ETSI TS 132 158 V15.2.0 (2019-06)

LTE;
5G;
Management and orchestration;
Design rules for REpresentational State Transfer (REST)
Solution Sets (SS)
(3GPP TS 32.158 version 15.2.0 Release 15)

Reference
RTS/TSGS-0532158vf20

Keywords
5G,LTE

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Legal Notice

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

The present document defines design rules for REpresentational State Transfer (REST) Solution Sets (SS). These rules are applied when specifying REST Solution Sets.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] IETF RFC 7231: "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content".

[3] 3GPP TS 32.300: "Telecommunication management; Configuration Management (CM); Name convention for Managed Objects".

[4] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

[5] IETF RFC 7230: "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing".

[6] IETF RFC 7159: " The JavaScript Object Notation (JSON) Data Interchange Format".

[7] draft-wright-json-schema-01 (October 2017): "JSON Schema: A Media Type for Describing JSON Documents".
Editor's note: The above document cannot be formally referenced until it is published as an RFC.

[8] draft-wright-json-schema-validation-01 (October 2017: "JSON Schema Validation: A Vocabulary for Structural Validation of JSON".
Editor's note: The above document cannot be formally referenced until it is published as an RFC.

[9] draft-wright-json-schema-hyperschema-01 (October 2017): "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON.
Editor's note: The above document cannot be formally referenced until it is published as an RFC.

[10] OpenAPI Specification (https://github.com/OAI/OpenAPI-Specification)

[11] IETF RFC 5789: "PATCH Method for HTTP".

[12] IETF RFC 7396: "JSON Merge Patch".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

CRUD	Create, Retrieve, Update, Delete
DC	Domain Component
DN	Distinguished Name
DNS	Domain Name Service
FQDN	Fully Qualified Doman Name
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LDN	Local Distinguished Name
MnS	Management Service
REST	REpresentational State Transfer
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier

4 General rules

4.1 Information models and resources

4.1.1 Information models

An information model is a representation of a system. Typical models do not reflect all facets of the system, but only certain aspects required to solve the management problem the model is designed for. 3GPP follows an object-oriented modelling approach. Models are built from managed object classes. Relationships between classes represent the logical connections. Models are specified formally with class diagrams of the Unified Modelling Language (UML).

The instantiation of a managed object is called managed object instance. All managed object instances together with the relationships between them are depicted in an object diagram.

4.1.2 Resources

HTTP uses a different terminology based on the notion of resources, as defined in clause 2 of RFC 7231 [2]. Each resource is represented by a resource representation as defined in clause 3 of RFC 7231 [2]. Valid resource representations are e.g. XML instance documents or JSON instance documents.

4.1.3 Resource archetypes

Resources can be classified according to their structure and behaviour into resource archetypes. This helps specifying clear and understandable interfaces. The following three archetypes are defined:

- **Document resource**: This is the standard resource containing data in form of name value pairs and links to related resources. This kind of resource typically represents a real-world object or a logical concept.
- **Collection resource**: A collection resource is grouping resources of the same kind. The resources below the collection resource are called items of the collection. An item of a collection is normally a document resource. Collection resources typically contain links to the items of the collection and information about the collection like the total number of items in the collection. Collection resources can be further distinguished into server-managed and client-managed resources. Collection resources are also known as container resources.
- **Operation resource**: Operation resources represent executable functions. They may have input and output parameters. Operation resources allow some sort of fall back to an RPC style design in case application specific actions cannot be mapped easily to CRUD style operations.

4.1.4 Mapping of information models to resources

RESTful SS shall be specified in a way that managed object instances are described by document resources. Collection resources have no equivalent in an information model unless some dedicated collection class is introduced.

