



# Common definitions and specifications for OMA REST interfaces

Candidate Version 1.0 – 11 Jan 2011

---

**Open Mobile Alliance**  
OMA-TS-REST\_Common-V1\_0-20110111-C

Use of this document is subject to all of the terms and conditions of the Use Agreement located at <http://www.openmobilealliance.org/UseAgreement.html>.

Unless this document is clearly designated as an approved specification, this document is a work in process, is not an approved Open Mobile Alliance™ specification, and is subject to revision or removal without notice.

You may use this document or any part of the document for internal or educational purposes only, provided you do not modify, edit or take out of context the information in this document in any manner. Information contained in this document may be used, at your sole risk, for any purposes. You may not use this document in any other manner without the prior written permission of the Open Mobile Alliance. The Open Mobile Alliance authorizes you to copy this document, provided that you retain all copyright and other proprietary notices contained in the original materials on any copies of the materials and that you comply strictly with these terms. This copyright permission does not constitute an endorsement of the products or services. The Open Mobile Alliance assumes no responsibility for errors or omissions in this document.

Each Open Mobile Alliance member has agreed to use reasonable endeavors to inform the Open Mobile Alliance in a timely manner of Essential IPR as it becomes aware that the Essential IPR is related to the prepared or published specification. However, the members do not have an obligation to conduct IPR searches. The declared Essential IPR is publicly available to members and non-members of the Open Mobile Alliance and may be found on the “OMA IPR Declarations” list at <http://www.openmobilealliance.org/ipr.html>. The Open Mobile Alliance has not conducted an independent IPR review of this document and the information contained herein, and makes no representations or warranties regarding third party IPR, including without limitation patents, copyrights or trade secret rights. This document may contain inventions for which you must obtain licenses from third parties before making, using or selling the inventions. Defined terms above are set forth in the schedule to the Open Mobile Alliance Application Form.

NO REPRESENTATIONS OR WARRANTIES (WHETHER EXPRESS OR IMPLIED) ARE MADE BY THE OPEN MOBILE ALLIANCE OR ANY OPEN MOBILE ALLIANCE MEMBER OR ITS AFFILIATES REGARDING ANY OF THE IPR'S REPRESENTED ON THE “OMA IPR DECLARATIONS” LIST, INCLUDING, BUT NOT LIMITED TO THE ACCURACY, COMPLETENESS, VALIDITY OR RELEVANCE OF THE INFORMATION OR WHETHER OR NOT SUCH RIGHTS ARE ESSENTIAL OR NON-ESSENTIAL.

THE OPEN MOBILE ALLIANCE IS NOT LIABLE FOR AND HEREBY DISCLAIMS ANY DIRECT, INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF DOCUMENTS AND THE INFORMATION CONTAINED IN THE DOCUMENTS.

© 2011 Open Mobile Alliance Ltd. All Rights Reserved.

Used with the permission of the Open Mobile Alliance Ltd. under the terms set forth above.

# Contents

1.	SCOPE.....	4
2.	REFERENCES.....	5
2.1	NORMATIVE REFERENCES.....	5
2.2	INFORMATIVE REFERENCES.....	5
3.	TERMINOLOGY AND CONVENTIONS.....	6
3.1	CONVENTIONS.....	6
3.2	DEFINITIONS.....	6
3.3	ABBREVIATIONS.....	6
4.	INTRODUCTION.....	7
4.1	VERSION 1.0.....	7
5.	COMMON SPECIFICATIONS FOR OMA REST INTERFACES.....	8
5.1	USE OF REST GUIDELINES.....	8
5.2	UNSUPPORTED FORMATS.....	8
5.3	AUTHORING STYLE.....	8
5.3.1	Names.....	8
5.3.2	Case usage for names.....	8
5.4	CONTENT TYPE NEGOTIATION.....	8
5.5	RESOURCE CREATION.....	9
5.6	JSON ENCODING IN HTTP REQUESTS/RESPONSES.....	9
5.6.1	Serialization rules: general conversion.....	9
5.6.2	Serialization rules: structure-aware conversion.....	11
5.6.3	Rules for JSON-creating and JSON-consuming applications.....	13
6.	DATA ITEMS.....	15
6.1	ADDRESS DATA ITEMS.....	15
7.	ERROR HANDLING.....	16
7.1	HTTP RESPONSE CODES.....	16
7.2	HANDLING OF NOT ALLOWED HTTP METHODS.....	16
	APPENDIX A. CHANGE HISTORY (INFORMATIVE).....	17
A.1	APPROVED VERSION HISTORY.....	17
A.2	DRAFT/CANDIDATE VERSION 1.0 HISTORY.....	17
	APPENDIX B. DEPLOYMENT CONSIDERATIONS (INFORMATIVE).....	18
B.1	REST CLIENT APPLICATION EXECUTING IN A SERVER EXECUTION ENVIRONMENT.....	18
B.2	REST CLIENT APPLICATION EXECUTING IN A MOBILE DEVICE EXECUTION ENVIRONMENT.....	19
B.3	REST CLIENT APPLICATION EXECUTING IN A FIXED DEVICE EXECUTION ENVIRONMENT.....	20

