ETSI TS 132 158 V16.1.0 (2020-11)

LTE;
5G;
Management and orchestration;
Design rules for REpresentational State Transfer (REST) Solution Sets (SS)
(3GPP TS 32.158 version 16.1.0 Release 16)

Reference

RTS/TSGS-0532158vg10

Keywords

5G,LTE

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".

[13] IETF RFC 6902: "JavaScript Object Notation (JSON) Patch".

[14] IETF RFC 6901: "JavaScript Object Notation (JSON) Pointer".

[15] XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999 (https://www.w3.org/TR/xpath-10/)

[16] 3GPP TR 32.160: "Management and orchestration; Management service template".

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]).

The following JSON patch documents for partial resource modifications may be supported:

- application/merge-patch+json (RFC 7396 [12]).
- application/json-patch+json (RFC 6902 [13]).

This specification defines two new media types for JSON patch documents:

- application/3gpp-merge-patch+json.
- application/3gpp-json-patch+json.

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

4.4 URI structure

4.4.1 Introduction

MnS producers can be divided into two categories. The first category exposes MnS(s) to manipulate resources representing managed object instances. In this case the URI structure is governed by the mapping rules defined in clause 4.2.3. The second category exposes MnS(s) to manipulate resources not representing managed object instances. In this case the DN concept is not relevant. The URI structure for both categories is different.

4.4.2 URI structure for resources representing managed object instances

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

```
{scheme}://{URI-DN-prefix}/{root}/{MnSName}/{MnSVersion}/{URI-LDN}
```

with:

{scheme}	Scheme component "http" or "https"
{URI-DN-prefix}	Authority component (host identifier and optional TCP port), the host name is constructed from the DN prefix as defined in clause 4.2.3.
{root}	Part of the path component, allows specifying optional path segments for structuring the resource hierarchy on a HTTP server.
{MnSName}	Part of the path component, allows specifying an optional MnS name in a single path segment.
{MnSVersion}	Part of the path component, allows specifying an optional MnS version in a single path segment.
{URI-LDN}	Part of the path component, constructed from the LDN as defined in clause 4.2.3, one or more path segments.

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 optional path segments "/{root}/{MnSName}/{MnSVersion}".

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

Note it may be required when specifying a MnS to clearly identify the last RDN of "{URI-LDN}" and to use the following instead of "{URI-LDN}"

```
{URI-LDN-first-part}/{RDN}
```

or

```
{URI-LDN-first-part}/{className}={id}.
```

For the sake of brevity, "MnSRoot" is introduced that includes the "{scheme}" part, the two slash characters ("//"), the "{authority}" part, a single slash character ("/") and the "{root}" part. When using "{MnSRoot}" the abbreviated URI structure is given by

```
{MnSRoot}/{MnSName}/{MnSVersion}/{URI-LDN}
```

or

```
{MnSRoot}/{MnSName}/{MnSVersion}/{URI-LDN-first-part}/{className}={id}
```

It is recommended to use this abbreviated version of the URI structure when defining Management Services.

4.4.3 URI structure for resources not representing managed object instances

URIs identifying other resources shall follow a structure given by

```
{scheme}://{authority}/{root}/{MnSName}/{MnSVersion}/{MnSResourcePath}
```

with:

{scheme}	Scheme component "http" or "https"
{authority}	Authority component (host identifier and optional TCP port)
{root}	Part of the path component, allows specifying optional path segments for structuring the resource hierarchy on a HTTP server.
{MnSName}	Part of the path component, specifies the mandatory MnS name in a single path segment.
{MnSVersion}	Part of the path component, specifies the mandatory MnS version in a single path segment.
{MnSResourcePath}	Part of the path component, one or more path segments, specifies a resource of the MnS

For the sake of brevity, {MnSRoot} is introduced that includes the "{scheme}" part, the two slash characters ("//"), the "{authority}" part, a single slash character ("/") and the "{root}" part. When using "{MnSRoot}" the abbreviated URI structure is given by

```
{MnSRoot}/{MnSName}/{MnSVersion}/{MnSResourcePath}
```

It is recommended to use this abbreviated version of the URI structure when defining Management Services

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.

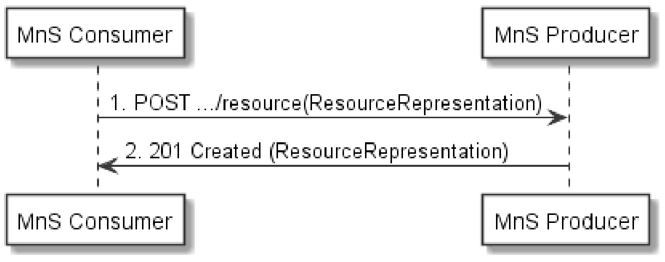
MnS Consumer
MnS Producer
1. POST .../resource(ResourceRepresentation)
2. 201 Created (ResourceRepresentation)
MnS Consumer
MnS Producer

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. The message body shall carry a complete resource representation. In case the resources representation format mandates the presence of a resource identifier it shall carry null semantics. If the identifier carries nevertheless a value, the MnS Producer may consider that as a non-binding recommendation by the MnS Consumer.

