ETSI TS 132 158 V17.1.0 (2022-07)

LTE ;
5G ;
Management and orchestration;
Design rules for REpresentational State Transfer (REST)
Solution Sets (SS)
(3GPP TS 32.158 version 17.1.0 Release 17)

Reference
RTS/TSGS-0532158vh10

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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

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".

[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. Each object class contains information elements called attributes. Relationships between classes represent the logical connections. Models are specified formally with class diagrams produced using 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.

Besides this primary resource, RFC 3986 [4], clause 3.5 introduces the concept of secondary resource. Secondary resources are specific portions or subsets of primary resources, that are identifiable.

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 (primary) document resources. Collection resources have no equivalent in an information model unless some dedicated collection class is introduced.

Attributes are conceptually mapped to secondary resources.

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.

4.4.4 Resource "../{MnSName}/{MnSVersion}"

This resource represents the conceptual parent of the top-level managed object instances. It is created by the MnS Producer. A MnS Consumer cannot create or delete this resource.

The resource is the target resource for many HTTP requests, such as requests to retrieve all top-level managed object instances in case there are multiple top-level managed object instances, or for requests to create objects in case there are no manged object instances yet and the creation request needs to be directed to the parent of the resource to be created.

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 (single) resource shall be specified with the HTTP POST method, when the MnS Producer shall create the identifier of the new resource.

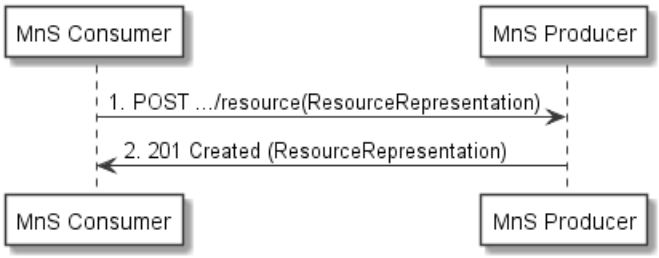

Figure 5.1.1-1: Flow for creating a resource with HTTP POST

The procedure is as follows:

1) The MnS Consumer sends an HTTP POST request to the MnS Producer. The target URI identifies the parent resource below which the new resource shall be created. The target URI shall have no query and no fragment component. The message body shall carry a representation of the resource to be created. The resource representation shall not contain the identifier of the new resource, unless the resource representation format mandates the presence of a resource identifier in which case 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. The object class name of the resource to be created shall be specified in the message body as well.

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 shall be 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 representation of the new resource. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

The resource representation in the request and response message may not be identical, and may not contain all properties (attributes) that are defined in a schema specifying the format of the representation.

For example, assume the schema for the representation of the resource defines the attributes "attrA", "attrB" and "attrC". When the MnS Consumer has valid values only for the attributes "attrA" and "attrB", then the representation sent to the MnS Producer shall include only these two attributes. When the MnS Producer has no valid value for "attrC" and no default value is defined for attrC, then the response is identical to the request, and a subsequent HTTP GET request for all attributes returns only a representation with the attributes "attrA" and "attrB", but not with the attribute "attrC". However, if the MnS Producer populates "attrC" with some value or a default value is defined for attrC, then the HTTP POST response shall include all three attributes. Likewise, a subsequent HTTP GET request for all attributes returns all three attributes.

A MnS Producer may also modify attribute values included in the request. In this case, the modified values shall be sent back to the MnS Cosumer.

It is also possible that a MnS Producer removes attributes received in the request and includes only a subset of the received attributes in the response.

When the created resource has child resources that are included in the schema definition of the created resource, a representation of these child resources shall neither be included in the resource representation sent to the MnS Producer nor in the resource represententation returned to the MnS Consumer. Including child resources would be an attempt to create multiple resources with a single request. HTTP POST shall be used for the creation of a single resource only.

5.1.2 Creating a resource with identifier creation by the MnS Consumer

Operations to create a (single) resource shall be specified with the HTTP PUT method, when the MnS Consumer creates the identifier of the new resource.

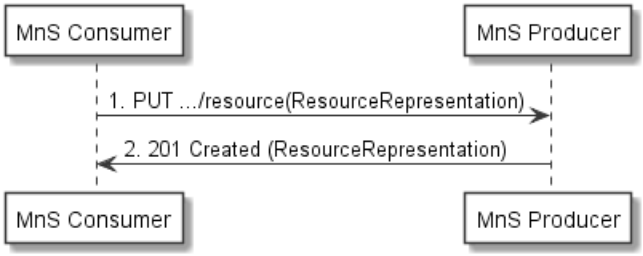

Figure 5.1.2-1: Flow for creating a resource with HTTP PUT

The procedure is as follows:

1) The MnS Consumer sends an HTTP PUT request to the MnS Producer. The target URI identifies the location of the resource to be created. The target URI shall have no query and no fragment component. The message body shall carry the representation of the resource to be created. The representation shall include the identifier and object class name of the new resource.

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 representation of the new resource. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

As for resource creation with HTTP POST, the resource representation in the request and response message may not be identical and may not contain all properties (attributes) that may be defined in a schema specifying the format of the representation. Also, just like for resource creation with HTTP POST, the resource representation sent to the MnS Producer or returned to the MnS Consumer shall not contain the representation of any child resources of the resource to be created.

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.

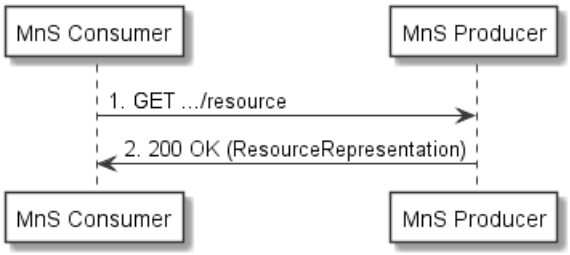

Figure 5.2-1: Flow for reading a resource

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 (single) resource shall be specified with the HTTP PUT method. The resource to be updated is identified with the target URI.

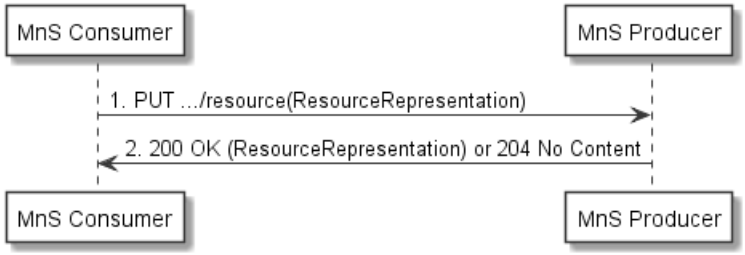

Figure 5.3-1: Flow for updating a resource

The procedure is as follows:

1) The MnS Consumer sends an HTTP PUT request to the MnS Producer. The resource to be updated is identified with the target URI. The target URI shall have no query and no fragment component. The message body carries the new representation that shall completely replace the existing resource representation on the MnS Producer.

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 shall carry the representation of the updated resource in the message body. In the latter case the response shall have 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 shall be created if resource creation by MnS consumers is supported for that resource (see clause 5.1.2).

Note that the HTTP PUT method has replace semantics and not merge semantics. A complete resource update in this context does not mean that all properties (attributes) defined by a schema for the representation of the resource need to be contained in the request, but that the existing representation on the MnS producer is replaced completely by the received representation (assuming no default values are defined for any of the attributes of the resource and the MnS Producer does not populate any of the attributes not received in the request with a value).

For example, assume the schema for the representation of a resource defines the attributes "attrA", "attrB" and "attrC". No default value is defined for these attributes. The current representation of the resource on the MnS Producer contains only "attrA" and "attrB".

- To update "attrA" and "attrB", the received resource representation needs to contain "attrA" with the new value and "attrB" with the new value.

- To update only "attrA", the received resource representation needs to contain "attrA" with the new value and "attrB" with the old value. Sending only a representation with "attrA" deletes "attrB" on the MnS Producer. Vice versa, to update only "attrB", the received resource representation needs to contain "attrA" with the old value and "attrB" with the new value. Sending only a representation with "attrB" deletes "attrA" on the MnS Producer.

- In case the received representation contains only "attrC" with some value, the new representation after the update contains only "attrC". The existing attributes "attrA" and "attrB" are deleted.

As for resource creation with HTTP PUT, this behavior is modified if default values are defined for attributes or if the MnS Producer populates attributes not contained in the HTTP PUT request with values. In both cases these attributes shall be returned in the response with the default value or assigned value.

Also, as for resource creation with HTTP PUT, a MnS Producer may modify attribute values included in the request and return the modified values to the MnS Consumer, or remove attributes received in the request and include only a subset of the received attributes in the response.

When the target resource has child resources that are included in the schema definition of the target resource, the representation of these child resources shall neither be included in the resource representation sent to the MnS Producer nor in the resource representation returned to the MnS Consumer. The overwrite semantic of PUT refers only to the target resource and not to child resources.