## Figures

Figure 1	REST API accessed from a server execution environment (e.g. 3 <sup>rd</sup> party Service Provider application).....	18
Figure 2	REST API accessed from a mobile device execution environment.....	19
Figure 3	REST API accessed from a fixed device execution environment.....	20

# 1. Scope

The scope of this specification is to provide common definitions and specification material for REST interfaces in OMA.

## 2. References

### 2.1 Normative References

- [ISO4217] “ISO 4217 currency names and code elements”, URL: <http://www.iso.org/>
- [JSON] “The application/json Media Type for JavaScript Object Notation (JSON)”, D. Crockford, July 2006, URL: <http://www.ietf.org/rfc/rfc4627.txt>
- [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, June 1999, URL: <http://www.ietf.org/rfc/rfc2616.txt>
- [RFC3261] “SIP: Session Initiation Protocol”, J. Rosenberg, et. Al, June 2002, URL: <http://www.ietf.org/rfc/rfc3261.txt>
- [RFC3986] “Uniform Resource Identifier (URI): Generic Syntax”, T. Berners-Lee, R. Fielding, L. Masinter, January 2005, URL: <http://www.ietf.org/rfc/rfc3986.txt>
- [RFC3966] “The tel URI for Telephone Numbers”, H. Schulzrinne, December 2004, URL: <http://www.ietf.org/rfc/rfc3966.txt>
- [RFC4122] “A Universally Unique Identifier (UUID) URN Namespace”, P. Leach, M. Mealling, R. Salz, July 2005, URL: <http://www.ietf.org/rfc/rfc4122.txt>
- [SCRRULES] “SCR Rules and Procedures”, Open Mobile Alliance™, OMA-ORG-SCR\_Rules\_and\_Procedures, URL: <http://www.openmobilealliance.org/>
- [XMLSchema1] W3C Recommendation, XML Schema Part 1: Structures Second Edition, URL: <http://www.w3.org/TR/xmlschema-1/>
- [XMLSchema2] W3C Recommendation, XML Schema Part 2: Datatypes Second Edition, URL: <http://www.w3.org/TR/xmlschema-2/>

### 2.2 Informative References

- [OMADICT] “Dictionary for OMA Specifications”, Version 2.8, Open Mobile Alliance™, OMA-ORG-Dictionary-V2\_8, URL: <http://www.openmobilealliance.org/>
- [REST\_WP] “White Paper on Guidelines for REST API specifications”, Open Mobile Alliance™, OMA-WP-Guidelines-for-REST-API-specifications, URL:<http://www.openmobilealliance.org/>
- [XML2JSON] Open source XML to JSON conversion tool URL: <http://forge.morfeo-project.org>

## 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 [OMA-DICT].

[N/A]

[N/A]

### 3.3 Abbreviations

<b>AGPL</b>	Affero General Public License
<b>API</b>	Application Programming Interface
<b>DNS</b>	Domain Name Server
<b>HTTP</b>	Hypertext Transfer Protocol
<b>ID</b>	Identifier
<b>IP</b>	Internet Protocol
<b>JSON</b>	JavaScript Object Notation
<b>OMA</b>	Open Mobile Alliance
<b>PLMN</b>	Public Land Mobile Network
<b>REST</b>	Representational State Transfer
<b>URI</b>	Uniform Resource Identifier
<b>URL</b>	Uniform Resource Locator
<b>UUID</b>	Universal Unique Identifier
<b>XML</b>	Extensible Markup Language
<b>XSD</b>	XML Schema Definition

## 4. Introduction

To ensure consistency for developers using the various REST interfaces specified in OMA, this “Common” technical specification aims to contain all items that are common across all HTTP protocol bindings using REST architectural style for the various individual interface definitions, such as naming conventions, content type negotiation, representation formats and serialization, and fault definitions.

