFY" refers to delivery using the Notification Channel [REST_NetAPI_NotificationChannel].

5.3.1 [Title of flow scenario]

This figure below shows a scenario for [description of scenario].

If the flow includes a subscription for notifications step, and if the use of Notification Channel is supported, include/adapt this paragraph, otherwise delete it. If there are more scenarios for subscriptions for notifications, in order to avoid repetition this paragraph can be placed one level above (under 5.3) instead.The notification URL passed by the client during the subscription step can be a Client-side Notification URL, or a Server-side Notification URL. Refer to [REST_NetAPI_NotificationChannel] for sequence flows illustrating the creation of a Notification Channel and obtaining a Server-side Notification URL on the server-side, and its use by the client via Long Polling.

The resources:

- To [description of operation], [create/read/update/delete] resource under [resource URL]
- To [description of operation], [create/read/update/delete] resource under [resource URL]

<< Include a flow diagram, and add a figure caption

Use solid lines for requests

Use dotted lines for responses

Use numbers if you want to reference in the text

If multiple servers are involved, name them (e.g. Foo Server, Bar Server), otherwise do not name the server

An editable PPT versions of the figure is provided below, as editing the embedded figure is problematic

example-flow.zip >>

<< Example 1: Signalling flow which includes neither a subscription for notifications nor notifications to the application. Delete this comment.>>

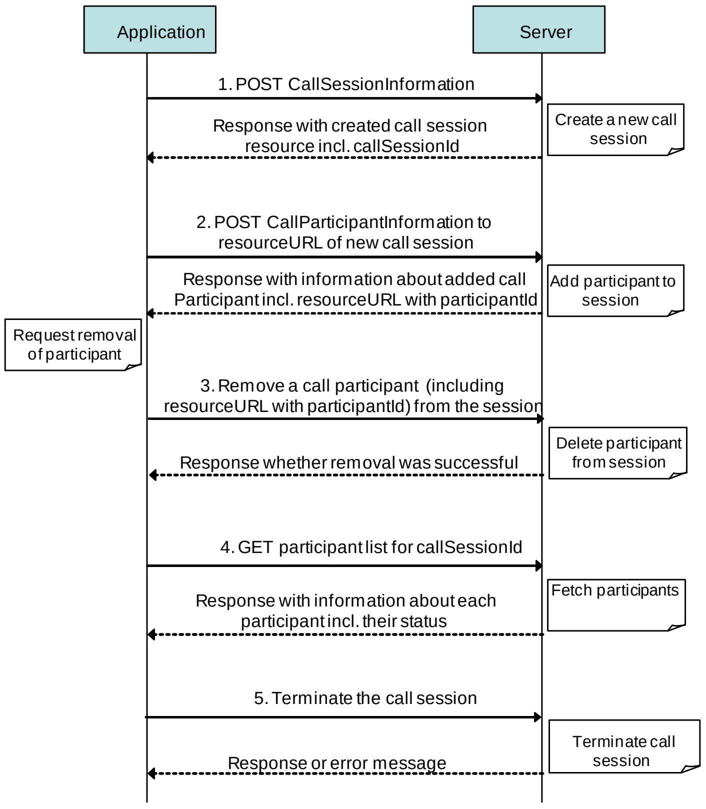

Application
Server
1. POST CallSessionInformation
Response with created call session resource incl. callSessionId
Create a new call session
2. POST CallParticipantInformation to resourceURL of new call session
Response with information about added call Participant incl. resourceURL with participantId
Add participant to session
Request removal of participant
3. Remove a call participant (including resourceURL with participantId) from the session
Response whether removal was successful
Delete participant from session
4. GET participant list for callSessionId
Response with information about each participant incl. their status
Fetch participants
5. Terminate the call session
Response or error message
Terminate call session

Figure 2 [Caption of this flow]

Outline of the flows:

1. [High-level description of 1 or more steps in the flow diagram]
    a) Alternative 1
    b) Alternative 2
2. [High-level description of 1 or more steps in the flow diagram]

3. [High-level description of 1 or more steps in the flow diagram]

<< Example 2: Signalling flow which includes a subscription for notifications and notifications to the application. The notifications to the application can be delivered either by POST or through the use of Notification Channel, which is indicated by "POST or NOTIFY". Delete this comment. >>