5.4 Design pattern for deleting a resource

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

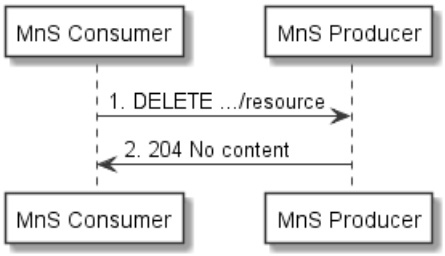

Figure 5.4-1: Flow for deleting a resource

The procedure is as follows:

1) The MnS Consumer sends an HTTP DELETE request to the MnS Producer. The resource to be deleted is identified with the URI. The target URI shall have no query and no fragment component. 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 shall be empty. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

When resources are structured with parent-child relations in a hierarchical tree, it shall not be possible to delete other resources than leaf resources. Attempts to delete other resources shall result in an error and the 409 (Conflict) status code shall be returned by the MnS Producer.

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 an HTTP POST request to the subscription resource.

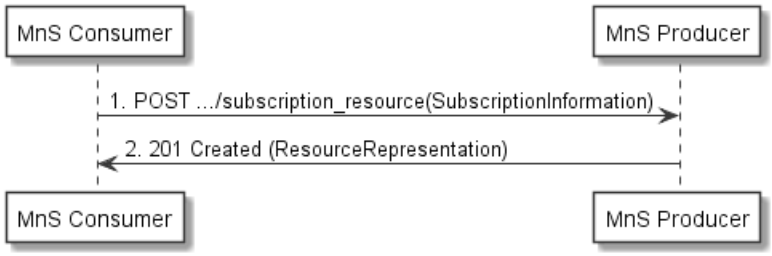

Figure 5.5.2-1: Flow for creating a subscription

The procedure is as follows:

1) The MnS Consumer (notification subscriber) sends an HTTP POST request to the MnS Producer. The URI shall indicate a subscriptions collection resource. The resources representing existing subscriptions are created below the collection resource. The subscriber shall indicate in the message body the URI to which notifications will be sent (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 shall return "201 Created" on success. The message body shall carry 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.

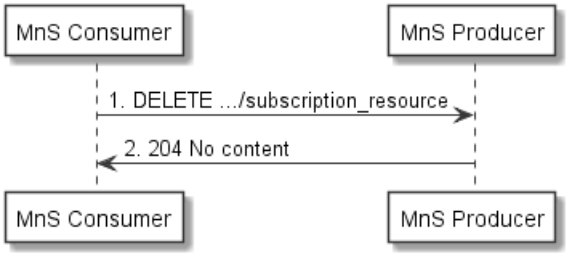

Figure 5.5.3-1: Flow for deleting a subscription

The procedure is as follows:

1) The MnS Consumer (notification subscriber) sends an HTTP DELETE request 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 shall be 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 an HTTP POST request to the notification sink.

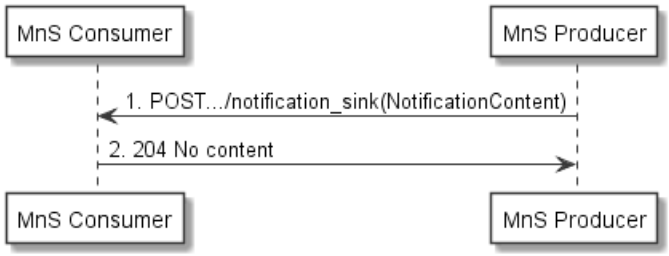

Figure 5.5.4-1: Flow for sending a notification

The procedure is as follows:

1) The MnS Producer sends an HTTP POST request to the MnS Consumer. The URI identifies the notification sink. The notification content shall be 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 receiving HTTP POST responses, 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 invoking the HTTP GETmethod on the URI returned by the server upon creation of this subscription. Information about all subscriptions can be read by invoking the HTTP GET method on the subscriptions collection resource.

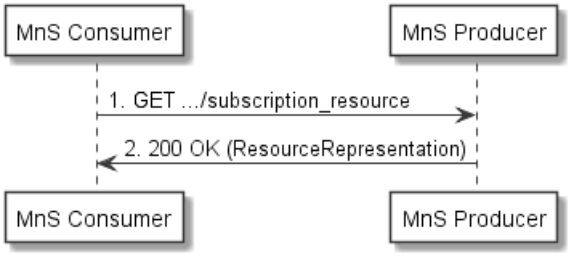

Figure 5.5.5-1: Flow for subscription retrieval

The procedure is as follows:

1) The MnS Consumer sends an HTTP GET request to the MnS Producer. The URI specifies the subscription resource or subscriptions collection 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 or subscriptions collection resource shall be 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. 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 infinite, 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. This is also the default case, when no "scopeType" query parameter is present in the request.
BASE_ALL	Selects the base resource and all of its descendant 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 descendant 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. It is not supported by any other method.

The URI query component shall be used for carrying the filter construct. The name of the query parameter is "filter".

XPath 1.0 [15] shall be used for specifying the filter construct.

The XPath expression is applied to an XML document constructed based on the following rules:

- The root element is the object identified by the path component of the target URI.
- The document includes scoped objects only.
- The document is constructed with the scoped objects using the hierarchical response construction method defined in clause.6.1.4.
- The JSON document constructed according to the first three bullet points is mapped into a conceptual XML document.

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 final response message according to clause 6.1.4.

The XPath expression shall be an absolute path expression, and hence always start with a backslash "/", that identifies the root node of the input document.

6.1.4 Construction rules for the response message body

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

Flat response construction method: The resources are returned as a flat list of JSON objects. Their location in the hierarchical containment tree shall be specified by, e.g. , their URI or Distinguished Name (DN) which needs to be returned for each resource. The object class name of each resource should be returned as well.

Hierarchical response construction method: The resources are returned inside the containment tree as specified by the JSON schema definition of the information model. For the resources that are not selected, the following applies:

- A resource is not returned at all if it is not an ancestor of any of the selected resources.
- A resource is returned empty, except for the resource identifiers, if it is a descendant of the base resource and an ancestor of any of the selected resources

The containment tree present in the response message shall always start with the base resource.

6.2 Design patterns for attribute and attribute field selection

6.2.1 Introduction

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

Often attributes have no scalar values but are complex structured data types with an own hierarchy and many attribute fields. In this case it may be desirable to identify not only the complete attribute but also individual 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, unselected attributes and attribute fields are removed from the resource representation.

6.2.2 Query parameters for attribute and attribute field selection

In case one or more attributes (with all attribute fields) are to be 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.The naming attribute "id" shall always be returned.

In case one or more fields of one or more attributes are to be retrieved, the name of the query parameter shall be "fields". The value of "fields" shall be a comma (",") separated list of entries that follow the syntax of JSON Pointer in JSON String Representation [14]. The context resource for the construction of the JSON Pointer is the resource identified by the target URI.

6.3 Design pattern for partially updating a resource

6.3.1 Introduction

HTTP PUT allows to replace (overwrite) a complete resource on the MnS Producer with the new representation in the request body. It cannot be used for partial updates of a resource.

For partial updates of a single resource HTTP PATCH (RFC 5789 [11]) shall be used. With PATCH, a set of changes to be applied to the target resource is described in the request message body. The set of changes carried in the message body is called patch document. The format of the patch document is identified by its media type. RFC 5789 [11] does not define any patch format, only the PATCH method.

The HTTP PATCH method is atomic, as per RFC5789 [11]. The MnS Producer shall apply the entire set of changes atomically and never provide (e.g., in response to a GET during this operation) a partially modified representation. If the entire patch document cannot be successfully applied, then the MnS Producer shall not apply any of the changes. PATCH thus has transaction semantics.

For JSON, IETF has defined two patch formats for the use with the HTTP Patch method: JSON Merge Patch (RFC 7396 [12]) and JSON Patch (RFC 6902 [13]). The usage of these patch formats is described in the following clauses.

6.3.2 JSON Merge Patch

RFC 7396 [12] specifies a simple patch format in JSON called JSON Merge Patch. It allows to describe a set of modifications to be applied to the target resource representation. JSON Merge Patch works at the level of name/value pairs. The received patch document is merged into the target resource representation. The media type of the patch document 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.

The target resource is identified by the target URI. The target URI shall have no query and no fragment component. The target resource must exist, otherwise the error status code 404 (Not Found) shall be returned.

The "id" of the resource shall be present in the patch document and shall be identical to the "id" of the patched resource in the request URI. This ensures uniformity of resource representations in message bodies, though, strictly speaking, the presence of the "id" in the patch document is redundant.

JSON Merge Patch does not allow manipulation of arrays other than replacing the complete array value (an array with all present items) with a new value (an array with all new items). It is not possible to change individual items in an array or to add/delete individual items.

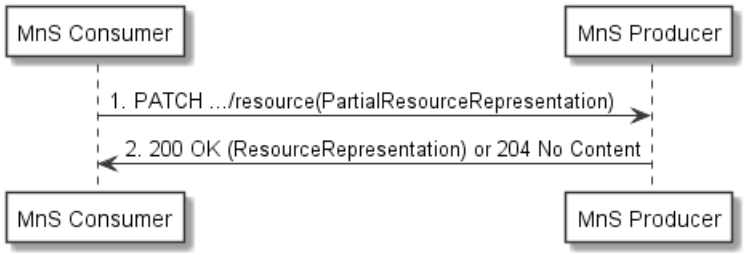

Figure 6.3.2-1: Flow for partially updating a resource with JSON Merge Patch

The procedure flow is as follows:

1) The MnS Consumer sends an HTTP PATCH request to the MnS Producer. The resource to be updated is identified with the target URI. The message body shall carry the JSON Merge Patch document describing a set of modifications 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.