### 4.1 Version 1.0

This version of the REST Common Technical Specification contains naming conventions, content type negotiation, representation formats and serialization, and fault definitions for OMA REST interfaces.

## 5. Common Specifications for OMA REST interfaces

### 5.1 Use of REST Guidelines

Representational State Transfer (REST) is an architectural style for defining distributed systems. Entities in these systems communicate using the interfaces they expose. Guidelines for defining REST interfaces in OMA, including general key principles, have been collected in [REST\_WP].

### 5.2 Unsupported Formats

Servers must return a 406 Not Acceptable error if a message body format (e.g. XML or JSON) requested by the application is not supported [RFC2616].

### 5.3 Authoring Style

#### 5.3.1 Names

Names will be meaningful, and not abbreviated in a way that makes the name hard to understand for users of the REST interfaces that are not literate in computer programming. This does not preclude the use of commonly understood acronyms within names (e.g. ID) or commonly used abbreviations (e.g. max). However, the resulting name must still be meaningful.

#### 5.3.2 Case usage for names

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

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

The following data types will have a leading uppercase letter – Type names and names in an enumeration.

The following data types will have a leading lowercase letter – all other names.

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

## 5.4 Content type negotiation

The Content type of a response used SHALL be established using the following methodology:

As a general rule, content type used in response message body must match content type used in request body. At least XML and JSON content types MUST be supported.

Support for other content types will be specified on a case-by-case basis (e.g. simple name-value pair parameters may be accepted in the URL when using GET and www-form-urlencoded may be supported for the **request** message body when using POST or PUT).

Content type of the request message body SHALL always be determined by Content-Type header of the HTTP message.

Content type of the response body SHALL be determined using the following methodology. When invoking the API, the requesting application SHOULD include the ‘Accept’ request header, and provide the primary content type choice, and **OPTIONALLY** any supported substitute content types, in this request Accept header.

- a. If the server does not support the content type choice listed as priority in the Accept header, it SHALL attempt to return the next preferred choice if one was provided.
- b. If the requesting application does not provide an Accept header or any other indication of desired content type of the response (see further below), and the request message body content type is XML or JSON, then the server



SHALL provide a response message body with the content type matching that of the request message body. For example, a request with an XML body and no Accept header will trigger an XML response.

- c. If the requesting application requires the response message body to be of a different content type than the one determined by the request message body and the Accept header negotiations, it MUST request that content type by inserting in the URL path the query parameter “?resFormat={content type}”, where content type SHALL be either XML or JSON. This option overrides the Accept header provided by the application, if present, and the response format SHALL be determined solely by the “resFormat” parameter. Note that this allows an application that does not have sufficient control over the HTTP headers to enforce a response format regardless of the value of the Accept header.
- d. If the server cannot return any of the content types based on the negotiation steps described, it SHALL return a 406 response code as per [RFC2616].
- e. The default format for notification payloads SHALL be XML, unless the client has specified *notificationFormat= "JSON"* in the subscription.
- f. Content type SHALL accompany HTTP response codes 200, 201, 400, 409 in the conditions dictated by the above specified methodology, and MAY be omitted in other cases.

## 5.5 Resource creation

Typically, a resource is created either following a POST request (to create a child of an existing resource that is addressed by the request), or following a PUT request (to create a new resource as addressed by the request).

If a resource has been created on the server, the server SHALL return an HTTP response with a "201 Created" header and the Location header containing the location of the created resource, and SHALL include in the response body either a resourceReference element, or a representation of the created resource. Note that this allows the server to control the traffic.

Further note that REST resource representations are designed in such a way that they can include a self reference. (i.e. resourceURL element.). A self reference is always present in any data structure that is a representation of a resource created by POST, and can be included as necessary in other cases. Since a self reference can be defined as a mandatory or optional element to accommodate different situations, the normative aspects on the client and on the server in each optional usage instance in the specification are clarified as follows: the resourceURL SHALL NOT be included in POST requests, and MUST be included in responses to any HTTP method that returns an entity body, and in PUT requests.

Generally resources are used to access entire data structure and those resources are regarded as heavy-weight resources. To access a part of the data structure or an individual elements in the data structure, another type of resources called light-weight resources are used. Compared to heavy-weight resources, light weight resources are created following PUT request only (see [REST\_WP] for more details about light-weight resources).