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 a new path segment containing this identifier to the request URI. The message body should 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.

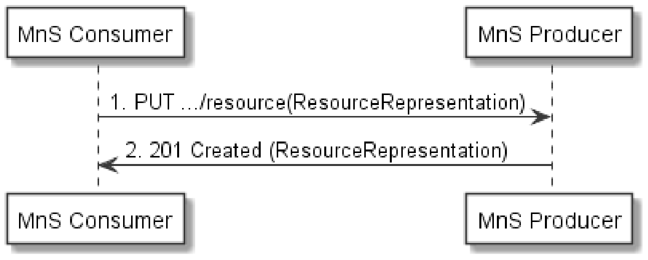
MnS Consumer
MnS Producer
1. PUT .../resource(ResourceRepresentation)
2. 201 Created (ResourceRepresentation)
MnS Consumer
MnS Producer

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 location of 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. The message body should contain 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.

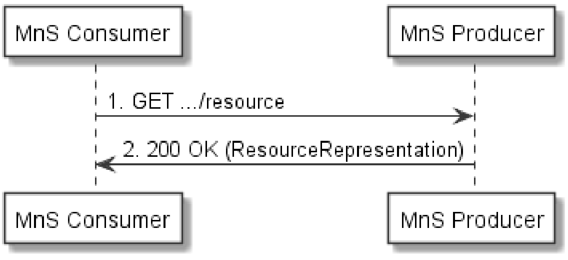

Figure 5.2-1: Flow for reading a resource

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 the target URI.

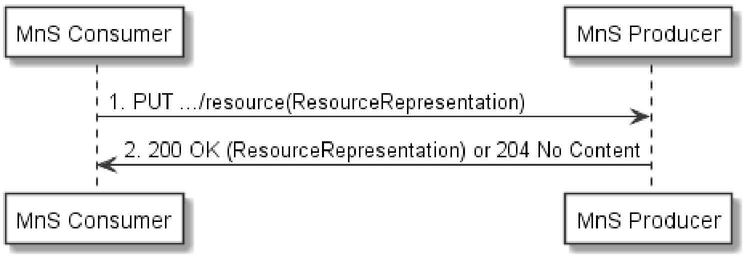
MnS Consumer
MnS Producer
1. PUT .../resource(ResourceRepresentation)
2. 200 OK (ResourceRepresentation) or 204 No Content
MnS Consumer
MnS Producer

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 target URI. The message body carries the representation the target resource shall replaced with. Note, the complete resource representation needs to included in the body. Partial representations of the resource to be updated are not allowed.

2) The MnS Producer returns the HTTP PUT response to the MnS Consumer. On success, "200 OK" or "204 No Content" shall be returned. In the former case the response carries the representation of the updated resource in the message body. In the latter case the response has no message body. A "200 OK" response including the representation of the updated resource shall be sent when the updated representation of the resource is not identical to the representation received in the request. 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 the target URI in the request message.

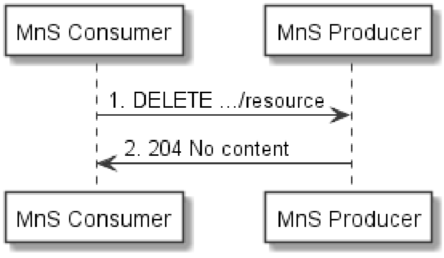

Figure 5.4-1: Flow for deleting a resource

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.

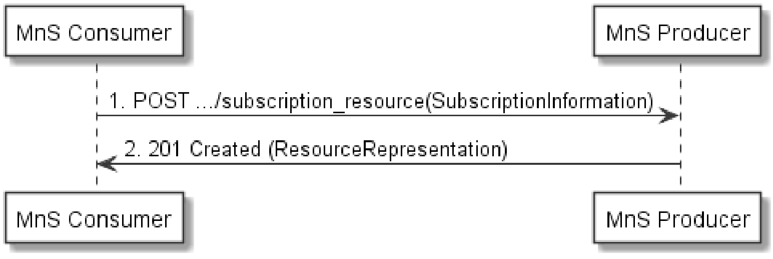
MnS Consumer
MnS Producer
1. POST .../subscription_resource(SubscriptionInformation)
2. 201 Created (ResourceRepresentation)
MnS Consumer
MnS Producer

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.