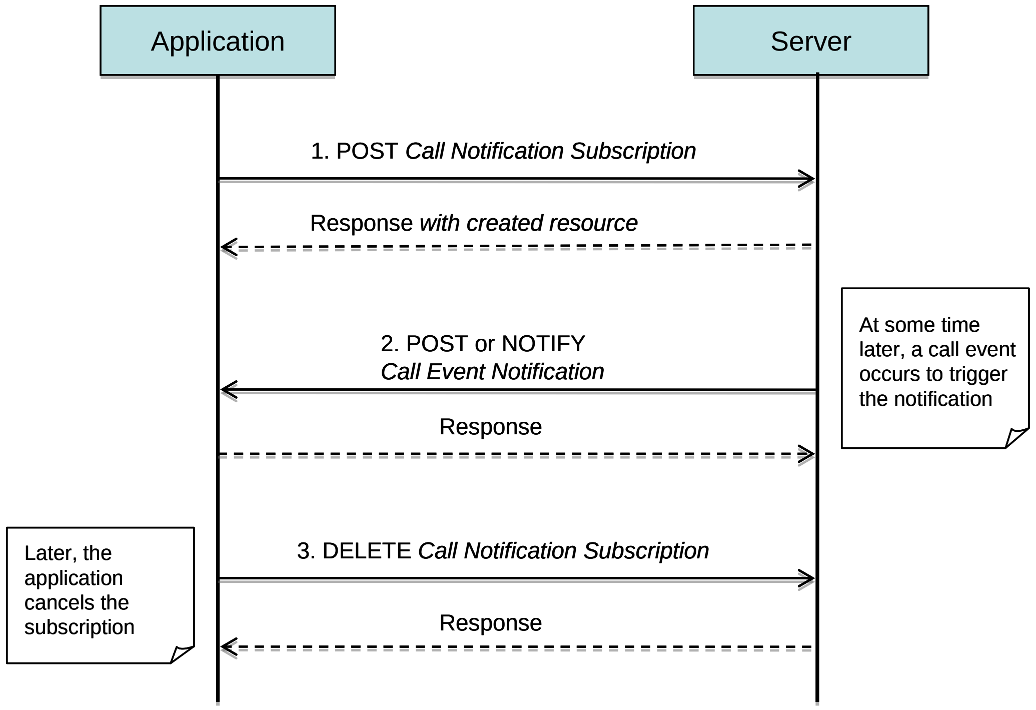

Application
Server
1. POST *Call Notification Subscription*
Response *with created resource*
2. POST or NOTIFY *Call Event Notification*
At some time later, a call event occurs to trigger the notification
Response
3. DELETE *Call Notification Subscription*
Later, the application cancels the subscription
Response

Figure 3 [Caption of this flow]

Outline of the flows:

1. [High-level description of 1 or more steps in the flow diagram]
   a) Alternative 1
   b) Alternative 2
2. [High-level description of 1 or more steps in the flow diagram]. If the step relates to a notification to the application either with POST or NOTIFY, after the high-level description of the action with POST include/adapt the following sentence; otherwise, if POST is supported only, delet it. Alternatively, the application obtains the notifications using a Notification Channel [REST_NetAPI_NotificationChannel].
3. [High-level description of 1 or more steps in the flow diagram]

6. Detailed specification of the resources

The following applies to all resources defined in this specification regardless of the representation format (i.e. XML, JSON, application/x-www-form-urlencoded):

- Reserved characters in URL variables (parts of a URL denoted below by a name in curly brackets) MUST be percent-encoded according to [RFC3986]. Note that this always applies, no matter whether the URL is used as a Request URL or inside the representation of a resource (such as in “resourceURL” and “link” elements).
- If a user identifier (e.g. address, participantAddress, etc.) of type anyURI is in the form of an MSISDN, it MUST be defined as a global number according to [RFC3966] (e.g. tel:+19585550100). The use of characters other than digits and the leading “+” sign SHOULD be avoided in order to ensure uniqueness of the resource URL. This applies regardless of whether the user identifier appears in a URL variable or in a parameter in the body of an HTTP message.
- If an equipment identifier of type anyURI is in the form of a SIP URI, it MUST be defined according to [RFC3261].
- If a user identifier (e.g. address, userId, etc) of type anyURI is in the form of an Anonymous Customer Reference (ACR), it MUST be defined according to [REST_NetAPI_ACR], i.e. it MUST include the protocol prefix 'acr:' followed by the ACR.
    - The ACR ‘auth’ is a supported reserved keyword, and MUST NOT be assigned as an ACR to any particular end user. See E.1.2 for details regarding the use of this reserved keyword.
- For requests and responses that have a body, the following applies: in the requests received, the server SHALL support JSON and XML encoding of the parameters in the body, and MAY support application/x-www-form-urlencoded parameters in the body. The Server SHALL return either JSON or XML encoded parameters in the response body, according to the result of the content type negotiation as specified in [REST_NetAPI_Common]. In notifications to the Client, the server SHALL use either XML or JSON encoding, depending on which format the client has specified in the related subscription. The generation and handling of the JSON representations SHALL follow the rules for JSON encoding in HTTP Requests/Responses as specified in [REST_NetAPI_Common].