4.2 Managed object naming and resource identification

4.2.1 Managed object naming

4.2.1.0 Distinguished Name (DN)

The Distinguished Name (DN) is used in 3GPP to uniquely identify a managed object instance within a specific name space. The DN is a comma (",") separated list of Relative Distinguished Names (RDNs). Each managed object instance has an associated RDN. The sequence of RDNs is governed by name containment relationships in the UML class diagram describing the modelled network. The RDN consists of a naming attribute name separated by an equal sign ("=") from the naming attribute value. The naming attribute name is equal to the class name of the MOI.

In addition to the RDNs associated to a managed object instance the DN may have as leftmost RDN whose naming attribute name is "DC" (Domain Component) and whose value is a domain name. A DN with DC is globally unique.

The DN concept is described in detail in TS 32.300 [3].The following example DN has a DC.

```
DN = "DC=operatorA.com,subNetwork=south,managedElement=a,eNBFunction=1,cell=1"
```

4.2.1.1 Global and local namespaces

A DN in the global name space is globally unique and starts with the RDN of the global root. A DN in a local name space starts with the RDN of the local root and is unique only within this name space. A DN in a local namespace is also referred to as Local Distinguished Name (LDN). The DN of the local root relative to the global root is called DN prefix. The concatenation of DN prefix and LDN is equal to the globally unique DN of a managed object.

The local root is typically the root of the network resource model representing the managed network.

4.2.2 Resource identification

HTTP uses a subset of the generic Uniform Resource Identifier (URI) scheme (RFC 3986 [4]) defined in RFC 7230 [5] for target resource identification.

```
http-URI = "http:" "//" authority path-abempty [ "?" query ] [ "#" fragment ]
```

The path component is an absolute path (one that starts with a single slash character) or empty.

The origin server is identified by the authority component, which includes a host identifier and an optional path TCP port. The hierarchical path component and optional query component serve as an identifier for a potential target resource within that origin server’s name space. The optional fragment component allows for indirect identification of a secondary resource.The host identifier is either an IP address or an indirect identifier such as a FQDN to be resolved with DNS.

URIs are used by HTTP for routing and addressing of target resources. They shall not be used for other purposes or as an alternative for DNs.

4.2.3 Mapping of DNs to URIs

URIs are globally unique. For this reason only a globally unique DN with DC is mappable into a URI. The mapping rules are as follow:

- The DN prefix is mapped semantically to the authority component of the URI. The syntax of the DN prefix is modified to match the syntax of the authority component.

- The LDN is mapped semantically to the path component of the URI. The syntax of the LDN is modified to match the syntax of the path component.

When mapping a LDN the equal sign "="shall be used as delineator between the naming attribute name and naming attribute value when constructing a RDN.

```
URI-RDN = {namingAttributeName} "=" {namingAttributeValue}
```

The URI-LDN is the concatenation of URI-RDNs separated by a slash "/".

```
URI-LDN = *( "/" RDN )
```

For example, the LDN

```
LDN = "subNetwork=south,managedElement=a,eNBFunction=1,cell=1"
```

maps to

```
URI-LDN = "/subNetwork=south/managedElement=a/eNBFunction=1/cell=1"
```

and the LDN

```
LDN = "managedElement=a,eNBFunction=1,cell=1"
```

to

```
URI-LDN = "/managedElement=a/eNBFunction=1/cell=1"
```

When constructing the authority part from the DN prefix, it shall be reformatted according to the name conventions applying to FQDNs. For example, the DN prefix

```
DN-prefix = "DC=operatorA.com"
```

maps to

```
URI-DN-prefix = "operatorA.com"
```

and the DN prefix

```
DN-prefix = "DC=operatorA.com,subNetwork=south"
```

to

```
URI-DN-prefix = "south.subNetwork.operatorA.com"
```

The complete URIs for the examples are

```
http://operatorA.com/subNetwork=south/managedElement=a/eNBFunction=1/cell=1
http://south.subNetwork.operatorA.com/managedElement=a/eNBFunction=1/cell=1
```

The constructed URI-DN-prefix is a FQDN that can be registered into a name resolution service such as DNS. The sole presence of a constructed FQDN does not mean it can be resolved to an IP address and there is a server listening at that address.