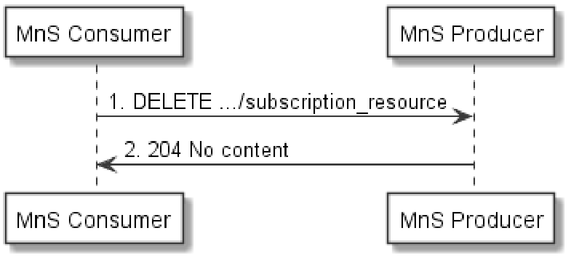

Figure 5.5.3-1: Flow for deleting a subscription

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.

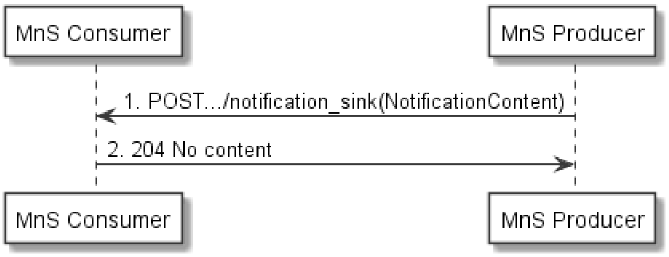

Figure 5.5.4-1: Flow for sending a notification

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.

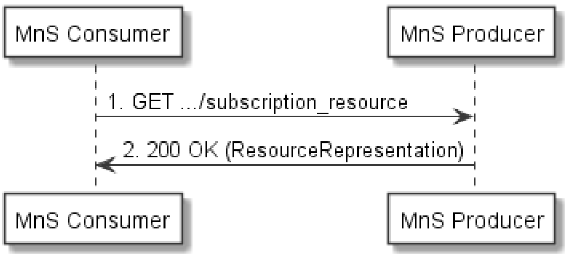
MnS Consumer
MnS Producer
1. GET .../subscription_resource
2. 200 OK (ResourceRepresentation)
MnS Consumer
MnS Producer

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

6.1.1 Introduction

In stage 2 specifications a scope construct is often used for selecting multiple managed object instances. The scope construct, together with a so called base managed object instance, selects a set of object instances from the name-containment tree starting at the document root. This set contains some or all object instances name-contained by the base object instance. It may contain the base object itself.

In operations, the base object instance and the scope construct are specified as an input parameter. In NRM control fragments, the base object instance is the object instance that name-contains the control object instance of the NRM control fragment, and the scope construct is an attribute of the control object instance.

A filter construct is also often used in stage 2 specifications to select a subset of the managed object instances selected by the base managed object instance and scope construct. The filter is specified in operations as input parameter and in NRM control fragments as an attribute of a control object.

When scoping and filtering is specified using NRM control fragments, no special considerations are required for the REST SS, since the scope construct and the filter are normal attributes of a managed object.

When scoping and filtering is specified as part of the input parameters of an operation, however, it is necessary to define how to map these parameters in the REST SS.

6.1.2 Query parameters for scoping

Scoping may be supported by the HTTP GET method or the HTTP DELETE method. It is not supported by any other method.

The URI path component identifies the base resource. The URI query component shall be used for carrying the scope construct. Multiple query parameters shall be separated by an ampersand character ("&").

With one query parameter the base resource and all resources until the level indicated by the query parameter can be selected. When the value of the query parameter is set to inifinite, the complete subtree starting at the base resource is selected.

Two query parameters for scoping allow for more sophisticated selection methods.

An example scoping method uses a "scopeType" and a "scopeLevel" query parameter. The allowed values are defined in Table 6.1.2-1.

Table 6.1.2-1: Allowed values of the "scopeType" query parameter

Value	Description
BASE_ONLY	Selects only the base resource. The "scopeLevel" parameter shall be absent or ignored if present.
BASE_ALL	Selects the base resource and all of its subordinate resources (incl. the leaf resources). The "scopeLevel" parameter shall be absent or ignored if present.
BASE_NTH_LEVEL	Selects all resources on the level, which is indicated by the "scopeLevel" parameter, below the base resource. The base resource is at "scopeLevel" zero.
BASE_SUBTREE	Selects the base resource and all of its subordinate resources down to and including the resources on the level indicated by the "scopeLevel" parameter. The base resource is at "scopeLevel" zero.

6.1.3 Query parameters for filtering

Filtering may be supported by the HTTP GET method or the HTTP DELETE method. It is not supported by any other method.

The URI query component shall be used for carrying the filter construct.

XPath 1.0 [15] shall be used for specifying the filter construct. The context resource for the XPath path expression is the resource identified by the targrt URI of the GET request.

A valid XPath expression returns a flat list of selected resources. Name-contained resources included in the selected resources shall be removed before constructing the response message according to clause 6.1.1.

The name of the query parameter shall be "filter".

6.1.4 Construction rules for the response message body

When multiple resources are selected for retrieval by HTTP GET, the respone message body with the selected resource set shall be constructed according to one of the following rules.

Flat response construction method: The resources are basically returned as a flat list of JSON objects. Their location in the hierarchical containment tree needs to be specified by e.g. their URI which needs to be returned for each resource.