JSON Merge Patch shall be used for patching the target resource only. The patch format shall not be used for creating, modifying or deleting child resources of the target resource in the same request, even if the child resources are included in the schema definition of the target resource. This limitation is introduced, because child resources (of one object class) are represented as items of an array that is a property of the target resource (alongside with the attributes of the target resource), and JSON Merge Patch does not allow to modify individual array items. With JSON Merge Patch, only the complete array value with the representations of all child resources (of one class) could be replaced. Note that child resources can have child resources as well. The patch document would hence need to include the representations of all descendant resources. This is very inefficient and against the principle of PATCH to provide the changes only.

Assume an "XyzFunction" resource has no attribute "attrA" yet, then the following PATCH request creates it.

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

{
  "id": "XYZF1",
  "attributes": {
    "attrA": "abc"
  }
}
```

The following subsequently executed PATCH request replaces its value with "def".

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

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

6.3.3 JSON Patch

The JSON Patch format is specified in RFC 6902 [13]. The patch document is a JSON array. Each array item is a JSON object describing a modification to be applied to the target resource. The modifications shall be applied to the target resource sequentially in the order they appear in the array. The media type of JSON Patch is "application/json-patch+json".

Each modification is defined by three properties: The operation ("op"), the identification of the secondary resource within the target resource to be manipulated ("path") and a value ("value") that is not present when removing a secondary resource. When moving or copying an existing value, the "value" property is absent and a "from" property is present instead. The value of the "from" and "path" property is a JSON Pointer in string representation as defined in Section 5 of RFC 6901 [14].

In contrast to JSON Merge Patch, JSON Patch allows to modify individual items of an array. Array items are identified based on their position (index) in an array. The first item has the index "0". The "-" character is used by the operations "add" and "move" to index the end of the array for appending a new array item. Its use in any other operation is forbidden.

The target URI identifies the resource to be modified. As for JSON Merge Patch, the target URI shall have no query and no fragment component. The target resource must exist, otherwise the error status code 404 (Not Found) shall be returned.

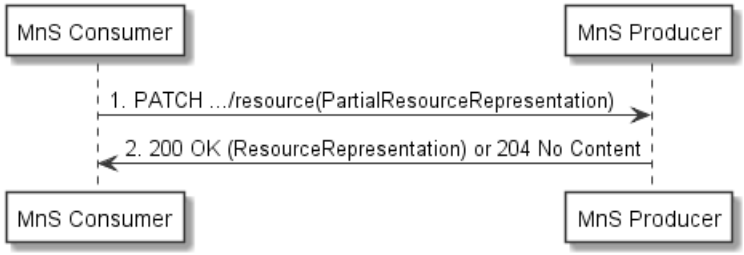

Figure 6.3.3-1: Flow for partially updating a resource with JSON Patch

The procedure flow is as follows:

1) The MnS Consumer sends an HTTP PATCH request to the MnS Producer. The resource to be updated is identified with the target URI. The message body shall carry a JSON 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.

As JSON Merge Patch, also JSON Patch shall be used for patching the target resource only. The patch format shall not be used for creating, modifying or deleting child resources of the target resource in the same request, even if the child resources are included in the schema definition of the target resource. This is because JSON Patch can address items in an array only based on the position of the item in the array, and not based on an identifier independent from the position of the item in the array. A patch document could hence not address descendant resources of the target resource based on their "id". This is prone to conflicts in multi-client scenarios, where the position of resource items in an array can change due to the concurrent creation or deletion of resource items in the same array. Risk mitigation would require complex ETag calculations in the resource hierarchy.

The following example adds a new attribute "attrA" to an "XyzFunction".

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

[
  {
    "op": "add",
    "path": "/attributes",
    "value": {
      "attrA": "abc"
    }
  }
]
```

The following example replaces its value with "def".

```
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": {
      "attrA": "def"
    }
  }
]
```

6.4 Design patterns 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 specifies methods allowing to manipulate (create, update, delete) multiple resources with a single HTTP PATCH request.

The HTTP PATCH method is atomic as defined in clause 6.3.1.

JSON Merge Patch and JSON Patch are used for partial updates of a single resource. Extensions to these patch formats are specified below to allow efficient manipulation of multiple resources (target resource and descendant resources) with one patch document. The extended patch formats are called 3GPP JSON Merge Patch and 3GPP JSON Patch.

6.4.2 3GPP JSON Merge Patch

3GPP JSON Merge Patch is a 3GPP defined extension to JSON Merge Patch (RFC 7396 [12]). It allows, using a single patch document, to update the target resource (as does JSON Merge Patch) and to update, create or delete descendant resources, which JSON Merge Patch does not allow, at least not in an efficient manner. This is achieved by relaxing for arrays that contain resources (of a single object class) as array items the constraint that the complete updated array value needs to be provided in the merge document. Instead, only resources to be manipulated are present in the patch document. These resources are identified with their "id". Resources that are not manipulated are either absent or present with their "id" only, when this is required to navigate along the containment tree to the resource to be patched. The same considerations as for the hierarchical response construction method (clause 6.1.4) apply.

The merge semantic of JSON Merge Patch is hence extended to descendant resources of the target resource. Note that the behaviour of patching attributes of type array does not change in 3GPP JSON Merge Patch compared to JSON Merge Patch. The complete updated array value needs to be provided for attributes of type array also in a 3GPP JSON Merge Patch document. It is not possible to patch individual array items only.

As for JSON Merge Patch, the target URI shall have no query and no fragment component. The target resource must exist, otherwise the error status code 404 (Not Found) shall be returned. The target URI shall identify a resource that is a common ancestor of the resources to be patched. The patch document itself shall start with the resource identified by the target URI.

A resource is deleted by setting the "attributes" property of the resource to "null". In case a complete subtree is deleted, all resources from the base resource of the subtree down to the leaf resources shall be marked for deletion. When creating new resources, the object class name of the resource to be created shall be contained in the patch document for the resources to be created.

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.

The procedure is as follows:

1) The MnS Consumer sends an HTTP PATCH request to the MnS Producer. The message body shall carry 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 6.1.4, 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.

6.4.3 3GPP JSON Patch

3GPP JSON Patch is a 3GPP defined extension to JSON Merge Patch (RFC 6902 [13]).

Like 3GPP JSON Merge Patch, it allows, using a single patch document, to update the target resource (as does JSON Patch) and to update, create or delete descendant resources, which JSON Patch does not allow, at least not based on resource identifiers.

This extension is that the "path" and "from" properties of a patch operation define an offset to the target resource as specified by the request URI. This offset is relative to the target URI. It has a first component pointing to a resource below the target resource, and a second component pointing to a secondary resource within the resource identified by the first component.

The first component of "path" or "from" is built from URI path components. It follows the same syntax as the path components of the target URI. The second component is a URI fragment with a JSON pointer in the URI fragment identifier representation as defined in clause 6 of RFC 6901 [14], i.e. the second component starts with the "#" character. Both components are concatenated without a delimiter.

For example, assume the target URI is "/SubNetwork=SN1" and the "userLabel" attribute of a child of class "ManagedElement" with the id "ME1" is to be patched, then the first path component is "/ManagedElement=ME1/" and the second path component is "#attributes/userLabel". This results in the following path:

"path": "/ManagedElement=ME1/#attributes/userLabel".

The target URI shall identify a common ancestor resource of the resources to be patched.

When creating new resources ("op"="add"), the object class name of the resource to be created shall be included in the "value" property of the operation.

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.

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 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 either the flat or 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.

A single operation in a 3GPP JSON Patch document shall patch a single (primary) resource only. Different operations in a patch document can patch different resources though. The consequence of this restriction is for example that subtrees with multiple resources cannot be created or deleted with a single patch operation. Each resource needs to be created or deleted with an own patch operation in the patch document. This behaviour is aligned with those of the PUT and DELETE methods.

Note that the "replace" operation of (3GPP) JSON Patch has replace semantics like PUT and not merge semantics like JSON Merge Patch. When multiple attributes or attribute fields of a resource are patched, then a patch operation for each update is required, for example

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

[
  {
    "op": "replace",
    "path": "#/attributes/userLabel",
    "value": "Berlin NW-1"
  },
  {
    "op": "replace",
```

```
    "path": "#/attributes/plmnId/mcc",
    "value": 654
  }
]
```

To streamline partial updates of single resources, 3GPP JSON Patch introduces a new patch operation named "merge". For that operation, the JSON object contained in the "value" property shall be merged into the target resource referenced by "path" using the rules of JSON Merge Patch (RFC 7396 [12]). An MnS Producer shall verify if a "merge" operation is for a single resource by checking if the "path" property contains the string "#/attributes" and shall reject the request with "422 Unprocessable Entity" if it doesn't.

With the "merge" operation, the updates in the previous example can be expressed as follows.

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

[
  {
    "op": "merge",
    "path": "#/attributes",
    "value": {
      "userLabel": "Berlin NW-1",
      "plmnId": {
        "mcc": 654
      }
    }
  }
]
```

The following example is invalid. It attempts to patch the contained "ManagedElement" resources, which is not allowed.