6.1 Resource: [Description of the resource]

<< Description of the resource in the title heading should match the description of the resource from the first column of the purpose table in section 5.1 >>

The resource used is:

[resource URL]

[without Light-weight Resources usually http://{serverRoot}/*funcarea*/{apiVersion}/...]

[with Light-weight Resources usually http://{serverRoot}/*funcarea*/{apiVersion}/.../[ResourceRelPath]]

This resource is used for [descriptive explanation of the resource].

If the resource is on the server side and supports creating a subscription for notifications, and if the use of Notification Channel is supported, include/adapt this paragraph, otherwise delete it. This resource can be used in conjunction with a Client-side Notification URL, or in conjunction with a Server-side Notification URL. In this latter case, the application MUST first create a Notification Channel (see [REST_NetAPI_NotificationChannel]) before creating a subscription.

Alternatively, if the resource is a notification resource to which the server provides notifications based on a previously created subscription, and if the use of Notification Channel is supported, include/adapt this paragraph and the following Note, otherwise delete them. This resource is a callback URL provided by the client for notification about FOO. The RESTful [Functional Area] API does not make any assumption about the structure of this URL. If this URL is a Client-side Notification URL, the server will POST notifications directly to it. If this URL is a Server-side Notification URL, the server uses it to determine the address of the Notification Server to which the notifications will subsequently be POSTed. The way the server determines the address of the Notification Server is out of scope of this specification.

Note: In the case when the client has set up a Notification Channel in order to use Long Polling to obtain the notifications, in order to retrieve the notifications, the client needs to use the Long Polling mechanism described in [REST_NetAPI_NotificationChannel], instead of the mechanism described below in section 6.x.y. 6.x.y to be replaced by the reference to the section that describes the actual POST method on THIS resource (e.g. in this case 6.1.5)

6.1.1 Request URL variables

The following request URL variables are common for all HTTP methods:

Name	Description
serverRoot	Server base url: hostname+port+base path. Port and base path are OPTIONAL. Example: example.com/exampleAPI
apiVersion	Version of the API client wants to use. The value of this variable is defined in section 5.1
[ResourceRelPath]	Relative resource path for a Light-weight Resource, consisting of a relative path down to an element in the data structure. For more information about the applicable values (strings) for this variable see [section number entitled “Light-weight relative resource paths” applicable for the current resource]. [This row is only present in case the resource has Light-weight child resources]
<< Add/Remove rows to this table as needed - DELETE This Row>>	

See section 6 for a statement on the escaping of reserved characters in URL variables.

<< Light-weight Resource relative paths. This subsection is only applicable if the resource allows accessing individual sub-trees in the data structure using the Light-weight Resource mechanism (i.e. [ResourceRelPath is part of the resourceURL]>>

6.1.1.1 Light-weight Resource relative paths

The following table describes the types of Light-weight Resources that can be accessed by using this resource, applicable methods, and links to data structures that contain values (strings) for those relative resource paths.

Light-weight Resource type	Method supported	Description
[Description of the type]	[list of HTTP methods, POST not allowed]	[Description and reference to the allowed values]
<< Example - DELETE This and the following Row >>		
Person attributes	GET, PUT, DELETE	Enables access to a single presence attribute related to a person. See data structure 5.2.2.4 for possible values for the Light-weight relative resource path.

6.1.2 Response Codes and Error Handling

For HTTP response codes, see [REST_NetAPI_Common].

For Policy Exception and Service Exception fault codes applicable to [Functional Area], see section 7.

6.1.3 GET

<< This is a blueprint for GET in case it is **not** a valid operation>>

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: [PUT/POST/DELETE]' field in the response as per section 14.7 of [RFC 2616].