Elements in data structures with a key properties (keys) are normally not accessible by using light-weight resources, however when accessing other elements using light-weight resources they may appear in both the light-weight resource URL and in the body of the request. In case the server receives PUT request with keys, it SHALL ensure that the key value(s) specified in the URL match those value(s) specified in the body of the request. If not, the server SHALL respond with “409 Conflict” indicating key value(s) conflict.

## 5.6 JSON encoding in HTTP Requests/Responses

### 5.6.1 Serialization rules: general conversion

Specifications of REST APIs MAY include XML schema files defining the data structures used by the API, for its direct usage in XML format. The following are general rules for mapping between XML and JSON data formats:

- a. XML elements that appear at the same XML hierarchical level (i.e. either root elements or within the same XML parent element), are mapped to a set of *name:value* pairs within a JSON object, as follows:

- (i) Each XML element appearing only once at the same hierarchical level (“*single element*”) is mapped to an individual *name:value* pair. The *name* is formed according to bullet b, while the *value* is formed according to bullet c.
  - (ii) XML elements appearing more than once at the same hierarchical level (“*element list*”) are mapped to only one, individual *name:value* pair. The *name* is formed according to bullet b, while the *value* is a JSON array containing one *value* per each occurrence of the XML element. The name is formed according to bullet b whilst values are formed according to bullet c.
  - (iii) *Name* and *Value* of JSON objects will go between “”. Additionally, any JSON representation of an element of complex type will go between {}, according to [JSON].
- b. The *name* of the *name:value* pair is the name of the XML elements (i.e. XML\_element\_name:value)
  - c. The *value* is formed as follows:
    - (i) when the XML element has neither attributes nor child XML elements, the *value* is equal to the value of the XML element. In case the element is null (i.e it has no value), it will be indicated as having a “null” value within JSON.
    - (ii) when the XML element has child elements and/or attributes, the *value* is a JSON object containing the following *name:value* pairs:
      - one *name:value* pair per each attribute, where *name* is the name of the attribute and *value* is the value of the attribute.
      - one *name:value* pair associated to the text value (simple type content) of the XML element, where *name* is the string “\$t” and *value* is the value of the XML element.
      - *name:value* pairs associated to XML child elements. These *name:value* pairs are formed in accordance with bullet a.

Within JSON, there is no need to reflect:

- the first <?xml version="1.0" encoding="UTF-8" ?> tag
- declaration of namespaces or schemaLocations

In order to generate unambiguous JSON from XML instances, based on the rules defined above, the following limitations need to be imposed on the XML data structures:

- it is not allowed that two different elements from different namespaces have the same name, in case they appear at the same level
- within an XML parent element, no attribute is allowed to have the same name as a child element of this parent element.

**Note:** These general rules have been used to generate the JSON examples from the XML examples in the Technical Specifications of the REST Enabler.

### 5.6.1.1 Utility which implements the general conversion rules (Informative)

The general conversion rules are implemented with UNICA XML2JSON utility, an open source tool, distributed, under an AGPL license, within the open source community MORFEO [XML2JSON].

### 5.6.1.2 Example

(Informative)

The following is an example illustrating the guidelines:

Input XML content:

```
<Animals>
  <dog>
    <name attr="1234">Rufus</name>
    <Breed>labrador</Breed>
  </dog>
  <dog>
    <name>Marty</name>
    <Breed>whippet</Breed>
  </dog>
  <dog/>
  <cat name="Matilda"/>
  <a/>
</Animals>
```

Transformed JSON:

```
{"Animals": {
  "a": null,
  "cat": {"name": "Matilda"},
  "dog": [
    {
      "Breed": "labrador",
      "name": {
        "$t": "Rufus",
        "attr": "1234"
      }
    },
    {
      "Breed": "whippet",
      "a": null,
      "name": "Marty"
    },
    null
  ]
}}
```

## 5.6.2 Serialization rules: structure-aware conversion

The general approach as defined above relies only on the information in the XML data instance.

The structure-aware approach defined in this section considers information in a data instance (e.g. XML) plus further information about the data structure definition (such as the allowed number of element occurrences), as documented in the API specifications and XML Schemas.

This structure-aware approach allows having always the same JSON structure to convey lists of elements.