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

[
  {
    "op": "merge",
    "path": "",
    "value": {
      "attributes": {
        "userLabel": "Berlin NW-1",
        "plmnId": {
          "mcc": 654
        }
      },
      "ManagedElement": [
        {
          ...
        }
      ]
    }
  }
]
```

6.5 Large queries

Clauses 6.1 and 6.2 have introduced a pattern that allows querying resources by passing query parameters in the query part of the target URI of a GET request. However, there can be scenarios where the query string can get very long, exceeding the URI length that can be expected to be supported by all implementations.

IETF RFC 7130 [5] recommends that a request URI length of at least 8000 octets should be supported. Further, IETF RFC 7130 [5] requires that implementations shall respond with 414 (URI Too Long) in case the actual request URI is longer than the supported request URI length.

When the URI length exceeds the supported limit, the query may be passed in the payload body of a POST request instead of the target URI of a GET request. To signal that the semantics of this POST request is actually the same as a GET request, the "X-HTTP-Method-Override: GET" HTTP header shall be included in the request.

If the data format of the query in the POST request payload body is a list of name-value pairs separated by the "&" character (as defined in clauses 6.1 and 6.2 of the present document), the "Content-Type" header of the POST request shall be set to "application/x-www-form-urlencoded". Using other data formats for long queries and signalling them appropriately in the "Content-Type" request header is possible but needs to be documented in the specific MnS documentation.

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 RESTful HTTP-based solution set

W.1 Mapping of operations

W.1.1 Introduction

The IS operations are mapped to SS equivalents according to table W.1.1-1.

Table W.1.1-1: Mapping of IS operations to SS equivalents

IS operation	HTTP Method	Resource URI	S

W.1.2 Operation <operation 1>

The IS operation parameters are mapped to SS equivalents according to table W.1.2-1 and table W.1.2-2.

Table W.1.2-1: Mapping of IS operation input parameters to SS equivalents (<HTTP method>)

IS parameter name	SS parameter location	SS parameter name	SS parameter type	S

Table W.1.2-2: Mapping of IS operation output parameters to SS equivalents (<HTTP method>)

IS parameter name	SS parameter location	SS parameter name	SS parameter type	S

W.1.3 Operation <operation 2>

Same as for <operation 1>.

W.2 Mapping of notifications

W.2.1 Introduction

The IS notifications are mapped to SS equivalents according to table W.2.1-1.

Table W.2.1-1: Mapping of IS operations to SS equivalents

IS notification	HTTP Method	Resource URI	S

W.2.2 Notification <notification 1>

The IS notification parameters are mapped to SS equivalents according to table W.2.2-1.

Table W.2.2-1: Mapping of IS notification parameters to SS equivalents

IS parameter name	SS parameter location	SS parameter name	SS parameter type	S

W.2.3 Notification <notification 2>

Same as for <notification 1>.

W.3 Usage of HTTP

W.4 Resources

W.4.1 Resource structure

W.4.1.1 Resource structure on the MnS producer

Figure W.4.1.1-1 shows the resource structure of the <XYZ> MnS on the MnS producer.

<Figure>

Figure W.4.1.1-1: Resource URI structure of the <XYZ> MnS on the MnS producer

Table W.4.1.1-1 provides an overview of the resources and applicable HTTP methods.

Table W.4.1.1-1: Resources and methods overview

Resource name	Resource URI	HTTP method	Description

W.4.1.2 Resource structure on the MnS consumer

Figure W.4.1.2-1 shows the resource structure of the <XYZ> MnS on the MnS consumer.

<Figure>

Figure W.4.1.2-1: Resource URI structure of the <XYZ> MnS on the MnS consumer

Table W.4.1.2-1 provides an overview of the resources and applicable HTTP methods.

Table W.4.1.2-1: Resources and methods overview

Resource name	Resource URI	HTTP method	Description

W.4.2 Resource definitions

W.4.2.1 Resource <resource 1>

W.4.2.1.1 Description

Description of the resource.

W.4.2.1.2 URI

Resource URI: <URI>

The resource URI variables are defined in table W.4.2.1.2-1.

Table W.4.2.1.2-1: URI variables

Name	Definition

W.4.2.1.3 HTTP methods

W.4.2.1.3.1<method 1>

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

Table W.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 W.2.1.3.1-2 and the response data structures and response codes specified in table W.2.1.3.1-3.

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

Data type	P	Cardinality	Description

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

Data type	P	Cardinality	Response codes	Description

W.4.2.1.3.2<method 2>

Same as for <method 1>.

W.4.2.2 Resource <resource 2>

Same as for <resource 1>.

W.5 Data type definitions

W.5.1 General

This clause defines the data types used by the <XYZ> MnS. Table W.4.1-1 specifies the data types defined in the present document and table W.4.1-2 the data types imported

Table W.4.1-1: Data types defined in the present document

Data type	Reference	Description

Table W.4.1-2: Data types imported

Data type	Reference	Description

W.5.2 Structured data types

W.5.2.1 Type <TypeName 1>

Table W.4.2.1-1: Definition of type <TypeName 1>

Attribute name	Data type	P	Cardinality	Description

W.5.2.2 Type <TypeName 2>

Same as for <TypeName 1>.

W.5.3 Simple data types and enumerations

W.5.3.1 General

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

W.5.3.2 Simple data types

Table W.5.3.2-1: Simple data types

Type Name	Type Definition	Description

W.5.3.3 Enumeration <EnumType1>

Table W.5.3.3-1: Enumeration < EnumType1>

Enumeration value	Description

W.5.3.4 Enumeration <EnumType2>

Annex A (normative)

OpenAPI definition

A.1 Introduction

This clause contains the OpenAPI definition of the <XYZ> MnS in YAML format.

A.2 OpenAPI document "<ABC>.yaml"

OpenAPI definition

Annex A (informative): Examples

A.1 Example data model

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

```
{
  "SubNetwork": [
    {
      "id": "SN1",
      "objectClass": "SubNetwork",
      "objectInstance": "SubNetwork=SN1",
      "attributes": {
        "userLabel": "Berlin NW",
        "userDefinedNetworkType": "5G",
        "plmnId-id": {
          "mcc": 456,
          "mnc": 789
        }
      },
      "ManagedElement": [
        {
          "id": "ME1",
          "objectClass": "ManagedElement",
          "objectInstance": "SubNetwork=SN1,ManagedElement=ME1",
          "attributes": {
            "userLabel": "Berlin NW 1",
            "vendorName": "Company XY",
            "location": "TV Tower"
          },
          "XyzFunction": [
            {
              "id": "XYZF1",
              "objectClass": "XyzFunction",
              "objectInstance":"SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF1",
              "attributes": {
                "attrA": "xyz",
                "attrB": 551
              }
            },
            {
              "id": "XYZF2",
              "objectClass": "XyzFunction",
              "objectInstance":"SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF2",
              "attributes": {
                "attrA": "abc",
                "attrB": 552
              }
            }
          ]
        },
        {
          "id": "ME2",
          "objectClass": "ManagedElement",
          "objectInstance": "SubNetwork=SN1,ManagedElement=ME2",
          "attributes": {
            "userLabel": "Berlin NW 2",
            "vendorName": "Company XY",
            "location": "Grunewald"
          }
        }
      ],
      "PerfMetricJob": [
        {
          "id": "PMJ1",
          "objectClass": "PerfMetricJob",
          "objectInstance": "SubNetwork=SN1,PerfMetricJob=PMJ1",
          "attributes": {
            "granularityPeriod": "5",
            "perfMetrics": [
              "Metric1",
```

```
              "Metric2"
            ],
            "objectInstances": [
              "Obj1",
              "Obj2"
            ]
          }
        }
      ],
      "ThresholdMonitor": [
        {
          "id": "TM1",
          "objectClass": "ThresholdMonitor",
          "objectInstance": "SubNetwork=SN1,ThresholdMonitor=TM1",
          "attributes": {
            "metric": "Metric1",
            "thresholdLevels": [
              {
                "level": "1",
                "thresholdValue": 10
              },
              {
                "level": "2",
                "thresholdValue": 20
              },
              {
                "level": "3",
                "thresholdValue": 30
              }
            ]
          }
        }
      ]
    }
  ]
}
```

The corresponding JSON schema is

```
{
  "SubNetwork": {
    "type": "array",
    "items": {
      "type": "object",
      "properties": {
        "id": {
          "type": "string"
        },
        "objectClass": {
          "type": "string"
        },
        "objectInstance": {
          "type": "string"
        },
        "attributes": {
          "type": "object",
          "properties": {
            "userLabel": {
              "type": "string"
            },
            "userDefinedNetworkType": {
              "type": "string"
            },
            "plmnId": {
              "type": "object",
              "properties": {
                "mcc": {
                  "type": "integer"
                },
                "mnc": {
                  "type": "integer"
                }
              }
            }
          }
        },
        "ManagedElement": {
```