<< This is a blueprint for GET in case it is a valid operation>>

This operation is used for [description of operation].

<< The following table is optional and is used only if query parameters are supported in request URL for GET, otherwise it needs to be deleted >>

Supported parameters in the query string of the Request URL are:

Name	Type/Values	Optional	Description
[Parameter name]	[Type/Values]	[Yes/No]	[Parameter description]
<< Add/Remove rows to this table as needed - DELETE This Row>>			

When using query parameters the following conventions apply: >>

- Query parameters are appended to the resource URL starting with a question mark "?" character and then followed by query parameter name – value pairs.
- Multiple query parameter name-value pairs are separated by an ampersand "&" character. Example: ?par1=par1Val&par2=par2Val&..
- Multiple values for the same query parameters are specified as a list of name-value pairs using the same name, separated by an ampersand "&" character. Example: ?par1=par1Val1&par1=par1Val2&...

6.1.3.1 Example 1: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove the sequence number from the title heading >>

<< **Prior to final emission of the specification with XML examples it is suggested to perform an XML validation of the example**.

The examples must use real-world values.

The following conventions apply:

- {serverRoot} http://example.com/exampleAPI also to be updated in the tables where {serverRoot} is defined in section 6.x.
- {version} In our case this is v1 matching the TS version.

- {userId} E-mail names: mailto:alice@example.com mailto:bob@example.com or phone numbers: tel:+1-555-555-0100 to tel:+1-555-555-0199. In fact, only 555-0100 through 555-0199 are now specifically reserved for fictional use, with the other numbers having been released for actual assignment.
- {deviceAddress}, {senderAddress} Typically a phone number
- {equipmentId} Typically a manufacturer type name or serial number
- {memberListId} Typically a group name, "friend", "list123"
- {contactId} Typically a person's name, "bob"
- {memberId} Typically a phone number or e-mail address or SIP URI
- {subscriptionId} Typically a number or a sequence of digits and letters, "sub123"
- {messageId} Typically a number or a sequence of digits and letters, "msg123"
- {interactionId} Typically a number or a sequence of digits and letters, "int123"
- {registrationId} Typically a number or a sequence of digits and letters, "reg123"
- {requestId} Typically a number or a sequence of digits and letters, "req123"
- {ruleId} Typically a number or a sequence of digits and letters, "rule123">>

6.1.3.1.1 Request

[HTTP headers]

[XML request (if applicable), starting with <?xml]

6.1.3.1.2 Response

[HTTP headers]

[XML response (if applicable), starting with <?xml]

6.1.3.2 Example 2: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove this section >>

6.1.3.2.1 Request

[HTTP headers]

[XML request (if applicable), starting with <?xml]

6.1.3.2.2 Response

[HTTP headers]

[XML response (if applicable), starting with <?xml]

6.1.4 PUT

<< This is a blueprint for PUT in case it is **not** a valid operation>>

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the 'Allow: [GET/POST/DELETE]' field in the response as per section 14.7 of [RFC 2616].

<< This is a blueprint for PUT in case it is a valid operation>>

This operation is used for [description of operation].

6.1.4.1 Example 1: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove the sequence number from the title heading >>

6.1.4.1.1 Request

[HTTP headers]

[XML request (if applicable), starting with <?xml]

6.1.4.1.2 Response

[HTTP headers]

[XML response (if applicable), starting with <?xml]

6.1.4.2 Example 2: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove this section >>

6.1.4.2.1 Request

[HTTP headers]

[XML request (if applicable), starting with <?xml]

6.1.4.2.2 Response

[HTTP headers]

[XML response (if applicable), starting with <?xml]

6.1.5 POST

<< This is a blueprint for POST in case it is **not** a valid operation>>

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: [GET/PUT/DELETE]’ field in the response as per section 14.7 of [RFC 2616].

<< This is a blueprint for POST in case it is a valid operation>>

This operation is used for [description of operation].

If the resource is on the server side and it supports creating a subscription for notifications, and if the use of Notification Channel is supported, include/adapt this paragraph, otherwise delete it. The notifyURL in the callbackReference either contains the Client-side Notification URL (as defined by the client) or the Server-side Notification URL (as obtained during the creation of the Notification Channel [REST_NetAPI_NotificationChannel]).