In this conversion approach, the rules above apply, except for the following modification to the conditions in a (i) and a (ii): If an element is allowed to appear more than once at the same hierarchical level, it SHALL be treated according to a (ii) as element list, otherwise it SHALL be treated according to a (i) as single element.

### 5.6.2.1 Example

(Informative)

The following example illustrates the structure-aware serialization.

In the example, the data instance is represented as XML document:

```
<Animals>
  <dog>
    <name attr="1234">Rufus</name>
    <Breed>labrador</Breed>
  </dog>
  <dog>
    <name>Marty</name>
    <Breed>whippet</Breed>
    <a/>
  </dog>
</dog/>
<cat name="Matilda"/>
<a/>
</Animals>
```

The information about the data structure is represented as XML schema in this example. Note that the maximum cardinality of the elements is the only piece of information that is used here.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Animals">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="dog" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" minOccurs="0">
                <xsd:complexType>
                  <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                      <xsd:attribute name="attr" type="xsd:string"/>
                    </xsd:extension>
                  </xsd:simpleContent>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="Breed" type="xsd:string" minOccurs="0"/>
              <xsd:element name="a" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

</xsd:element>
<xsd:element name="cat" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="a"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

```

Transformed JSON:

```

{"Animals": {
  "dog": [
    {
      "name": {
        "$t": "Rufus",
        "attr": "1234"
      }
      "Breed": "labrador",
    },
    {
      "name": "Marty"
      "Breed": "whippet",
      "a": null,
    },
    null
  ]
  "cat": [{"name": "Matilda"}],
  "a": null,
}}

```

### 5.6.3 Rules for JSON-creating and JSON-consuming applications

A JSON-creating application SHALL use either the *structure-aware* or the *general* approach, but not both.

Applications that consume a JSON representation SHALL accept the following two different JSON representations for an array that contains one element:

1. a pair of name and value (e.g. “name”: “one”)

2. a pair of name and array of one value (e.g. "name": ["one"])

**Note:** In JSON, according to [RFC4627], the order of objects is not significant, whilst the order of values within an array is.

## 6. Data Items

### 6.1 Address data items

Addresses, unless the specification provides specific additional instruction, MUST conform to the address portion of the URI definition provided in [RFC3966] for 'tel:' addresses, [RFC3261] for 'sip:' addresses or the definition given below for shortcodes or aliased addresses. Optional additions to the address portion of these URI definitions MUST NOT be considered part of the address accepted by the REST interfaces, and an implementation MAY choose to reject an address as invalid if it contains any content other than the address portion.