```
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "objectClass": {
                "type": "string"
              },
              "objectInstance": {
                "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"
                    },
                    "objectClass": {
                      "type": "string"
                    },
                    "objectInstance": {
                      "type": "string"
                    },
                    "attributes": {
                      "type": "object",
                      "properties": {
                        "attributeA": {
                          "type": "string"
                        },
                        "attributeB": {
                          "type": "integer"
                        }
                      }
                    },
                    "required": ["id"]
                  }
                }
              },
              "required": ["id"]
            }
          }
        },
        "PerfMetricJob": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "objectClass": {
                "type": "string"
              },
              "objectInstance": {
                "type": "string"
              },
              "attributes": {
                "type": "object",
                "properties": {
                  "granularityPeriod": {
```

```
                    "type": "integerstring"
                  },
                  "perfMetrics": {
                    "type": "array",
                    "items": {
                      "type": "string"
                    }
                  }
                },
                "objectInstances": {
                  "type": "array",
                  "items": {
                    "type": "string"
                  }
                }
              },
              "required": ["id"]
            }
          }
        },
        "ThresholdMonitor": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "objectClass": {
                "type": "string"
              },
              "objectInstance": {
                "type": "string"
              },
              "attributes": {
                "type": "object",
                "properties": {
                  "thresholdLevels": {
                    "type": "array",
                    "items": {
                      "type": "object",
                      "properties": {
                        "level": {
                          "type": "string"
                        },
                        "thresholdValue": {
                          "type": "integer"
                        }
                      }
                    }
                  }
                }
              },
              "required": ["id"]
            }
          }
        },
        "required": ["id"]
      }
    }
  }
}
```

The corresponding XML instance document is provided below as well. It can be helpful when evaluating XPath expressions.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SubNetwork>
    <id>SN1</id>
    <objectClass>SubNetwork</objectClass>
    <objectInstance>SubNetwork=SN1</objectInstance>
    <attributes>
        <userLabel>Berlin NW</userLabel>
        <userDefinedNetworkType>5G</userDefinedNetworkType>
        <plmnId>
            <mcc>456</mcc>
```

```
            <mnc>789</mnc>
        </plmnId>
    </attributes>
    <ManagedElement>
        <id>ME1</id>
        <objectClass>ManagedElement</objectClass>
        <objectInstance>SubNetwork=SN1,ManagedElement=ME1</objectInstance>
        <attributes>
            <userLabel>Berlin NW 1</userLabel>
            <vendorName>Company XY</vendorName>
            <location>TV Tower</location>
        </attributes>
        <XyzFunction>
            <id>XYZF1</id>
            <objectClass>XyzFunction</objectClass>
            <objectInstance>SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF1</objectInstance>
            <attributes>
                <attrA>xyz</attrA>
                <attrB>551</attrB>
            </attributes>
        </XyzFunction>
        <XyzFunction>
            <id>XYZF2</id>
            <objectClass>XyzFunction</objectClass>
            <objectInstance>SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF2</objectInstance>
            <attributes>
                <attrA>abc</attrA>
                <attrB>552</attrB>
            </attributes>
        </XyzFunction>
    </ManagedElement>
    <ManagedElement>
        <id>ME2</id>
        <objectClass>ManagedElement</objectClass>
        <objectInstance>SubNetwork=SN1,ManagedElement=ME2</objectInstance>
        <attributes>
            <userLabel>Berlin NW 2</userLabel>
            <vendorName>Company XY</vendorName>
            <location>Grunewald</location>
        </attributes>
    </ManagedElement>
    <PerfMetricJob>
        <id>PMJ1</id>
        <objectClass>PerfMetricJob</objectClass>
        <objectInstance>SubNetwork=SN1,PerfMetricJob=PMJ1</objectInstance>
        <attributes>
            <granularityPeriod>5</granularityPeriod>
            <perfMetrics>Metric1</perfMetrics>
            <perfMetrics>Metric2</perfMetrics>
            <objectInstances>Obj1</objectInstances>
            <objectInstances>Obj2</objectInstances>
        </attributes>
    </PerfMetricJob>
    <ThresholdMonitor>
        <id>TM1</id>
        <objectClass>ThresholdMonitor</objectClass>
        <objectInstance>SubNetwork=SN1,ThresholdMonitor=TM1</objectInstance>
        <attributes>
            <ThresholdLevels>
              <level>1</level>
              <thresholdValue>10</thresholdValue>
            </ThresholdLevels>
            <ThresholdLevels>
              <level>2</level>
              <thresholdValue>20<thresholdValue>
            </ThresholdLevels>
            <ThresholdLevels>
              <level>3</level>
              < thresholdValue>30</thresholdValue>
            </ThresholdLevels>
        </attributes>
    </ThresholdMonitor>
</SubNetwork>
```

NOTE: Void

The following examples do not always follow the URI structure specified in clause 4.4. For simplicity reasons, the path component "/{MnSName}/{MnSVersion}" is often omitted. Also the Domain Component (DC) is omitted in DNs carried by "objectInstance" attributes. Though this is a valid implementation as per TS 32.300 [3], it is recommended to have "DC=example.org" or "DC=org, DC=example" as first components of DNs.

A.2 Retrieval of resources

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

To retrieve a complete "XyzFunction" 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
  }
}
```

Alternatively, the response 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. Alternatively, "XyzFunction" might also be an object, since the JSON schema specifying the response message body is not required to be identical to the JSON schema specifying the resources contained by a 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
    }
  }
}
```

Alternatively, 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
      }
    }
  }
}
```

When building the response based on the flat response construction method, the response is given by

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

[
  {
    "id": "XYZF1",
    "objectClass": "XyzFunction",
    "objectInstance": "SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF1",
    "attributes": {
      "attrA": "xyz",
      "attrB": 551
    }
  }
]
```

The exact syntax of the response body is specified by the JSON schema included in the concrete MnS definition. In current MnS definitions the format of the first example response above is used. This style is also followed in subsequent examples.

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

To retrieve only the "userLabel" attribute and the "mnc" attribute field of the "plmnId" attribute of the "SubNetwork", the MnS Consumer might send:

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

Alternatively one might send as well

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

The response contains only the selected attribute "userLabel" and the selected attribute field "mnc":

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

{
  "id": "SN1",
  "attributes": {
    "userLabel": "Berlin NW",
    "plmnId": {
      "mnc": 789
    }
  }
}
```

In this example, the MnS Consumer retrieves the "userLabel" and "vendorName" of the "ManagedElement" whose "id" is equal to "ME1":

```
GET /SubNetwork=SN1/ManagedElement=ME1?attributes=userLabel,vendorName HTTP/1.1
Host: example.org
Accept: application/json
```

The MnS Producer responds as follows:

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

{
  "id": "ME1",
  "attributes": {
    "userLabel": "Berlin NW 1",
    "vendorName": "Company XY"
  }
}
```

The following request selects all attributes:

```
GET /SubNetwork=SN1/ManagedElement=ME1?fields=/attributes HTTP/1.1
Host: example.org
Accept: application/json
```

It is thus identical to:

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

Both requests return the complete resource representation with all attributes:

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

{
  "id": "ME1",
  "attributes": {
    "userLabel": "Berlin NW 1",
    "vendorName": "Company XY",
    "location": "TV Tower"
  }
}
```

The following request returns the first item of the "perfMetrics" attribute, which is of type array:

```
GET /SubNetwork=SN1/ManagedElement=ME1/PerfMetricJob=PMJ1?fields=attributes/perfMetrics/0 HTTP/1.1
Host: example.org
Accept: application/json
```

Note indices start with "0" in JSON Pointer. The response looks like:

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

{
  "id": "PMJ1",
  "attributes": {
    "perfMetrics": [
      "Metric1"
    ]
  }
}
```

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

The following example selects the "SubNetwork" as base object at scope level "0" and all objects at scope level "1":

```
GET /SubNetwork=SN1?scopeType=BASE_SUBTREE&scopeLevel=1 HTTP/1.1
Host: example.org
Accept: application/json
```

All objects at scope level "1" are included in the response irrespective of their object class. When using the hierarchical response construction method, the response looks as follows:

```
{
  "id": "SN1",
  "attributes": {
    "userLabel": "Berlin NW",
    "userDefinedNetworkType": "5G",
    "plmnId": {
      "mcc": 456,
      "mnc": 789
    }
  },
  "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"
      }
    }
  ],
  "PerfMetricJob": [
    {
      "id": "PMJ1",
      "attributes": {
        "granularityPeriod": 5,
        "perfMetrics": [
          "Metric1",
          "Metric2"
        ],
        "objectInstances": [
          "Obj1",
          "Obj2"
        ]
      }
    }
  ],
  "ThresholdMonitor": [
    {
      "id": "TM1",
      "attributes": {
        "metric": "Metric1",
        "thresholdLevels": [
          {
            "level": "1",
            "thresholdValue": 10
          },
          {
            "level": "2",
            "thresholdValue": 20
          },
          {
            "level": "3",
            "thresholdValue": 30
          }
        ]
```

```
      }
    }
  ]
}
```