Using the mapping rulea, a DN is mapped predictably into the URI authority component and path component.

The leftmost part of the path component may include one or more path segments ("label")

```
http://operatorA.com/{label}/subNetwork=south/.../cell=1
```

allowing to structure the resource hierarchy, for example

```
http://operatorA.com/3GPPmanagemen/ProvMnS/v1500/subNetwork=south/.../cell=1
```

The character set allowed in DNs is much bigger than the character set allowed in the path component and authority component of a URI. Care needs to be taken when selecting the naming attribute names und values that the mapping from a DN to a URI does not become impossible as a consequence of not mappable characters.

4.3 Media types

The format of resource representations carried in the message body is indicated by the media type in the Content-Type and Accept header fields. Media types that shall be supported are:

- application/json (RFC 7159 [6])
- application/merge-patch+json (RFC 7396 [12])

JSON resource representations shall conform to JSON Schema ([7], [8], [9]).

4.4 URI structure

URIs identifying resources representing managed object instances shall follow a common structure given by

```
http://{URI-DN-PREFIX}/{root}/{MnSName}/{MnSVersion}/{URI-LDN}
```

where:

{URI-DN-PREFIX} indicates the authority part of the URI constructed from the DN prefix.

{root} indicates an optional root for structuring the resource hierarchy.

{MnSName} indicates the optional MnS name.

{MnSVersion} indicates the optional version of the MnS.

{URI-LDN} indicates the resource path constructed from the LDN.

As seen above, to construct the URI from a DN, it is necessary to map the "DNPrefixPlusRDNSeparator" as defined in clause 7.3 of [3], the “LocalDN” as defined in clause 7.3 of [3], and to add the additional path components "/{root}/{MnSName}/{MnSVersion}".

To allow for a predictive mapping from the URI to the original DN it is necessary to specify "/{MnSName}/{MnSVersion}" in such a way that the beginning of the "LocalDN" can be identified.

4.5 Response status codes

The response status codes as defined in section 6 of RFC 7231 [2] shall be supported.

5 Basic design patterns

5.1 Design pattern for creating a resource

5.1.1 Creating a resource with identifier creation by the MnS Producer

Operations to create a resource shall be specified with the HTTP POST method, when the MnS Producer shall create the identifier of the new resource.

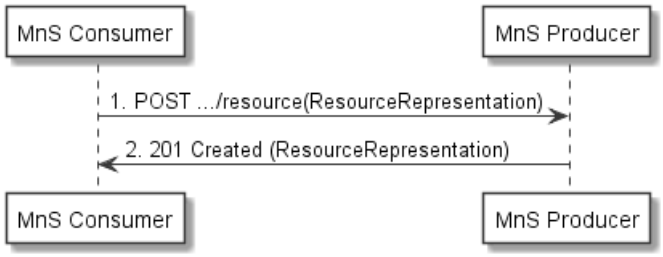

Figure 5.1.1-1: Flow for creating a resource with HTTP POST

The procedure is as follows:

1) The MnS Consumer sends a HTTP POST request to the MnS Producer. The target URI identifies the parent resource below which the new resource shall be created. Only container resources are valid target resources. The message body shall carry a complete resource representation.

2) The MnS Producer returns the HTTP POST response. On success, "201 Created" shall be returned. The "Location" header shall be present and carry the URI of the new resource. The URI is constructed by the MnS Producer by creating an identifier for the new resource and appending it to the request URI. The message body shall carry the complete representation of the new resource. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

5.1.2 Creating a resource with identifier creation by the MnS Consumer

Operations to create a resource shall be specified with the HTTP PUT method, when the MnS Consumer wishes to impose the identifier of the new resource to the MnS Producer.