When processing a 'tel:' URI, as specified in [RFC3966], the REST interface MUST accept national addresses (those not starting with '+' and a country code) and MUST accept international addresses (those starting with '+' and a country code).

When specified in the definition of a service operation, the URI may contain wildcard characters in accordance with the appropriate specification (i.e. [RFC3966] or [RFC3261]).

Shortcodes are short telephone numbers, usually 4 to 6 digits in length reserved for telecom service providers' own functionality. They shall be differentiated from national addresses by the use of a 'short:' rather than 'tel:' URI scheme. The short code defined in the URI consists of a string of digits with no non-digit characters.

Support for aliases in addresses is provided by use of the URI defined in [RFC3986]. One can not assume that the resource the alias references can be determined without using the URI to access the resource.

An alias is generally a relatively short character string that holds a scrambled address such that only the application identified in the URI can expand it.

## 7. Error Handling

### 7.1 HTTP Response Codes

Following is the list of commonly used HTTP response codes for OMA REST interfaces.

**200** - Success

**201** - Created

**204** - No Content

**303** - See Other - The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource.

**304** - ConditionNotMet - Not Modified: The condition specified in the conditional header(s) was not met for a read operation.

**400** - Invalid parameters in the request

**401** - Authentication failure

**403** - Application don't have permissions to access resource due to the policy constraints (request rate limit, etc)

**404** - Not Found - The specified resource does not exist.

**405** - Method not allowed by the resource

**406** – Not Acceptable

**409** - Conflict

**411** - Length Required: The Content-Length header was not specified.

**412** - Precondition Failed: The condition specified in the conditional header(s) was not met for a write operation.

**413** - RequestBodyTooLarge - Request Entity Too Large: The size of the request body exceeds the maximum size permitted.

**415** – Unsupported Media Type

**416** - InvalidRange - Requested Range Not Satisfiable: The range specified is invalid for the current size of the resource

**500** - Internal server error

**503** - ServerBusy - Service Unavailable: The server is currently unable to receive requests. Please retry your request.

### 7.2 Handling of not allowed HTTP methods

If a method is not allowed by the resource, then server SHOULD also include the 'Allow: {GET|PUT|POST|DELETE}' HTTP header in the response as per section 14.7 in [RFC2616].