Hierarchical response construction method: The resources are returned inside the containment tree as specified by the JSON schema definition of the information model. The resources not selected are either not returned at all or returned empty, except for the resource identifiers, when their presence is required in the containment tree. The containment tree present in the response message shall always start with the root resource of the information model (document root) or the base resource.

6.2 Design pattern for attribute and attribute field selection

6.2.1 Introduction

This design pattern allows to specify attributes of resources selected by the target URI and scoping and filtering.

Often attributes have no scalar values but are complex structured data types with an own hierarchy. In this case it may be desirable to identify not only the complete attribute but also attribute fields.

The attributes or attribute fields to be returned shall be specified in the query part of the URI.

Attribute selection or attribute field selection may be supported by the HTTP GET method. It is not applicable to any other method.

For constructing the response not selected attributes and attribute fields are removed from the resource representation.

6.2.2 Query parameters for attribute and attribute field selection

In case only complete attributes are retrieved the name of the query parameter shall be "attributes". The value of "attributes" shall be a list with the names of the attributes to be selected. Attribute names are separated by a comma (","). An empty "attributes" query parameter is allowed and has the special meaning that no attributes shall be returned, except for the naming attribute "id".

In case it shall be possible to select attribute fields the syntax of JSON Pointer in JSON String Representation [14] shall be used. The context resource for the construction of the JSON Pointer is the resource identified by the target URI. When multiple attribute fields shall be selected the corresponding JSON Pointer String Representations shall be separated by a comma (","). The name of the query parameter shall be "fields".

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 its 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.

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

When individual items of an array shall be manipulated or items shall be added to arrays at specific positions, JSON Patch as described in RFC 6902 [13]) should be used as patch format. The media type of JSON Patch is "application/json-patch+json". The target URI identifies the resource to be modified. Secondary resources of the target resource to be manipulated are identified in the JSON patch document using JSON Pointer [14].

The JSON Patch document is a JSON array with each item being a JSON object specifying one suboperation. Suboperations shall be applied sequentially in the order they appear in the array, as defined in Section 3 of RFC 6902 [13].

According to RFC 5789 [11], Section 2 patches shall be applied atomically. Either all changes specified in the patch document are applied or, if at least one change cannot be applied, no change shall be applied.

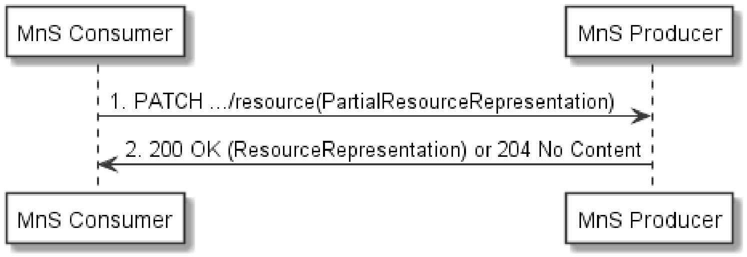
MnS Consumer
MnS Producer
1. PATCH .../resource(PartialResourceRepresentation)
2. 200 OK (ResourceRepresentation) or 204 No Content
MnS Consumer
MnS Producer

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 target URI. The message body carries a JSON Patch or JSON Merge Patch document describing a set of modification instructions to be applied to the target resource.

2) The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated resource in the message body 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.

6.4 Design pattern for patching multiple resources

6.4.1 Introduction

Clause 6.1 discusses a method for retrieving multiple resources with a single GET request. This clause presents methods allowing to manipulate (create, update, delete) multiple resources with a single PATCH request.

6.4.2 3GPP JSON Merge Patch

3GPP JSON Merge Patch is a 3GPP defined extension to JSON Merge Patch (RFC 6902 [13]) allowing to manipulate individual items in an array supposed each item has an identifier that is unique within the name space of the array. The identifier of an array item has to be present in any 3GPP JSON Merge Patch document. This patch format allows to update attributes and attribute fields, to create resources with "id" creation by the MnS Consumer, and to delete resources.

The target URI shall identify the resource that is the first common parent resource of the resources to be manipulated or the document root.

Resources are deleted by setting all NRM attributes to the "null" value. If the NRM attributes are members of a special "attributes" object this object shall be set to "null".

A 3GPP JSON Merge Patch document is applied in atomic manner (RFC 5789 [11]). Either all changes are applied or, if at least one mofification cannot be applied, no change shall be applied. 3GPP JSON Merge Patch thus has transaction semantics.

The procedure is as follows:

1) The MnS Consumer sends a HTTP PATCH request to the MnS Producer. The message body carries a 3GPP JSON Merge Patch document describing a set of modification instructions to be applied to the identified resources.

2) The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated resources, constructed according to the hierarchical response construction method described in clause 6.1.1, in the message body 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.