6.1.5.1 Example 1: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove the sequence number from the title heading >>

6.1.5.1.1 Request

[HTTP headers]

[XML request (if applicable, starting with <?xml]

6.1.5.1.2 Response

[HTTP headers]

[XML response (if applicable, starting with <?xml]

6.1.5.2 Example 2: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove this section >>

6.1.5.2.1 Request

[HTTP headers]

[XML request (if applicable, starting with <?xml]

6.1.5.2.2 Response

[HTTP headers]

[XML response (if applicable, starting with <?xml]

6.1.6 DELETE

<< This is a blueprint for DELETE in case it is **not** a valid operation>>

Method not allowed by the resource. The returned HTTP error status is 405. The server should also include the ‘Allow: [GET/PUT/POST]’ field in the response as per section 14.7 of [RFC 2616].

<< This is a blueprint for DELETE in case it is a valid operation>>

This operation is used for [description of operation].

6.1.6.1 Example 1: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove the sequence number from the title heading >>

6.1.6.1.1 Request

[HTTP headers]

[XML request (if applicable, starting with <?xml]

6.1.6.1.2 Response

[HTTP headers]

[XML response (if applicable, starting with <?xml]

6.1.6.2 Example 2: [Example title] (Informative)

<< Section 6.1.3.1 provides guidance w.r.t. naming conventions and validation of examples.

If there is only one example, remove this section >>

6.1.6.2.1 Request

[HTTP headers]

[XML request (if applicable, starting with <?xml]

6.1.6.2.2 Response

[HTTP headers]

[XML response (if applicable, starting with <?xml]

7. Fault definitions

7.1 Service Exceptions

<< This section provides details about Service Exception type of faults specific for that particular API. Some APIs do have specific Service Exception fault definitions, some don't have. Pick the right text block. Delete this comment. >>

<< If API has **no** specific Service Exception codes defined either in that particular API version, or in the baseline version, use the following text. Delete this comment. >>

For common Service Exceptions refer to [REST_NetAPI_Common]. There are no additional Service Exception codes defined for the RESTful [Functional Area] API.

<< If API **has** specific Service Exception codes defined either in that particular version, or in the baseline version, use the following text and include the codes in the table(s).

Service Exception codes consists of a prefix "SVC" followed by 4 digit code number.
The original Service Exception codes from the baseline product (if any) are included unchanged.
For a new Service Exception code, 4 digit code number The API provider may double check the current used OMA OMNA reserved code in order to avoid overlap.Delete this comment >>

For common Service Exceptions refer to [REST_NetAPI_Common]. The following additional Service Exception codes are defined for the RESTful [Functional Area] API.

7.1.1 SVC[code number]: [Text for exception header]

Name	Description
MessageID	SVC[code number]
Text	[Text describing the fault with optional replacement variables marked with %n, where n is an index into the list of <Variables> elements, starting at 1]
Variables	[%n variables to substitute into the string, or "None"]
HTTP status code(s)	[HTTP status code(s) where that particular Service Exception code can be used with]

7.2 Policy Exceptions

<< This section provides details about Policy Exception type of faults specific for that particular API. Some APIs do have specific Policy Exception fault definitions, some don't have. Pick the right text block. Delete this comment. >>

<< If API has **no** specific Policy Exception codes defined either in that particular API version, or in the baseline version, use the following text. Delete this comment. >>

For common Policy Exceptions refer to [REST_NetAPI_Common]. There are no additional Policy Exception codes defined for the RESTful [Functional Area] API.

<< If API **has** specific Policy Exception codes defined either in that particular version, or in the baseline version, use the following text and include the codes in the table(s).

Policy Exception code consists of a prefix "POL" followed by 4 digit code number.
The original Policy Exception codes from the baseline product (if any) are included unchanged.
For a new Policy Exception code, 4 digit code number The API provider may double check the current used OMA OMNA reserved code in order to avoid overlap Delete this comment. >>

For common Policy Exceptions refer to [REST_NetAPI_Common]. The following additional Policy Exception codes are defined for the RESTful [Functional Area] API.