## Appendix A. Change History (Informative)

### A.1 Approved Version History

Reference	Date	Description
n/a	n/a	No prior version -or- No previous version within OMA

### A.2 Draft/Candidate Version 1.0 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-TS-REST_Common-V1_0	18 Aug 2010	All	First draft, derived from draft Common TS V1_1 (August 13) See OMA-ARC-REST-2010-0413R01-INP_Splitting_the_Common_TS
	31 Aug 2010		Implementation of contribution OMA-ARC-REST-2010-0395-CR_Common_TS_additional_status_codes OMA-ARC-REST-2010-0446R01- CR_Common_changes_regarding_self_reference OMA-ARC-REST-2010-0468- INP_CR_HTTP_Code_303_in_TS_REST_Common
	27 Sep 2010		Implementation of contribution OMA-ARC-REST-2010-0263R01-CR_Common_handling_HTTP_responses OMA-ARC-REST-2010-0493R01- CR_Common_update_for_light_weight_resources
	07 Oct 2010	All	Editorial fixes: Styles as per template History table
	22 Nov 2010	All	Implementation of OMA-CONRR-ParlayREST-V2_0: B001, AJ005 OMA-ARC-REST-2010-0634-CR_resformat_Note_in_general_TS_Common OMA-ARC-REST-2010-0635- CR_Case_convnetion_for_Resource_Names_TS_Common OMA-ARC-REST-2010-0648-CR_CONRR_resolutions_Generic_TS_Common
	07 Dec 2010		Incorporated: OMA-ARC-REST-2010-0677-CR_A243_alias_in_REST_Common
	10 Dec 2010	5.5	Implementation of OMA-ARC-REST-2010-0705- CR_A241_A242_CONRR_comments_AJ001_004_for_Common
Candidate Version: OMA-TS-REST_Common-V1_0	11 Jan 2011	All	Status changed to Candidate by TP: OMA-TP-2010-0531R01-INP_ParlayREST_2_0_for_Candidate_approval

## Appendix B. Deployment Considerations (Informative)

Applications using the REST API can be categorized by their execution environment:

- Application is a REST client application executing in a server execution environment (e.g. a 3rd party application).
- Application is a REST client application executing in a mobile device execution environment.
- Application is a REST client application executing in a fixed device execution environment.

An API client can execute in any of the above execution environments.

Issues that are dependent on the execution environment and can impact strategic deployment decisions, interoperability, and scalability include (non-exhaustive list):

- Notifications sent from server to client application, for example:
  - i. There must be an active "listener" on the application host (in this case the client device), ready to receive the incoming notification via the HTTP protocol.
  - ii. This does not have to be the application itself, but at least some host service/client which can invoke the specific application when needed.
  - iii. In a client-server HTTP binding, this requires that the client has the support of an HTTP listener service.
- Security aspects (e.g. client application authentication)

While solutions to particular issues related to the client application execution environment are out-of-scope for the REST enabler, other OMA enablers should be re-used (where applicable) to address such particular issues.