The response constructed with the flat response construction method looks like:

```
[
  {
    "id": "SN1",
    "objectClass": "SubNetwork",
    "objectInstance": "SubNetwork=SN1",
    "attributes": {
      "userLabel": "Berlin NW",
      "userDefinedNetworkType": "5G",
      "plmnId": {
        "mcc": 456,
        "mnc": 789
      }
    }
  },
  {
    "id": "ME1",
    "objectClass": "ManagedElement",
    "objectInstance": "SubNetwork=SN1,ManagedElement=ME1",
    "attributes": {
      "userLabel": "Berlin NW 1",
      "vendorName": "Company XY",
      "location": "TV Tower"
    }
  },
  {
    "id": "ME2",
    "objectClass": "ManagedElement",
    "objectInstance": "SubNetwork=SN1,ManagedElement=ME2",
    "attributes": {
      "userLabel": "Berlin NW 2",
      "vendorName": "Company XY",
      "location": "Grunewald"
    }
  },
  {
    "id": "PMJ1",
    "ojectClass": "PerfMetricJob",
    "objectInstance": "SubNetwork=SN1,PerfMetricJob=PMJ1",
    "attributes": {
      "granularityPeriod": "5",
      "perfMetrics": [
        "Metric1",
        "Metric2"
      ],
      "objectInstances": [
        "Obj1",
        "Obj2"
      ]
    }
  },
  {
    "id": "TM1",
    "ojectClass": "ThresholdMonitor",
    "objectInstance": "SubNetwork=SN1,ThresholdMonitor=TM1",
    "attributes": {
      "metric": "Metric1",
      "thresholdLevels": [
        {
          "level": "1",
          "thresholdValue": 10
        },
        {
          "level": "2",
          "thresholdValue": 20
        },
        {
          "level": "3",
          "thresholdValue": 30
        }
      ]
    }
```

```
  }
]
```

When only objects at scope level "1" are requested to be returned, the request looks like:

```
GET /SubNetwork=SN1?scopeType=BASE_NTH_LEVEL&scopeLevel=1 HTTP/1.1
Host: example.org
Accept: application/json
```

The response does not include the attributes of "SubNetwork" any more, only its "id" is included:

```
{
  "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"
      }
    }
  ],
  "PerfMetricJob": [
    {
      "id": "PMJ1",
      "attributes": {
        "granularityPeriod": 5,
        "perfMetrics": [
          "Metric1",
          "Metric2"
        ],
        "objectInstances": [
          "Obj1",
          "Obj2"
        ]
      }
    }
  ],
  "ThresholdMonitor": [
     {
      "id": "TM1",
      "attributes": {
        "metric": "Metric1",
        "thresholdLevels": [
          {
            "level": "1",
            "thresholdValue": 10
          },
          {
            "level": "2",
            "thresholdValue": 20
          },
          {
            "level": "3",
            "thresholdValue": 30
          }
        ]
      }
    }
  ]
}
```

Similarly, for reading all objects on scope level "2", the MnS Consumer may send:

```
GET /SubNetwork=SN1?scopeType=BASE_NTH_LEVEL&scopeLevel=2 HTTP/1.1
Host: example.org
Accept: application/json
```

When using the hierarchical response construction method, the response includes the complete representations of the "XyzFunction" objects. The "SubNetwork" and "ManagedElement" are present with their "id" only; they provide the containment nodes for the "XyzFunction" objects.

```
{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME1",
      "XyzFunction": [
        {
          "id": "XYZF1",
          "attributes": {
            "attrA": "xyz",
            "attrB": 551
          }
        },
        {
          "id": "XYZF2",
          "attributes": {
            "attrA": "abc",
            "attrB": 552
          }
        }
      ]
    }
  ]
}
```

The "PerfMetricJob" and "ThresholdMonitor" are not included altogether, not even with the "id" only. This is because these nodes do not represent necessary path components to the scoped objects on the second level.

When using the flat response construction method, the response includes only "XyzFunction" objects without containment nodes. The "objectInstance" of each returned object is present in this case, as required in clause 6.1.4.

```
[
  {
    "id": "XYZF1",
    "objectClass": "XyzFunction",
    "objectInstance": "SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF1",
    "attributes": {
      "attrA": "xyz",
      "attrB": 551
    }
  },
  {
    "id": "XYZF2",
    "objectClass": "XyzFunction",
    "objectInstance": "SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF2",
    "attributes": {
      "attrA": "abc",
      "attrB": 552
    }
  }
]
```

The following example selects all objects on scope level "1" that have a "location" attribute whose value is equal to "Grunewald":

```
GET /SubNetwork=SN1?\
      scopeType=BASE_NTH_LEVEL&scopeLevel=1\
      filter=/*/*[attributes[location="Grunewald"]] HTTP/1.1
Host: example.org
Accept: application/json
```

The response includes one "ManagedElement" object only:

```
{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME2",
      "attributes": {
        "userLabel": "Berlin NW 2",
```

```
        "vendorName": "Company XY",
        "location": "Grunewald"
      }
    }
  ]
}
```

The input document to the XPath filter is a document whose root node is the object identified by the path component of the URI and that includes the object representations of the scoped objects. In this example the root node is the "SubNetwork" without the "attributes" node. The input document includes all scoped objects on the scope level "1". These are the two "ManagedElement" objects and the "PerfMetricJob" object:

```
{
  "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"
        }
      }
    ],
    "PerfMetricJob":[
      {
        "id": "PMJ1",
        "attributes": {
          "granularityPeriod": 5,
          "perfMetrics": ["Metric1", "Metric2"],
          "objectInstances": ["Obj1", "Obj2"]
        }
      }
    ],
    "ThresholdMonitor": [
      {
        "id": "TM1",
        "attributes": {
          "metric": "Metric1",
          "thresholdLevels": [
            {
              "level": "1",
              "thresholdValue": 10
            },
            {
              "level": "2",
              "thresholdValue": 20
            },
            {
              "level": "3",
              "thresholdValue": 30
            }
          ]
        }
      }
    ]
  }
}
```

An implementation may be based on available XPath tools. In that case the JSON document may have to be converted to a XML document:

```
<SubNetwork>
    <id>SN1</id>
    <ManagedElement>
        <id>ME1</id>
        <attributes>
```

```
            <userLabel>Berlin NW 1</userLabel>
            <vendorName>Company XY</vendorName>
            <location>TV Tower</location>
        </attributes>
    </ManagedElement>
    <ManagedElement>
        <id>ME2</id>
        <attributes>
            <userLabel>Berlin NW 2</userLabel>
            <vendorName>Company XY</vendorName>
            <location>Grunewald</location>
        </attributes>
    </ManagedElement>
    <PerfMetricJob>
        <id>PMJ1</id>
        <attributes>
            <granularityPeriod>5</granularityPeriod>
            <perfMetrics>Metric1</perfMetrics>
            <perfMetrics>Metric2</perfMetrics>
            <objectInstances>Obj1</objectInstances>
            <objectInstances>Obj2</objectInstances>
        </attributes>
    </PerfMetricJob>
    <ThresholdMonitor>
        <id>TM1</id>
        <attributes>
            <ThresholdLevels>
              <level>1</level>
              <thresholdValue>10</thresholdValue>
            </ThresholdLevels>
            <ThresholdLevels>
              <level>2</level>
              <thresholdValue>20</thresholdValue>
            </ThresholdLevels>
            <ThresholdLevels>
              <level>3</level>
              <thresholdValue>30</thresholdValue>
            </ThresholdLevels>
        </attributes>
    </ThresholdMonitor>
</SubNetwork>
```

In this example the complete "ManagedElement" object is the result of applying the XPath expression:

```
<ManagedElement>
  <id>ME2</id>
  <attributes>
    <userLabel>Berlin NW 2</userLabel>
    <vendorName>Company XY</vendorName>
    <location>Grunewald</location>
  </attributes>
</ManagedElement>
```

XPath predicates allow to specify also ranges. The following example selects objects on scope level "2" that have an attribute with name "attrB" whose value is equal to or greater than 552 and less than 562.

```
GET /SubNetwork=SN1?\
      scopeType=BASE_NTH_LEVEL&scopeLevel=2\
      filter=/*/*/*[attributes[attrB>=552 and attrB<562]] HTTP/1.1
Host: example.org
Accept: application/json
```

The response includes one "XyzFunction" object only:

```
{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME1",
      "XyzFunction": [
        {
          "id": "XYZF2",
          "attributes": {
            "attrA": "abc",
            "attrB": 552
          }
        }
```

```
      ]
    }
  ]
}
```

An identical response is returned when using the following requests:

```
GET /SubNetwork=SN1?\
      scopeType=BASE_ALL\
      filter=//*[attributes[attrB>=552 and attrB<562]] HTTP/1.1
Host: example.org
Accept: application/json
```

or

```
GET /SubNetwork=SN1?\
      scopeType=BASE_SUBTREE&scopeLevel=2\
      filter=//*[attributes[attrB>=552 and attrB<562]] HTTP/1.1
Host: example.org
Accept: application/json
```

or

```
GET /SubNetwork=SN1?\
     scopeType=BASE_ALL\
     filter=//XyzFunction[attributes[attrB>=552 and attrB<562]] HTTP/1.1
Host: example.org
Accept: application/json
```