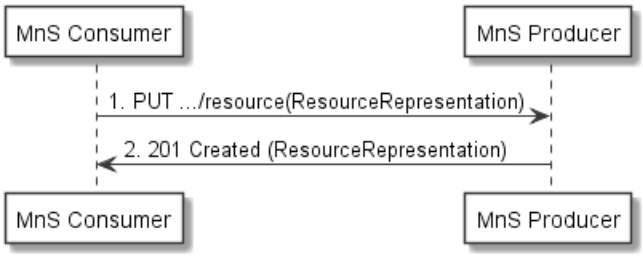

Figure 5.1.2-1: Flow for creating a resource with HTTP PUT

The procedure is as follows:

1) The MnS Consumer sends a HTTP PUT request to the MnS Producer. The target URI identifies the resource to be created. The message body carries the complete resource representation.

2) The MnS Producer returns the HTTP PUT response. On success, "201 Created" shall be returned. The Location header shall carry the URI of the new resource and the message body the complete representation of the new resource. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

5.2 Design pattern for reading a resource

Operations to read the representation of a resource shall be specified with the HTTP GET method. The resource to be read is identified with a URI.

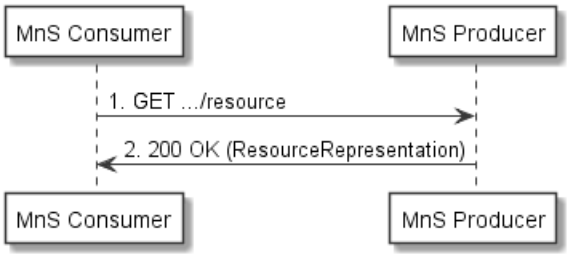

Figure 5.2-1: Flow for reading a resource

The procedure is as follows:

1) The MnS Consumer sends a HTTP GET request to the MnS Producer. The resource to be read is identified with the URI. The message body shall be empty.

   a) If the URI identifies a document resource, the document resource shall be returned.

   b) If the URI identifies a collection resource, all document resources of the collection shall be returned.

2) The MnS Producer returns the HTTP Get response. On success, "200 OK" shall be returned. The resource representation is carried in the response message body. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

5.3 Design pattern for updating a resource

Operations to update the complete representation of a resource shall be specified with the HTTP PUT method. The resource to be updated is identified with a URI.

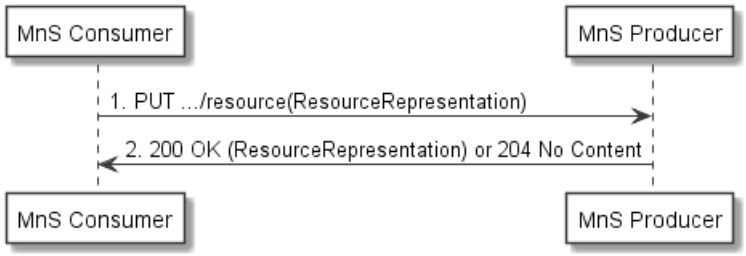

Figure 5.3-1: Flow for updating a resource

The procedure is as follows:

1) The MnS Consumer sends a HTTP PUT request to the MnS Producer. The resource to be updated is identified with the URI. The message body carries the new resource representation. Note, the complete resource representation needs to present.

2) The MnS Producer returns the HTTP PUT response to the MnS Consumer. On success, "200 OK" or "204 No Content" shall be returned. On failure, the appropriate error code shall be returned. The response message body may provide additional error information. In case the resource does not exist, the resource is created in case this is supported (see subclause 5.1.2).

5.4 Design pattern for deleting a resource

Operations to delete the representation of a resource shall be specified with the HTTP DELETE method. The resource to be deleted is identified with a URI in the request message.