### B.1 REST client application executing in a server execution environment

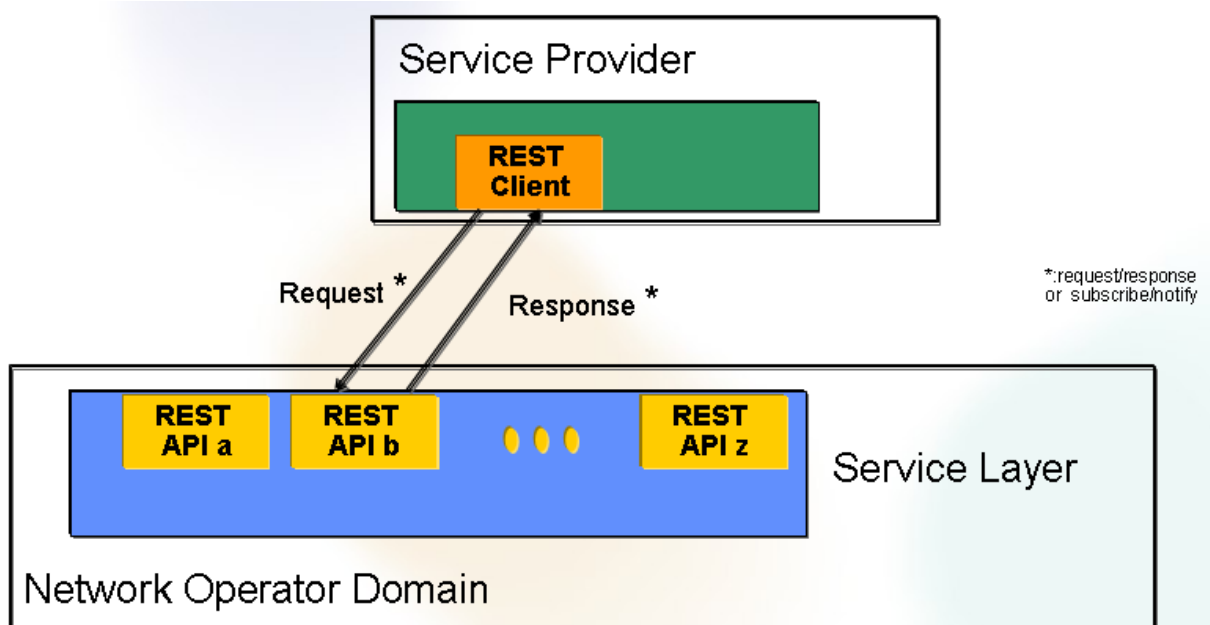
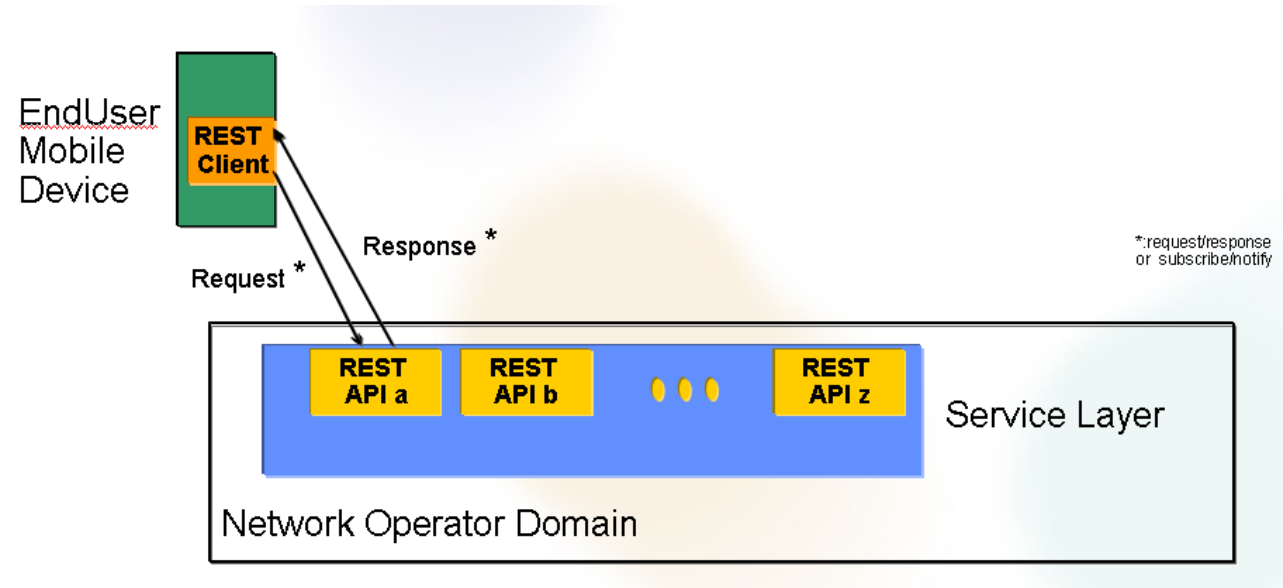


Figure 1 REST API accessed from a server execution environment (e.g. 3<sup>rd</sup> party Service Provider application)

The RESTful API exposed by the server deployed in the Network Operator service layer domain, may be accessed by a client application executing on a server resident in the Service Provider domain. This deployment can support all resources and operations specified in the REST enabler. There are no particular issues with support of notifications from a server to a client application. For security aspects, see the Common TS security considerations section.

## B.2 REST client application executing in a mobile device execution environment



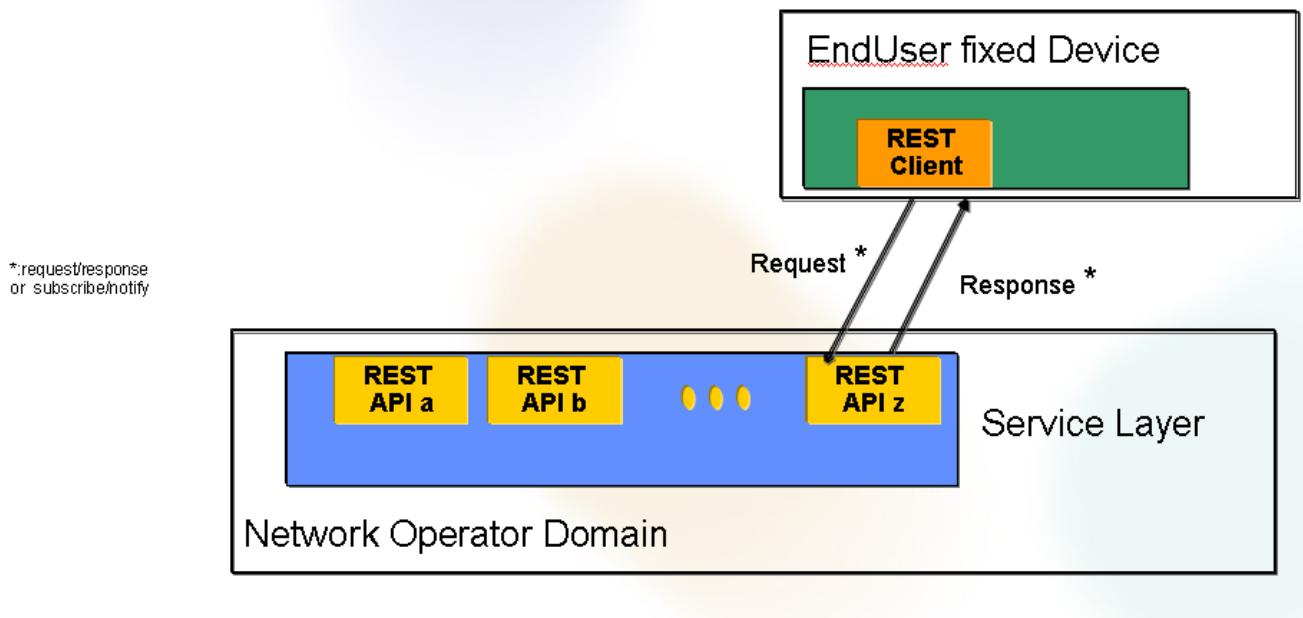
**Figure 2 REST API accessed from a mobile device execution environment**

The RESTful API exposed by a server deployed in the Network Operator service layer domain, may be accessed by a client application executing on an end user mobile device. This deployment can support most resources and operations specified in the enabler. There are however particular issues with support of notifications from server to client application:

- Typically in mobile devices, the client does not have the support for an HTTP listener service. The specified client notification APIs may have to be delivered by alternative means. OMA Push [Push] should be considered to be used to deliver the notifications to the client application.
- It must be possible to actually deliver the notification to the client application, i.e. there must be no boundary across which the protocol is typically blocked. In a client-server HTTP binding, this will typically be an issue as
  - The client is typically within some private network behind a firewall (e.g. PLMN Operator mobile network or home network)
  - The client does not have a fixed IP address or an IP address that is resolvable via DNS.
  - In such cases, a notification service such as OMA Push should be considered to be used to bridge the firewall border and resolve the target address of the notification to an actual client address.

For security aspects, see the Common TS security considerations section.

### B.3 REST client application executing in a fixed device execution environment



**Figure 3 REST API accessed from a fixed device execution environment**

The RESTful API exposed by a server deployed on the Network Operator service layer domain, may be accessed by a client application executing on a fixed device connected to the Network Operator.

This deployment can support most resources and operations specified in the enabler. Some issues with support of notifications from server to client applications may be similar to those mentioned in Appendix B.2. Solutions to those issues may however rely on other mechanisms (e.g. use of COMET).

For security aspects, see the Common TS security considerations section.