This example returns the containment tree only.

```
GET /SubNetwork=SN1?scopeType=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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME1",
      "XyzFunction": [
        {
          "id": "XYZF1"
        },
        {
          "id": "XYZF2"
        }
      ]
    },
    {
      "id": "ME2"
    }
  ],
  "PerfMetricJob": [
    {
      "id": "PMJ1"
    }
  ],
  "ThresholdMonitor": [
    {
      "id": "TM1"
    }
  ]
}
```

When the MnS Consumer does not know the root object of the containment tree and wants to retrieve the complete tree starting with the root, the target URI needs to identify the resource above the root object. According to clause 4.4.2 this resource is identified by the path segment "/{MnSName}/{MnSVersion}", for example "/ProvMnS/1700". In this example, the empty "attributes" parameter is defined to return only the name-containment hierarchy but no attributes.

```
GET /ProvMnS/1700?scopeType=BASE_ALL&attributes= HTTP/1.1
Host: example.org
Accept: application/json
```

The response is illustrated below. Properties of the MnS may be returned as siblings of "SubNetwork", as indicated in the example below by the placeholder "…".

```
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"
        }
      ],
      "PerfMetricJob": [
        {
          "id": "PMJ1"
        }
      ],
      "ThresholdMonitor": [
        {
          "id": "TM1"
        }
      ]
    }
  ]
}
```

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 object class name of the resource to be created is present in the request. The "id" of the new resource is "XYZF3" and created by the MnS Consumer. The "id" contained in the resource representation carried in the request message body and the "id" in the target URI are identical.

```
PUT /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF3 HTTP/1.1
Host: example.org
Content-Type: application/json

{
  "id": "XYZF3",
  "objectClass": "XyzFunction",
  "attributes": {
    "attrA": "ghi",
    "attrB": 553
  }
}
```

The response contains the location header with the URI of the created resource. The response body includes the complete representation of the new resource. This is necessary, since the MnS Producer may have modified or added attributes compared to those received in the creation request. The name of the object class may or may not be present in the response.

```
HTTP/1.1 201 Created
Date: Tue, 06 Aug 2019 16:50:26 GMT
Location: http://example.org/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF3
Content-Type: application/json

{
  "id": "XYZF3",
  "attributes": {
    "attrA": "ghi",
    "attrB": 553
  }
}
```

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 value as a non-binding recommendation by the MnS Consumer. The request message body includes the object class name of the resource to be created.

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

{
  "id": "null",
  "objectClass": "XyzFunction",
  "attributes": {
    "attrA": "ghi",
    "attrB": 553
  }
}
```

For the response body the same provisions as for resource creation with HTTP PUT apply.

```
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

{
  "id": "123e4567-e89b",
  "attributes": {
    "attrA": "ghi",
    "attrB": 553
  }
}
```

When creating a root resource of the model, the path component of the request URI refers to the parent resource of the top level managed object instances as defined in clause 4.4.4.

```
POST /ProvMnS/1700 HTTP/1.1
Host: example.org
Content-Type: application/json

{
  "id": "null",
  "objectClass": "SubNetwork",
  "attributes": {
    "userLabel": "Berlin NW",
    "userDefinedNetworkType": "5G",
    "plmnId": {
      "mcc": 456,
      "mnc": 789
```

```
    }
  }
}
```

A.3.3 Creation of multiple resources with 3GPP JSON Merge Patch

One or more resources can be created with a single 3GPP JSON Merge Patch request. The following example shows the creation of a complete subtree for a new "ManagedElement" below "SubNetwork".

The target URI has been chosen to identify the first common ancestor of the resources to be created. In this case, it is the parent of the base object of the tree to be created. The "objectClass" property is present for the resources to be created.

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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
        "location": "Spandau"
      },
      "XyzFunction": [
        {
          "id": "XYZF1",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "xyz",
            "attrB": 771
          }
        },
        {
          "id": "XYZF2",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "abc",
            "attrB": 772
          }
        }
      ]
    }
  ]
}
```

The next example shows how a new "XyzFunction" resource is added to each of the "ManagedElement" resouces.

In this case, the parent of the parent of the "XyzFunction" resources to be created has been chosen as the common ancestor referenced by the target URI. The "objectClass" property is present for the resources to be created.

The "ManagedElement" resources are present with their "id" only. These resources are required to bridge the containment tree from the "SubNetwork" target resource to the created "XyzFunction" resources.

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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME1",
      "XyzFunction": [
        {
```

```
          "id": "XYZF3",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "def",
            "attrB": 553
          }
        }
      ]
    },
    {
      "id": "ME2",
      "XyzFunction": [
        {
          "id": "XYZF1",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "def",
            "attrB": 661
          }
        }
      ]
    }
  ]
}
```

A.3.4 Creation of multiple resources with 3GPP JSON Patch

One or more resources can be created with a single 3GPP JSON Patch request. The following example shows the creation of a complete subtree for a new network entity represented by a "ManagedElement" resource and two "XyzFunction" resources. The target URI has been chosen to identify the first common ancestor of the resources to be created. The "path" specifies the offset to the resource to be created. The "path" has no fragment component. Parent resources are created before child resources following the order of the operations in the patch document. The class name of the object to be created is specified in each patch operation.

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

[
  {
    "op": "add",
    "path": "/ManagedElement=ME3",
    "value": {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
        "location": "Spandau"
      }
    }
  },
  {
    "op": "add",
    "path": "/ManagedElement=ME3/XyzFunction=XYZF1",
    "value": {
      "id": "XYZF1",
      "objectClass": "XyzFunction",
      "attributes": {
        "attrA": "xyz",
        "attrB": 771
      }
    }
  },
  {
    "op": "add",
    "path": "/ManagedElement=ME3/XyzFunction=XYZF2",
    "value": {
      "id": "XYZF2",
      "objectClass": "XyzFunction",
      "attributes": {
        "attrA": "abc",
        "attrB": 772
```

```
      }
    }
  }
]
```

Note that each resource to be created shall be specified with a dedicated "add" operation. The following patch document is hence invalid as it attempts to create three resources with a single "add" operation.

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

[
  {
    "op": "add",
    "path": "/ManagedElement=ME3",
    "value": {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
        "location": "Spandau"
      },
      "XyzFunction": [
        {
          "id": "XYZF1",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "xyz",
            "attrB": 771
          }
        },
        {
          "id": "XYZF2",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "abc",
            "attrB": 772
          }
        }
      ]
    }
  }
]
```

It is not an error if the target location of an "add" operation as specified by the "path" property does exist. In this case the content of the target location is replaced with the content of the "value" property. For example, in the following example, the first "ManagedElement" resource already exists. The patch document is applied successfully though. The representation of the first "ManagedElement" resource is replaced and the second "ManagedElement" resource is created.

Note that the attributes "vendorName" and "location" are removed from the representation of the first "ManagedElement" resource. The "userLabel" attribute is updated.

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

[
  {
    "op": "add",
    "path": "/ManagedElement=ME2",
    "value": {
      "id": "ME2",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 4"
      }
    }
  },
  {
    "op": "add",
```

```
    "path": "/ManagedElement=ME3",
    "value": {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
        "location": "Spandau"
      }
    }
  }
]
```

A.4 Deletion of resources

A.4.1 Deletion of a resource with HTTP DELETE

The following example deletes an instance of "ManagedElement". The resource to be deleted is identified with the target URI. The request body is absent.

```
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

The deletion of multiple resources with a single HTTP DELETE request is not supported. The following request is hence invalid.

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

A.4.3 Deletion of multiple resources with 3GPP JSON Merge Patch

One or more descendant resources of the target URI can be deleted with a single 3GPP JSON Merge Patch request. The following example deletes the "ManagedElement" resource with "ME1" including both its "XyzFunction" resources.

The target URI has been chosen to identify the first common ancestor of the resources to be deleted. The patch document starts with the target resource. All resources of the subtree to be deleted are marked for deletion.

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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME1",
      "attributes": "null",
      "XyzFunction": [
        {
          "id": "XYZF1",
          "attributes": "null"
        },
        {
          "id": "XYZF2",
```

```
          "attributes": "null"
        }
      ]
    }
  ]
}
```

A.4.4 Deletion of multiple resources with 3GPP JSON Patch

Multiple resources are deleted with an ordered sequence of "remove" operations. The following example removes a complete subtree for a "ManagedElement".

The target URI has been chosen to identify the parent resource of the "ManagedElement" resource to be deleted. The "path" specifies the offset to the resources to be deleted. The "path" has no fragment component.

Child resources are deleted before parent resources, starting with leaf resources.