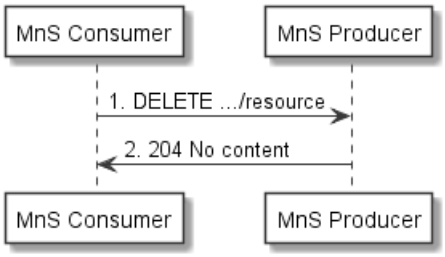

Figure 5.4-1: Flow for deleting a resource

The procedure is as follows:

1) The MnS Consumer sends a HTTP DELETE request to the MnS Producer. The resource to be deleted is identified with the URI. The message body is empty.

2) The MnS Producer returns the HTTP DELETE response to the MnS Consumer. On success, "204 No Content" shall be returned. The message body is empty. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

5.5 Design pattern for subscribe/notify

5.5.1 Concept

HTTP is based on requests and responses. There is no built-in support for notifications and subscriptions to notifications. These mechanisms need to be modelled based on special subscription resources and the available HTTP methods. When notifications are used the server shall expose at least one subscription resource.

5.5.2 Subscription creation

To subscribe to notifications the subscriber shall send a HTTP POST request to the subscription resource.

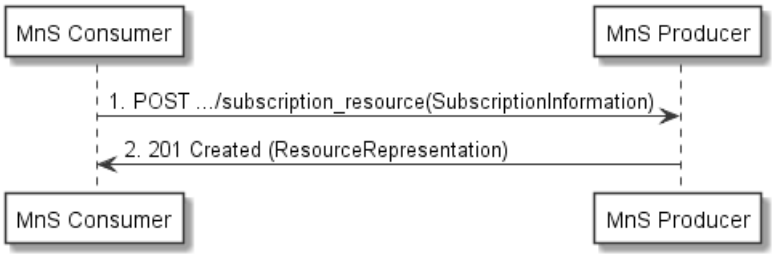

Figure 5.5.2-1: Flow for creating a subscription

The procedure is as follows:

1) The MnS Consumer (notification subscriber) sends a HTTP POST to the MnS Producer. The URI shall indicate a container subscription resource. The resources representing existing subscriptions are created below the container resource. The subscriber shall indicate in the message body the URI of the resource notifications shall be sent to (notification sink) and the type of notifications that are subscribed to. Additional filter information may be included in the message body.

2) The MnS Producer returns "201 Created" on success. The message body carries the representation of the created subscription resource. The Location header shall carry the URI of the created subscription resource. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

5.5.3 Subscription deletion

To cancel a subscription, the subscriber shall delete the corresponding resource with HTTP DELETE.

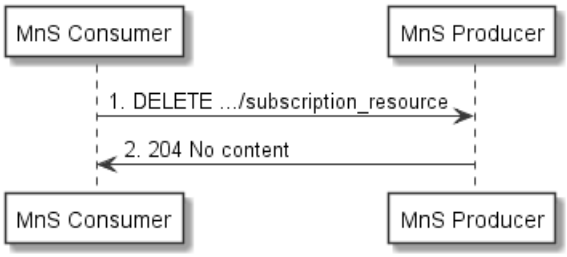

Figure 5.5.3-1: Flow for deleting a subscription

The procedure is as follows:

1) The MnS Consumer (notification subscriber) sends a HTTP DELETE to the MnS Producer. The URI shall indicate the subscription resource to be deleted.

2) The MnS Producer returns the HTTP DELETE response to the MnS Consumer. On success, "204 No Content" shall be returned. The message body is empty. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

5.5.4 Notification emission

To send a notification on the occurrence of a notifiable event the MnS Producer sends a HTTP POST request to the notification sink.

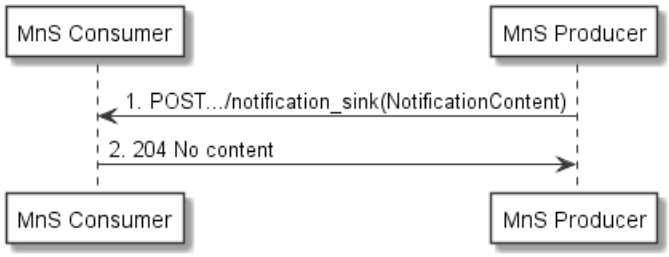

Figure 5.5.4-1: Flow for sending a notification

The procedure is as follows:

1) The MnS Producer sends a HTTP POST to the MnS Consumer. The URI identifies the notification sink. The notification content is included in the message body.