7.2.1 POL[code number]: [Text for exception header]

Name	Description
MessageID	POL[code number]
Text	[Text describing the fault with optional replacement variables marked with %n, where n is an index into the list of <variables> elements, starting at 1]
Variables	[%n variables to substitute into the string, or "None"]
HTTP status code(s)	[HTTP status code(s) where that particular Policy Exception code can be used with]

<< Example - DELETE this row and the following table. >>

7.2.1 POL1003: Refund exceeds original charge amount

Name	Description
MessageID	POL1003
Text	The refund amount exceeds the original amount charged %1
Variables	%1 – the original amount charged
HTTP status code(s)	403 Forbidden

Appendix A. Change History (Informative)

A.1 Version 1.0 History

Document Identifier	Date	Sections	Description
Version: REST_NetAPI _[FuncArea]-V1_0	[dd mmm yyyy]	[sections]	[List of all modifications]

Appendix B. JSON examples (Informative)

JSON (JavaScript Object Notation) is a Light-weight, text-based, language-independent data interchange format. It provides a simple means to represent basic name-value pairs, arrays and objects. JSON is relatively trivial to parse and evaluate using standard JavaScript libraries, and hence is suited for REST invocations from browsers or other processors with JavaScript engines. Further information on JSON can be found at [RFC7159].

The following examples show the request and response for various operations using the JSON data format. The examples follow the XML to JSON serialization rules in [REST_NetAPI_Common]. A JSON response can be obtained by using the content type negotiation mechanism specified in [REST_NetAPI_Common].

For full details on the operations themselves please refer to the section number indicated.

B.1 [Example Title] (section [section number cross reference])

<< Example title should be copied from title heading of the section with the corresponding XML example. And that section number should be added in brackets to the end of the example title >>

Request:

```
[HTTP headers copied from referenced example, type specific signalling updated for JSON]

[JSON example may be generated automatically from the equivalent XML example by using XML to JSON conversion tools]
```

Response:

```
[HTTP headers copied from referenced example, type specific signalling updated for JSON]

[JSON example may be generated automatically from the equivalent XML example by using XML to JSON conversion tools]
```

Appendix C. [Baseline specification] operations mapping(Informative)

<< This appendix is only needed for specifications which define REST bindings for an existing interface / API

.

In case there is no baseline, the headline is "Operations mapping to a pre-existing baseline specification (Informative)"

Delete this comment.>>

<<If there is **no** baseline, use the following wording. Delete this comment. >>

As this specification does not have a baseline specification, this appendix is empty.

<<If there is a baseline, use the following wording. Delete this comment. >>

The table below illustrates the mapping between REST resources/methods defined in this specification and [Baseline specification] [[BASELINE_REF]] equivalent operations.

REST Resource	REST Method	REST Section reference	[Baseline specification] equivalent operation
[Resource description from first column in one of the tables in section 5.1]	[GET/PUT/ POST/DEL ETE]	[section cross-reference]	[Operation name from Baseline specification]
<< Add/Remove rows to this table as needed - DELETE This Row>>			

Table 1 [Baseline specification] operations mapping

Appendix D. Light-weight Resources (Informative)

<< This appendix lists Light-weight Resources defined in this specification. Delete this comment>>

<<If there are **no** Light-weight Resource in the spec, the following wording is used. Delete this comment.>>

As this version of the specification does not define any Light-weight Resources, this appendix is empty.

<<If there are no Light-weight Resource in the spec, the following wording is used. Delete this comment.>>

The following table lists all [FuncArea] data structure elements that can be accessed individually as Light-weight Resources.

For each Light-weight Resource, the following information is provided: corresponding root element name, root element type and [ResourceRelPath] string.

Type of Light-weight Resources (and references to data structures)	Element/attribute that can be accessed as Light-weight Resource	Root element name for the Light-weight Resource	Root element type for the Light-weight Resource	[ResourceRelPath] string that needs to be appended to the corresponding Heavy-weight Resource URL
[Resource Type] ([section ref])	[child element name]	[root element name]	[root element type]	[ResourceRelPath]
	[child element name]	[root element name]	[root element type]	[ResourceRelPath]
<< Example below - DELETE this Row and the following table>>				
Presence data (5.2.3)	person	person	PersonAttributes	person
	service	service	ServiceAttributes	service/{serviceId}/{version}
	device	device	DeviceAttributes	device/{deviceId}

<<If [ResourceRelPath] strings in the above table include variables in curly brackets "{}" then the following paragraph shall be included also; otherwise it should be deleted. Delete this comment >>

Note: When appending [ResourceRelPath] string to its Heavy-weight Resource URL, all variables within curly brackets "{}" such as: [list of variable names from ResourceRelPath strings] have to be replaced by their real values.