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

[
  {
    "op": "remove",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF1"
  },
  {
    "op": "remove",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF2"
  },
  {
    "op": "remove",
    "path": "/ManagedElement=ME1"
  }
]
```

A.5 Complete update of a resource

The following example updates a "XyzFunction" resource. Only the "attrA" attribute is updated with a new value "def". The "attrB" attribute is set to the old value "551", but still the "attrB" attribute needs to be present in the resource representation contained in the request message body. Otherwise "attrB" would be deleted due to the replace semantics of HTTP PUT.

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

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

When a non leaf resource is updated, contained resources are not included. For example, the following resource representation in the message body updates the "ManagedElement" resource only. It does not delete the contained "XyzFunction" resources.

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

{
  "id": "ME1",
  "attributes": {
```

```
    "userLabel": "Berlin New Label",
    "vendorName": "Company XY",
    "location": "TV Tower"
  }
}
```

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

{
  "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

{
  "id": "SN1",
  "attributes": {
    "plmnId": {
      "mcc": 654
    }
  }
}
```

In the third example the item "Metric3" is added to the array "perfMetrics". The value of "perfMetrics" contains the two old items and the new item.

```
PATCH /SubNetwork=SN1/PerfMetricJob=PMJ1 HTTP/1.1
Host: example.org
Content-Type: application/merge-patch+json

{
  "id": "PMJ1",
  "attributes": {
    "perfMetrics": ["Metric1", "Metric2, Metric3"]
    }
  }
}
```

Also in case the items of an array have an identifier, the complete updated array value needs to be present in the patch request. In the following fourth example in this clause the old first threshold level is deleted, for the old second threshold level the "value" is updated from "20" to "22", the old third threshold level is left unchanged, and a new threshold level is appended as last item.

```
PATCH /SubNetwork=SN1/ThresholdMonitor=TM1 HTTP/1.1
Host: example.org
Content-Type: application/merge-patch+json

{
```

```
  "id": "TM1",
  "attributes": {
    "thresholdLevels": [
      {
        "level": "2",
        "thresholdValue": 22
      },
      {
        "level": "3",
        "thresholdValue": 30
      },
      {
        "level": "4",
        "thresholdValue": 40
      }
    ]
  }
}
```

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

When updating a single resource, there is no difference between JSON Merge Patch (see A.6.1) and 3GPP JSON Merge Patch.

A.6.3 Partial update of a resource with JSON Patch

When JSON Patch is used to request the same changes as the ones described in the four examples 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": "def"
  }
]
```

and

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

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

and

```
PATCH /SubNetwork=SN1/PerfMetricJob=PMJ1 HTTP/1.1
Host: example.org
Content-Type: application/json-patch+json

[
  {
    "op": "add",
    "path": "/attributes/perfMetrics/2",
    "value": "Metric3"
  }
]
```

and

```
PATCH /SubNetwork=SN1/ThresholdMonotor=TM1 HTTP/1.1
Host: example.org
Content-Type: application/json-patch+json

[
  {
    "op": "remove",
    "path": "/attributes/thresholdLevels/0"
  },
  {
    "op": "replace",
    "path": "/attributes/thresholdLevels/0/thresholdValue",
    "value": 22
  },
  {
    "op": "add",
    "path": "/attributes/thresholdLevels/-",
    "value":
      {
        "level": "4",
        "thresholdValue": 40
      }
  }
]
```

Note the patch operations are applied sequentially to the "thresholdLevels" array in the order they appear in the patch array. After removing the first array item with the first operation, the resulting array value becomes the target for the second operation. The array index "0" identifies the new first item, which was the second item before applying the first operation of the patch document. Issues with array positions can be avoided by placing "replace" operations at the beginning of the patch document.

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

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

```
PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Content-Type: application/3gpp-json-patch+json

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

and

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

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

and

```
PATCH /SubNetwork=SN1/ThresholdMonitor=TM1 HTTP/1.1
```

```
Host: example.org
Content-Type: application/3gpp-json-patch+json

[
  {
    "op": "remove",
    "path": "#/attributes/thresholdLevels/0"
  },
  {
    "op": "replace",
    "path": "#/attributes/thresholdLevels/0/value",
    "value":
      {
        "thresholdValue": 22
      }
  },
  {
    "op": "add",
    "path": "#/attributes/thresholdLevels/-",
    "value":
      {
        "level": "4",
        "thresholdValue": 40
      }
  }
]
```

When using 3GPP JSON Patch to update a single resource, the only difference compared to JSON Patch is the presence of "#" in the "path".

A.7 Manipulating multiple resources

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

JSON Merge Patch allows to update one resource only with a single HTTP PATCH request. The resource must exist. In contrast, 3GPP JSON Merge Patch allows to update multiple resources incl. resource creation and deletion with a single HTTP PATCH

In the following example the "userLabel" attribute and the "mcc" attribute field of the "SubNetwork" resource are updated. The "attrB" attribute of the "XyzFunction" resource, whose "id" is "XYZF1", is also updated. A new "XyzFunction" resource with id "XYZF3"is created as well as a new "ManagedElement" resource with id "ME3". The "XYzFunction" resource, whose "id" is "XYZF2", is deleted.

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

{
  "id": "SN1",
  "attributes": {
    "userLabel": "Berlin NW-1",
    "plmnId": {
      "mcc": 654
    }
  },
  "ManagedElement": [
    {
      "id": "ME1",
      "XyzFunction": [
        {
          "id": "XYZF1",
          "attributes": {
            "attrB": 1234
          }
        },
```

```
        {
          "id": "XYZF2",
          "attributes": "null"
        },
        {
          "id": "XYZF3",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "fgh",
            "attrB": 555
          }
        }
      ]
    },
    {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
        "location": "Spandau"
      }
    }
  ]
}
```

A.7.2 Manipulating multiple resources with 3GPP JSON PATCH

The same resource modifications as in the previous clause expressed using 3GPP JSON Patch are given by

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

[
  {
    "op": "replace",
    "path": "#/attributes/userLabel",
    "value": "Berlin NW-1"
  },
  {
    "op": "replace",
    "path": "#/attributes/plmnId/mcc",
    "value": 654
  },
  {
    "op": "replace",
    "path": "ManagedElement=ME1/XyzFunction=XYZF1#/attributes/attrB",
    "value": 1234
  },
  {
    "op": "add",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF3",
    "value": {
      "id": "XYZF3",
      "objectClass": "XyzFunction",
      "attributes": {
        "attrA": "ghi",
        "attrB": 553
      }
    }
  },
  {
    "op": "remove",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF2"
  },
  {
    "op": "add",
    "path": "/ManagedElement=ME3",
    "value": {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
```

```
        "location": "Spandau"
      }
    }
  }
]
```

The two modifications of the "userLabel" attribute and the "mcc" attribute filed can also be expressed by a single "merge" operation rather than two separate "replace" operations using 3GPP JSON Patch:

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

[
  {
    "op": "merge",
    "path": "#/attributes",
    "value": {
      "userLabel": "Berlin NW-1",
      "plmnId": {
        "mcc": 654
      }
    }
  }
]
```

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
2020-12	SA#90e	SP-201088	0016	-	F	Correct REST SS specification template	16.2.0
2021-06	SA#92e	SP-210406	0017	1	F	Correct definitions of resource creation	16.3.0
2021-06	SA#92e	SP-210406	0019	-	F	Correct definition of the REST SS specification template	16.3.0
2021-09	SA#93e	SP-210886	0018	2	F	Correct definitions of resource update	16.4.0
2021-09	SA#93e	SP-210886	0021	-	F	Clarify query parameters for filtering	16.4.0
2021-12	SA#94e	SP-211454	0020	2	F	Add more examples on how to use provisioning operations	16.5.0
2022-03	-	-	-	-	-	Update to Rel-17 version (MCC)	17.0.0
2022-06	SA#96	SP-220563	0023	-	F	Add definition of secondary resource	17.1.0
2022-06	SA#96	SP-220563	0025	-	A	Add definition of resource {MnSName}{MnSVersion}	17.1.0
2022-06	SA#96	SP-220563	0027	-	A	Clarify clause Creating a resource with identifier creation by the MnS Producer	17.1.0
2022-06	SA#96	SP-220563	0029	-	A	Clarify clause Creating a resource with identifier creation by the MnS Consumer	17.1.0
2022-06	SA#96	SP-220563	0031	-	A	Clarify clause Design pattern for updating a resource	17.1.0
2022-06	SA#96	SP-220563	0033	-	A	Clarify clause Design pattern for deleting a resource	17.1.0
2022-06	SA#96	SP-220563	0035	-	A	Clarify clause Design pattern for subscribe notify	17.1.0
2022-06	SA#96	SP-220563	0037	-	A	Clarify clause Design pattern for scoping and filtering	17.1.0
2022-06	SA#96	SP-220563	0039	-	A	Clarify clause Design patterns for attribute and attribute field selection	17.1.0
2022-06	SA#96	SP-220563	0041	-	A	Clarify clause Design patterns for partially updating a resource	17.1.0
2022-06	SA#96	SP-220563	0043	-	A	Clarify clause Design patterns for patching multiple resources	17.1.0
2022-06	SA#96	SP-220563	0045	-	A	Add missing clause Large queries	17.1.0
2022-06	SA#96	SP-220563	0047	-	A	Correct examples in Annex A	17.1.0

History

Document history		
V17.0.0	April 2022	Publication
V17.1.0	July 2022	Publication