The media type of 3GPP JSON Merge Patch is "3gpp-merge-patch+json". This media type is defined by 3GPP and is not registered with IANA. Patch documents using this media type must conform to the "application/json" media type.

6.4.3 3GPP JSON Patch

JSON Patch (RFC 6902 [13]) allows to manipulate multiple secondary resources of the target resource. The target resource is identified by the target URI. The secondary resources are specified with JSON Pointers included in the JSON Patch document enclosed in the HTTP Patch method request message body. The context object of JSON Pointer is the target resource. It is not possible to point to resources or secondary resources outside of the target resource.

Each suboperation is specified in a JSON Patch document by a JSON object whose property names are "op", "from", "path" and "value". Not all operations require all properties. The type of the "from" and "path" property value is a JSON Pointer in string representation as defined in Section 5 of RFC 6901 [14].

The present document defines an extentension to JSON Patch allowing to specify resources and secondary resources outside of the target resource. With this extension a single HTTP Patch request can manipulate multiple resources.

The extension is that the "path" and "from" properties define an offset to the target resource as specified by the request URI in the HTTP PATCH method. This offset has a first component pointing to a resource below the targert resource, and a sescond component pointing to secondary a resource within the resource identified by the first component. The first component of "path" or "from" is built from URI path components. The second component is a URI fragment with a JSON pointer in the URI fragment identifier representation as defined in clause 6 of of RFC 6901 [14]. An empty value of the "path" or "from" property ("") means that the resource specified by the request URI in the HTTP PATCH method is the target for the value of the patch "value" property.

The target URI can identify the document root, the first common parent resource of the resources to be manipulated or any resource between them.

The media type of 3GPP JSON Merge Patch is "3gpp-patch+json". This media type is defined by 3GPP and is not registered with IANA. Patch documents using this media type must conform to the "application/json" media type.

As all other patch media types, 3GPP JSON Patch shall be applied in atomic manner.

The procedure is as follows:

1) The MnS Consumer sends a HTTP PATCH request to the MnS Producer. The message body carries a JSON Patch document describing a set of modification instructions to be applied to the identified resources.

2) The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated resources, constructed according to the hierarchical response construction method described in clause 6.1.1, in the message body 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 (resources) are representations of managed object instances. They shall be compliant to the following JSON schema when one instance of a class is allowed.

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

or by the following schema when more than one instance of a class is allowed

```
{
  "type": "object",
  "properties": {
    "ClassName": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "href": { "type": "string" },
          "class": { "type": "string" },
          "id": { "type": "string" },
          "attributes": {
            "type": "object",
            "properties": {}
          }
        },
        "required": ["id"]
      }
    }
  }
}
```

An object, whose name is equal to the NRM class name, encapsulates the resource representation.

The "attributes" object contains NRM attributes as properties. In the generic schema above the "attributes" object has no properties. These properties are defined in other specifications.

Only the "id" is required to be always present. The "href" property with the URI of the resource and the "class" property with the name of the NRM class can be omitted, or not specified at all in concrete JSON schemas for resource representations.

TS 32.160 [16] specifies the complete mapping of stage 2 NRM definitions to stage 3 JSON schema definitions.

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

Multiple instance of the same NRM class are supported by a JSON array.

```
{
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "ClassName": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "href": { "type": "string" },
              "class": { "type": "string" },
              "id": { "type": "string" },
              "attributes": {
                "type": "object",
                "properties": {}
              }
            },
            "required": ["id"]
          }
        }
      }
    }
  }
}
```

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): Examples

A.1 Example information model

The following JSON instance document is used for the examples in this chapter.

```
{
  "SubNetwork": {
    "id": "SN1",
    "attributes": {
      "userLabel": "Berlin NW",
      "userDefinedNetworkType": "5G",
      "plmn-id": {
        "mcc": 456,
        "mnc": 789
      }
    },
    "ManagedElement": [
      {
        "id": "ME1",
        "attributes": {
          "userLabel": "Berlin NW 1",
          "vendorname": "Company XY",
          "location": "TV Tower"
        },
        "XyzFunction": [
          {
            "id": "XYZF1",
            "attributes": {
              "attrA": "xyz",
              "attrB": 551
            }
          },
          {
            "id": "XYZF2",
            "attributes": {
              "attrA": "abc",
              "attrB": 552
            }
          }
        ]
      },
      {
        "id": "ME2",
        "attributes": {
          "userLabel": "Berlin NW 2",
          "vendorname": "Company XY",
          "location": "Grunewald"
        }
      }
    ]
  }
}
```