Appendix E. Authorization aspects (Normative)

<< This appendix lists authorization aspects specific of the particular API, such as OAuth scope values. It is mandatory but may be empty. Delete this comment. >>

<< If there are **no** Authorization aspects specified for the specs, the following wording is used. Delete this comment. >>

None specified in this version of the specification.

<< If there **are** Authorization aspects specified for the specs, the following wording is used. Delete this comment. >>

This appendix specifies how to use the RESTful [Func Area] API in combination with some authorization frameworks.

E.1 Use with OMA Authorization Framework for Network APIs

The RESTful [FuncArea] API MAY support the authorization framework defined in [Autho4API_10].

A RESTful [FuncArea] API supporting [Autho4API_10]:

- SHALL conform to section D.1 of [REST_NetAPI_Common];
- SHALL conform to this section G.1.

E.1.1 Scope values

E.1.1.1 Definitions

In compliance with [Autho4API_10], an authorization server serving clients requests for getting authorized access to the resources exposed by the RESTful [FuncArea] API:

- SHALL support the scope values defined in the table below;
- MAY support scope values not defined in this specification.

Scope value	**Description**	**For one-time access token**
[Scope value]	[Scope value description]	[No/Yes]
[Scope value]	[Scope value description]	[No/Yes]
<< Example - DELETE this and next two Rows>>		
oma_rest_messaging.all_{apiVersion}	Provide access to all defined operations on the resources in this version of the API. The {apiVersion} part of this identifier SHALL have the same value as the "apiVersion" URL variable which is defined in section 5.1. This scope value is the union of the other scope values listed in next rows of this table.	No
oma_rest_messaging.in_regist	Provide access to all defined operations on inbound messages using registration	No

Table 2: Scope values for RESTful [FuncArea] API

E.1.1.2 Downscoping

In the case where the client requests authorization for “oma_rest_funcarea.all_{apiVersion}” scope, the authorization server and/or resource owner MAY restrict the granted scope to some of the following scope values:

- [list of scope values]

E.1.1.3 Mapping with resources and methods

Tables in this section specify how the scope values defined in section G.1.1.1 for the RESTful [FuncArea] API map to the REST resources and methods of this API. In these tables, the root “oma_rest_funcarea.” of scope values is omitted for readability reasons.

<< Note: this part of the TS uses a landscape layout, started and terminated by a section break. Delete this comment. >>

Resource	URL Base URL: http://{serverRoot}/Functional Area/{apiVersion}	Section reference	HTTP verbs			
			GET	PUT	POST	DELETE
[Description of the resource]	[URL for the resource]	[Section refrerence]	[supported scope value(s)]	[supported scope value(s)]	[supported scope value(s)]	[supported scope value(s)]
<< Example below - DELETE this and the following Row>>						
Inbound messages for a given registration	/inbound/registrations/{registrationId}/messages	6.1	**all_{apiVersion}** or **in_regist**	n/a	n/a	n/a

Table 3: Required scope values for: [text describing function(s) associated with that particular scope values]

E.1.2 Use of 'acr:auth'

<< Some APIs do have user identifiers in resource URL that could be a subject for 'acr: auth', some don't have. Pick the right text block. Delete this comment. >>

<<If **there are no** user identifiers candidate for 'acr: auth', the following wording is used. Delete this comment. >>

As this version of the specification does not define any parameter that could be a candidate for 'acr: auth', this appendix is empty

<< The text below is a blueprint of Appendix E.1.2 if **there are** user identifiers candidate for 'acr: auth'. Delete this comment. >>

This section specifies the use of 'acr:auth' in place of an end user identifier in a resource URL path.

An 'acr' URI of the form 'acr: auth', where 'auth' is a reserved keyword MAY be used to avoid exposing a real end user identifier in the resource URL path.

A client MAY use 'acr: auth' in a resource URL in place of a {senderAddress} replace/adapt "senderAddress" with a variable name of end user identifier which is a candidate for acr:auth. If multiple identifiers are candidate they shall be separated by comma. when the the RESTful [FuncArea] API is used in combination with [Autho4API_10].

In the case the RESTful [FuncArea] API supports [Autho4API_10], the server:

- SHALL accept 'acr:auth' as a valid value for the resource URL variable {senderAddress} replace/adapt "senderAddress" with a name of end user identifier which is a candidate for acr:Authorization. If multiple identifiers are candidate they shall be separated by comma.
- SHALL conform to [REST_Common_TS] section 5.8.1.1 regarding the processing of 'acr:auth'.