2) The MnS Consumer returns "204 No Content". The message body shall be empty. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

This design pattern requires the MnS Producer (HTTP server) to contain a reduced feature HTTP client for sending HTTP POST requests, and vice versa, the MnS Consumer (HTTP client) to contain a reduced feature HTTP server for receiving HTTP POST requests and sending HTTP POST responses.

5.5.5 Subscription retrieval

The subscriber can retrieve the information about a specific subscription by sending a HTTP GET request to the URI returned by the server upon creation of this subscription. Information about all subscriptions of a subscriber can be read by invoking a HTTP GET on the parent subscription resource whilst instructing the server, using the query component, to return only the subscriptions related to the client invoking the request.

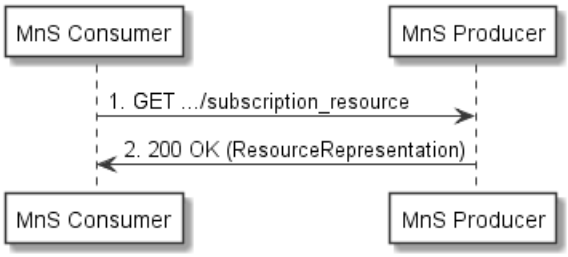

Figure 5.5.5-1: Flow for subscription retrieval

The procedure is as follows:

1) The MnS Consumer sends a HTTP GET to the MnS Producer. The URI specifies the subscription resource to be read.

2) The MnS Producer returns the HTTP Get response. On success, "200 OK" shall be returned. The representation of the subscription resource is carried in the response message body. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

6 Advanced design patterns

6.1 Design pattern for scoping and filtering

Scoping is the process of targeting more than one resource for manipulation with HTTP methods. The URI query component shall be used for scoping resources below the resource identified by the URI path component.

Filtering is the process of selecting a subset of the scoped resources based on filtering criteria applied to the scoped resources. The URI query component shall be used for filtering.

No query language is specified in the present document for scoping and filtering.

6.2 Design pattern for attribute selection

This design pattern allows to select the attributes to be returned by the GET method. This pattern is not applicable to any other HTTP methods.

The attributes to be returned are specified in the query part of the URI with a key value pair. The key is "fields", the value is equal to the attribute names separated by a comma.

6.3 Design pattern for partially updating a resource

HTTP PUT allows replacing only the complete resource. For partial resource updates HTTP PATCH (RFC 5789 [11]) shall be used. The set of changes to be applied to the target resource is described in the request message body (patch document). The format of the patch document is identified by a media type.

RFC 7396 [12] specifies a simple format in JSON (JSON Merge Patch) allowing to describe a set of modifications to be applied to the target resource's content. JSON Merge Patch works at the level of name/value pairs contained in a JSON object. The media type is application/merge-patch+json.

Three types of patches are described in RFC 7396 [12]:

1) Replacing the value of an already existing name/value pair by a new value.

2) Adding a new name/value pair.

3) Removing an existing name/value pair.

Not all three patch types are allowed on all resources.

JSON Merge Patch does not allow manipulation of arrays other than replacing the complete array. It is not possible to change an item in an array or to add a new item.