The corresponding JSON schema is

```
{
  "SubNetwork": {
    "type": "object",
    "properties": {
      "id": {
        "type": "string"
      },
      "attributes": {
        "type": "object",
        "properties": {
          "userLabel": "string",
          "userDefinedNetworkType": "string",
          "plmn-id": {
            "type": "object",
            "properties": {
              "mcc": "integer",
              "mnc": "integer"
            }
          }
        }
      },
      "ManagedElement": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "id": {
              "type": "string"
            },
            "attributes": {
              "type": "object",
              "properties": {
                "userLabel": {
                  "type": "string"
                },
                "vendorname": {
                  "type": "string"
                },
                "location": {
                  "type": "string"
                }
              }
            },
            "XyzFunction": {
              "type": "array",
              "items": {
                "type": "object",
                "properties": {
                  "id": {
                    "type": "string"
                  },
                  "attributes": {
                    "type": "object",
                    "properties": {
                      "attributeA": {
                        "type": "string"
                      },
                      "attributeB": {
                        "type": "integer"
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

NOTE: the following examples do not follow the URI structure specified in clause 4.4 for simplicity reasons. The "data" object in responses is omitted as well.

A.2 Retrieval of resources

A.2.1 Retrieval of a single complete resource with HTTP GET

To retrieve a complete "YxzFunction"resource the MnS Consumer might send the following request.

```
GET /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Accept: application/json
```

The response includes the resource representation

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "id": "XYZF1",
  "attributes": {
    "attrA": "xyz",
    "attrB": 551
  }
}
```

and might include a key ("XyzFunction") specifying the class name of the returned resource

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "XyzFunction": [
    {
      "id": "XYZF1",
      "attributes": {
        "attrA": "xyz",
        "attrB": 551
      }
    }
  ]
}
```

In the example above "XyzFunction" is of type array to align with the JSON schema of "XyzFunction" defined in clause A.1. "XyzFunction" may also be an object, since the schema specifing the message body is not required to be identical to the schema specifying the resources contained by another resource.

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "XyzFunction": {
    "id": "XYZF1",
    "attributes": {
      "attrA": "xyz",
      "attrB": 551
    }
  }
}
```

When using a "data" object the response might look like

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "data": {
    "XyzFunction": {
      "id": "XYZF1",
      "attributes": {
        "attrA": "xyz",
        "attrB": 551
      }
    }
  }
}
```

The exact syntax of the response body is specified by the JSON schema included in the concrete API definition.

A.2.2 Attribute and attribute field selection on a single resource

To retrieve only the "userLabel" attribute and the "mcc" attribute field of the "plmn-id" attribute the MnS Consumer might send

```
GET /SubNetwork=SN1?fields=attributes/userLabel,attributes/plmn-id/mcc HTTP/1.1
Host: example.org
Accept: application/json
```

Alternatively one might send as well

```
GET /SubNetwork=SN1?attributes=userLabel&fields=attributes/plmn-id/mcc HTTP/1.1
Host: example.org
Accept: application/json
```

The response contains only the selected attribute and the selected attribute field.

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "SubNetwork": {
    "id": "SN1",
    "attributes": {
      "userLabel": "Berlin NW",
      "plmn-id": {
        "mnc": 789
      }
    }
  }
}
```

A.2.3 Retrieval of multiple complete resources using scoping and filtering

The following example selects all "ManageElement" nodes with a "vendorname" of "Company XY".

```
GET /SubNetwork=SN1?scope=BASE_ALL&\
filter=/SubNetwork/ManagedElement/attributes[vendorname="Company XY"]/parent::node() HTTP/1.1
Host: example.org
Accept: application/json
```

Alternatively, the following XPath expression can be used.

```
GET /SubNetwork=SN1?scope=BASE_ALL&\
filter=/SubNetwork/ManagedElement[attributes/vendorname="Company XY"] HTTP/1.1
Host: example.org
Accept: application/json
```

When using the hierarchical response construction method the response looks as follows

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "SubNetwork": {
    "id": "SN1",
    "ManagedElement": [
      {
        "id": "ME1",
        "attributes": {
          "userLabel": "Berlin NW 1",
          "vendorname": "Company XY",
          "location": "TV Tower"
        }
      },
      {
        "id": "ME2",
        "attributes": {
          "userLabel": "Berlin NW 2",
          "vendorname": "Company XY",
          "location": "Grunewald"
        }
      }
    ]
  }
}
```

The following example returns the containment tree only

```
GET /SubNetwork=SN1?scope=BASE_ALL&attributes= HTTP/1.1
Host: example.org
Accept: application/json
```

```


HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json

{
  "SubNetwork": {
    "id": "SN1",
    "ManagedElement": [
      {
        "id": "ME1",
        "XyzFunction": [
          {
            "id": "XYZF1"
          },
          {
            "id": "XYZF2"
          }
        ]
      },
      {
        "id": "ME2"
      }
    ]
  }
}
```

A.3 Creation of resources

A.3.1 Creation of a resource with HTTP PUT

In this example a new "XyzFunction" resource is created. The target URI specifies the location of the new resource. The "id" of the new resource is "XYZF1" and created by the MnS Consumer.

```
PUT /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Content-Type: application/json

{
  "XyzFunction": [
    {
      "id": "XYZF1",
      "attributes": {
        "attrA": "xyz",
        "attrB": 551
      }
    }
  ]
}
```

The response contains the location header and the complete representation of the new resource.

```
HTTP/1.1 201 Created
Date: Tue, 06 Aug 2019 16:50:26 GMT
Location: http://example.org/ SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1
Content-Type: application/json

{
  "XyzFunction": [
    {
      "id": "XYZF1",
      "attributes": {
        "attrA": "xyz",
        "attrB": 551
      }
    }
  ]
}
```

A.3.2 Creation of a resource with HTTP POST

When creating a new resource with POST the target URI identifies the parent resource of the new resource to be created. The identifier of the new resource is created by the MnS Producer, hence the "id" is equal to "null" in the POST request. If the "id" carries a value, then the MnS Producer may consider that as a non-binding recommendation by the MnS Consumer.

```
POST /SubNetwork=SN1/ManagedElement=ME1 HTTP/1.1
Host: example.org
Content-Type: application/json

{
  "XyzFunction": [
    {
      "id": "null",
      "attributes": {
        "attrA": "xyz",
         "attrB": 551
      }
    }
  ]
}
```

```
HTTP/1.1 201 Created
Date: Tue, 06 Aug 2019 16:50:26 GMT
Location: http://example.org/ SubNetwork=SN1/ManagedElement=ME1/XyzFunction=123e4567-e89b
Content-Type: application/json

{
  "XyzFunction": [
    {
      "id": "123e4567-e89b",
      "attributes": {
        "attrA": "xyz",
        "attrB": 551
      }
    }
  ]
}
```

A.3.3 Creation of a resource with JSON Patch

This example shows the creation of a resource with JSON Patch. The target URI identifies the resource to be created. The "path" property of the patch is empty ("").

```
PATCH /SubNetwork=SN1/ManagedElement=ME1 HTTP/1.1
```

```
Host: example.org
Content-Type: application/json-patch+json

[
  {
  "op": "add",
    "path": "",
    "value": {
      "id": "ME1",
      "class": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 1",
        "vendorname": "Company XY",
        "location": "TV Tower"
      }
    }
  }
]
```

A.4 Deletion of resources

A.4.1 Deletion of a resource with HTTP DELETE

The following example deletes an instance of "ManagedElement".

```
DELETE /SubNetwork=SN1/ManagedElement=ME2 HTTP/1.1
Host: example.org
```

```
HTTP/1.1 204 No Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
```

A.4.2 Deletion of multiple resources with HTTP DELETE

This example deletes both "XyzFunction" resources.

```
DELETE /SubNetwork=SN1?scopeType= BASE_NTH_LEVEL&scopeLevel=2 HTTP/1.1
Host: example.org
```

```
HTTP/1.1 204 No Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
```

A.4.3 Deletion of a resource with JSON Patch

The following example deletes an instance of "ManagedElement". The target URI identifies the resource to be deleted. The "path" property of the patch is empty ("").

```
PATCH /SubNetwork=SN1/ManagedElement=ME1 HTTP/1.1
Host: example.org
Content-Type: application/json-patch+json

[
  {
    "op": "remove",
    "path": ""
  }
]
```

A.5 Complete update of a resource

The following example updates a "XyzFunction" resource. Only the "attrA" attribute is updated with a new value. The "attrB" attribute is set to the old value, but still the "attrB" attribute needs to be present.

```
PUT /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Content-Type: application/merge-patch+json

{
  "XyzFunction": [
    {
      "id": "XYZF1",
      "attributes": {
        "attrA": "newValue",
        "attrB": 551
      }
    }
  ]
}
```

A.6 Partial update of a resource

A.6.1 Partial update of a resource with JSON Merge Patch

The first example shows how the attribute "attrA" of the "XyzFunction with the "id" equal to "YXZF1" is changed from "xyz" to "def" using JSON Merge Patch.

```
PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Content-Type: application/merge-patch+json

{
  "XyzFunction": {
    "id": "XYZF1",
    "attributes": {
      "attrA": "def"
    }
  }
}
```

In the second example the "mcc" attribute field of the "plmnId" attribute is updated to "654". The employed patch method is again JSON Merge Patch.

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/merge-patch+json

{
  "SubNetwork": {
    "id": "SN1",
    "attributes": {
      "plmn-Id": {
        "mcc": 654
      }
    }
  }
}
```

Note that the value of "SubNetwork" needs to be a JSON object when using standard JSON Merge Patch. JSON arrays are not allowed. This needs to be taken into account when specifying the JSON schema for the PATCH method request body.