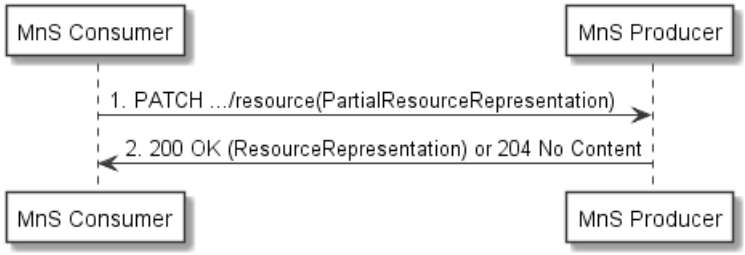

Figure 6.3-1: Flow for partially updating a resource

The procedure flow is as follows:

1) The MnS Consumer sends a HTTP PATCH request to the MnS Producer. The resource to be updated is identified with the URI. The message body carries a set of modification instructions in the form a partial resource representation to be applied to the identified resource.

2) The MnS Producer returns the HTTP PUT response to the MnS Consumer. On success, "200 OK" or "204 No Content" shall be returned. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

7 Resource representation formats

7.1 Introduction

According to clause 4.3 the media type specifies only that JSON is used as resource representation format carried in the HTTP request and HTTP response message bodies. Some resource patterns are quite common and it is desirable to use a common pattern throughout different APIs. This clause identifies some patterns frequently encountered and provides a JSON schema for them.

7.2 Top-level object

A single JSON object shall be at the top-level of the document carried in the message body of HTTP requests and HTTP responses.

```
{"type": "object"}
```

Example:

```
{}
```

Members of the top-level object can be either a data object, a data array or an error object.

7.3 Data objects

Data objects are carried in HTTP requests and in HTTP responses in case of success. One and only one data object shall be a member of a top-level object. If a data object is present, no error object shall be present.

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {}
    }
  }
}
```

Example:

```
{
  "data": {}
}
```

7.4 Data arrays

Data arrays are carried in HTTP requests and in HTTP responses when data is transferred. One and only one data array shall be a member of a top-level object. If a data array is present, no error object shall be present.

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "array",
      "items": {}
    }
  }
}
```

Example JSON instance:

```
{
  "data": []
}
```

7.5 Error objects

Error objects are carried in HTTP responses in case of failure. One and only one error object shall be a member of a top-level object.

```
{
  "type": "object",
  "properties": {
    "error": {
      "type": "object"
      "properties": {}
    }
  }
}
```

Example JSON instance:

```
{
  "error": {}
}
```

7.6 Resource objects

Resource objects are representations of managed object instances. They shall be compliant to the following schema:

```
{
  "type": "object",
  "properties": {
    "href": { "type": "string" },
    "class": { "type": "string" },
    "id": { "type": "string" },
    "attributes": {
      "type": "object",
      "properties": {}
    }
  }
}
```

The attributes object has an empty property object in this generic schema. The attribute properties shall be specified elsewhere. Mandatory attributes shall be indicated with the "required" keyword.

Example JSON instance::

```
{
  "type": "object",
  "properties": {
    "href": { "type": "string" },
    "class": { "type": "string" },
    "id": { "type": "string" },
    "attributes": {
      "type": "object",
      "properties": {
        "attribute1": { "type": "string" },
        "attribute2": { "type": "integer" }
      },
      "required": ["attribute1"]
    }
  }
```

Attribute definitions defined elsewhere are referenced, for example:

```
{
  "type": "object",
  "properties": {
    "href": { "type": "string" },
    "class": { "type": "string" },
    "id": { "type": "string" },
    "attributes": {
      "$ref": "http://3gpp.org/28623/genericNrm.json#definitions/managedElement"
    }
  }
}
```

7.7 Resource objects carried in data objects and arrays

When a resource object is carried in a data object the schema is given by

```
{
  "type": "object",
  "properties": {
    "data": {
        "type": "object",
        "properties": {
          "href": { "type": "string" },
          "class": { "type": "string" },
          "id": { "type": "string" },
          "attributes": {
            "type": "object",
            "properties": {}
          }
        }
    }
  }
}
```

Example JSON instance:

```
{
  "data": {
    "href": "/subnetwork/south/managedElement/6",
    "class": "managedElement",
    "id": "6",
    "attributes": {
      "attribute1": "This is a string.",
      "attribute2": 39
    }
  }
}
```

Multiple resource objects are carried in a data array.

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "array",
```

```
      "items": {
        "type": "object",
        "properties": {
          "href": { "type": "string" },
          "class": { "type": "string" },
          "id": { "type": "string" },
          "attributes": {
            "type": "object",
            "properties": {}
          }
        }
      }
    }
  }
}
```

Example JSON instance:

```
{
  "data": [{
    "href": "/subnetwork/south/managedElement/6",
    "class": "managedElement",
    "id": "6",
    "attributes": {
      "attribute1": "This is a string.",
      "attribute2": 39
    }
  },
  {
    "href": "/subnetwork/south/managedElement/5",
    "class": "managedElement",
    "id": "5",
    "attributes": {
      "attribute1": "This is another string.",
      "attribute2": 139
    }
  }]
}
```

8 REST SS specification template

This clause contains the REST SS specification template.

W Mapping of operations

W.1 Introduction

Table W.1-1: Mapping of IS operations to SS equivalents

IS operation	HTTP Method	Resource URI	Qualifier

W.2 Operation <operation 1>

W.3 Operation <operation 2>

X Usage of HTTP

Y Resources

Y.1 Resource structure

Y.2 Resource definitions

Y.2.1 Resource <resource 1>

Y.2.1.1 Description

Y.2.1.2 URI

Y.2.1.3 HTTP methods

Y.2.1.3.1 <method 1>

This method shall support the URI query parameters specified in table Y.2.1.3.1-1.

Table Y.2.1.3.1-1: URI query parameters supported by the <method 1> on this resource

Name	Data type	P	Cardinality	Description

This method shall support the request data structures specified in table Y.2.1.3.1-2 and the response data structures and response codes specified in table Y.2.1.3.1-3.

Table Y.2.1.3.1-2: Data structures supported by the <method 1> request body on this resource

Data type	P	Cardinality	Description

Table Y.2.1.3.1-3: Data structures supported by the <method 1> response body on this resource

Data type	P	Cardinality	Response codes	Description

Y.2.1.3.2 <method 2>

Y.2.2 Resource <resource 2>

Z Data type definitions

Z.1 General

Table Z.1-1: Data types defined in the present document

Data type	Reference	Description

Table Z.1-2: Data types imported

Data type	Reference	Description

Z.2 Structured data types

Z.2.1 Type <TypeName 1>

Table Z.2.1-1: Definition of type <TypeName 1>

Attribute name	Data type	P	Cardinality	Description

Z.2.2 Type <TypeName 2>

Z.3 Simple data types and enumerations

Z.3.1 General

This subclause defines simple data types and enumerations that are used by the data structures defined in the previous subclauses.

Z.3.2 Simple data types

Table Z.3.2-1: Simple data types

Type Name	Type Definition	Description

Z.3.3 Enumeration <EnumType1>

Table Z.3.3-1: Enumeration < EnumType1>

Enumeration value	Description

Z.3.4 Enumeration <EnumType2>

Annex A (normative)

OpenAPI specification

It contains this leading paragraph:

"This clause describes the capabilities of the service in the structure of the OpenAPI Specification Version 3.0.1 [10]. The OpenAPI document is represented in the JSON format option."

Annex A (informative): Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2018-09	SA#81					Upgrade to change control version	15.0.0
2018-09						Editorial fix (EditHelp/MCC)	15.0.1
2018-12	SA#82	SP-181051	0001	1	F	Extend resource representation format descriptions	15.1.0
2019-06	SA#84	SP-190378	0003	1	F	Correct the DN to URI mapping rules	15.2.0

History

Document history		
V15.0.1	October 2018	Publication
V15.1.0	April 2019	Publication
V15.2.0	June 2019	Publication