A.6.2 Partial update of a resource with 3GPP JSON Merge Patch

In these examples the same changes as in clause A.6.1 are requested, but 3GPP JSON Merge Patch is used. The value of "XyzFunction" can be an array in this case.

```
PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
```

```
Content-Type: application/3gpp-merge-patch+json

{
  "XyzFunction": [
    {
      "id": "XYZF1",
      "attributes": {
        "attrA": "def"
      }
    }
  ]
}
```

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-merge-patch+json

{
  "SubNetwork": {
    "id": "SN1",
    "attributes": {
      "plmn-Id": {
        "mcc": 654
      }
    }
  }
}
```

A.6.3 Partial update of a resource with JSON Patch

When JSON Patch is used to request the changes described in clause A.6.1, the MnS consumer may send

```
PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Content-Type: application/json-patch+json

[
  {
    "op": "replace",
    "path": "/attributes/attrA",
    "value": 654
  }
]
```

and

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/json-patch+json

[
  {
    "op": "replace",
    "path": "/attributes/plmn-Id/mcc",
    "value": 654
  }
]
```

A.6.4 Partial update of a resource with 3GPP JSON Patch

When 3GPP JSON Patch is used to request the changes described in clause A.6.1 the MnS consumer may send the following

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-json-patch+json

[
  {
    "op": "replace",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF1#attributes/attrA",
    "value": 654
  }
]
```

and

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-json-patch+json

[
  {
    "op": "replace",
    "path": "#attributes/plmn-Id/mcc",
    "value": 654
  }
]
```

In the first example the target URI of the HTTP PATCH method identifies the document root.

The value of "path" in the second example is just the URI fragment component beginning with "#".

A.7 Manipulating multiple resources

A.7.1 Manipulating multiple resources with 3GPP JSON Merge Patch

In this example the "userLabel" attribute and the "mcc" attribute field of the "subNetwork" resource is updated. A new "XyzFunction" resource is created as well as a new "ManagedElement" resource.

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-merge-patch+json

{
  "SubNetwork": {
    "id": "SN1",
    "attributes": {
      "userLabel": "Berlin NW-1",
      "plmn-id": {
        "mcc": 456
      }
    },
    "ManagedElement": [
      {
        "id": "ME1",
        "XyzFunction": [
          {
            "id": "XYZF3",
            "attributes": {
              "attrA": "fgh",
              "attrB": 555
            }
          }
        ]
      },
      {
        "id": "ME3",
        "attributes": {
          "userLabel": " Berlin NW 3",
          "vendorname": "Company XY",
          "location": "Spandau"
        }
      }
    ]
  }
}
```

In the following example a "XYzFunction" resource is deleted.

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-merge-patch+json

{
  "SubNetwork": {
    "id": "SN1",
    "ManagedElement": [
      {
        "id": "ME1",
        "XyzFunction": [
          {
            "id": "XYZF2",
            "attributes": null
          }
        ]
      }
    ]
  }
}
```

A.7.2 Manipulating multiple resources with 3GPP JSON PATCH

The same resource modifications as in the previous chapter expressed using JSON Patch are given by

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-json-patch+json

[
  {
    "op": "replace",
    "path": "/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1#/attributes/userLabel",
    "value": "Berlin NW-1"
  },
  {
    "op": "replace",
    "path": "/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1#/attributes/plmn-id/mcc",
    "value": 654
  },
  {
    "op": "add",
    "path": "/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF3",
    "value": {
      "id": "XYZF3",
      "attributes": {
        "attrA": "fgh",
        "attrB": 555
      }
    }
  },
  {
    "op": "add",
    "path": "/SubNetwork=SN1/ManagedElement=ME3",
    "value": {
      "id": "ME3",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorname": "Company XY",
        "location": "Spandau"
      }
    }
  }
]
```

Annex B (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
2019-12	SA#86	SP-191220	0004	3	F	Clarify design pattern for scoping and filtering	15.3.0
2019-12	SA#86	SP-191220	0005	-	F	Correct basic design patterns	15.3.0
2019-12	SA#86	SP-191220	0006	-	F	Add design pattern for patching multiple resources	15.3.0
2019-12	SA#86	SP-191220	0007	-	F	Correct resource representation formats	15.3.0
2019-12	SA#86	SP-191220	0008	-	F	Add examples	15.3.0
2019-12	SA#86	SP-191220	0010	2	F	Clarify design pattern for attribute field selection	15.3.0
2020-03	SA#87E	SP-200183	0011	1	F	Clarify HTTP PATCH methods	15.4.0
2020-07	SA#88E	SP-200504	0012	2	F	Add the missing definition for LDN-first-part	15.5.0
2020-07	-	-	-	-	-	Update to Rel-16 version (MCC)	16.0.0
2020-09	SA#89E	SP-200813	0015	1	F	Update the URI structure definition	16.1.0

History

Document history		
V16.0.0	August 2020	Publication
V16.1.0	November 2020	Publication