ETSI TS 132 158 V18.3.0 (2025-10)

TECHNICAL SPECIFICATION

LTE;
5G;
Management and orchestration;
Design rules for REpresentational State Transfer (REST)
Solution Sets (SS)
(3GPP TS 32.158 version 18.3.0 Release 18)

Reference
RTS/TSGS-0532158vi30

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 IPR online database.

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™**, **LTE™** and **5G™** logo 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 at 3GPP to ETSI numbering cross-referencing.

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-bhutton-json-schema-01 (June 2022): "JSON Schema: A Media Type for Describing JSON Documents".

Note: The above document is an individual draft from IETF. It cannot be formally referenced until it is published as an RFC. It is available from the following link: https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-01.

[8] draft-bhutton-json-schema-validation-01 (June 2022): "JSON Schema Validation: A Vocabulary for Structural Validation of JSON".

Note: The above document is an individual draft from IETF. It cannot be formally referenced until it is published as an RFC. It is available from the following link: https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-validation-01.

[9] draft-handrews-json-schema-hyperschema-02 (September 2019): "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON.

Note: The above document is an individual draft from IETF. It cannot be formally referenced until it is published as an RFC. It is available from the following link: https://datatracker.ietf.org/doc/html/draft-handrews-json-schema-hyperschema-02.

[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 TS 32.160: "Management and orchestration; Management service template".

[17] IETF RFC 4918: "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)"

[18] IETF RFC 6585: "Additional HTTP Status Codes"

[19] IETF RFC 7807: "Problem Details for HTTP APIs"

[20] IETF RFC 7725: "An HTTP Status Code to Report Legal Obstacles"

[21] 3GPP TS 32.161: "JSON expressions (Jex)"

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in 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 TR 21.905 [1].

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in 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 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 class is called managed object instance, or concisely just managed object or object. All managed object instances together with the relationships between them constitute an object tree. An object tree is also called containment tree.

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 one or more resource representations 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 resources. 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 mapped to secondary resources.

4.1.5 Usage of information models

Information models are used for two purposes when specifying interfaces to observe and act upon information models:

- They provide a means to identify information in request messages.
- They provide a format to transfer information in request and response messages.
- They provide constraints on the structure of information on the MnS Producer.
- They provide constraints on the possibilities to update information on the MnS Producer.

Identification of information is necessary when retrieving information from a MnS Producer; the MnS Consumer needs to be able to specify in his retrieveal request the information the MnS Producer shall return. But also when information needs to be updated or deleted the MnS Consumer needs to identify the information to be updated or deleted in his request. When information is added, the location of the new information is specified relative to the location of existing information.

Request and response message bodies carrying (some parts of) the information model are also constructed based on the information model supported by the MnS Producer. The message format is either identical to the information model format or identical to some transformation of the information model format.

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 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.

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 rule, a DN is mapped predictably into the URI authority component and path component.

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.

When no registered name can be used, the IP address shall be specified directly in the host component, for example:

```
http://168.212.226.204/SubNetwork=south/.../Cell=1
```

This might be required in multiple situations. For example, when a DN prefix is used but the corresponding URI-DN-prefix cannot be resolved, the MnS Consumer needs to specify an IP address in the target URI of HTTP request messages. The same is true when no DN prefix is used at all. Another example is when no DN prefix is configured into MnS Producers and the MnS Producer wants to report events, that occurred related to resources, using notifications sent to MnS Consumers. The MnS Producer has no other option than to put its own IP address into the host component of the URI identifying the resource where the event occurred.

4.2.4 Canonical URI

The URI defined in clause 4.2.3 is called canonical URI. It is the main or official URI of a resource. It shall be used whenever the resource as such shall be identified. The URI for sending HTTP requests to a resource may be different as described in clause 4.4. Special kinds of requests may have all their own URI. Therefore, a resource has typically one

canonical URI and one or more other URIs. The canonical URI may be looked at as a protocol specific version of the protocol neutral DN.

A canonical URI may or may not yield further information if dereferenced.

An example usage of a canonical URI is in event notifications such as alarm notifications for identifying the resource where the event occurred.

4.3 Message content formats

4.3.1 Media types

The format of HTTP request and response message content is indicated with media types consisting of a type, a subtype and optional parameters, as defined in clause 3.1.1.1 of RFC 7231 [2]. The "Content-Type" header field of a message contains the media type of the message content (clause 3.1.1.5 of RFC 7231 [2]).

If not otherwise stated, the media type of request and response message bodies in the REST SS is

- application/json (RFC 7159 [6]).

Exceptions are when JSON patch documents are contained in request bodies. They are identified with the media types

- application/merge-patch+json (RFC 7396 [12], and clause 6.3.2 of the present document),
- application/json-patch+json (RFC 6902 [13], and clause 6.3.3 of the present document).

Furthermore, this specification defines four new formats. Their media types are

- application/vnd.3gpp.merge-patch+json (clause 6.4.2 of the present document),
- application/vnd.3gpp.json-patch+json (clause 6.4.3 of the present document),
- application/vnd.3gpp.object-tree-hierarchical+json (clause 6.1.4 of the present document),
- application/vnd.3gpp.object-tree-flat+json (clause 6.1.4 of the present document).

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

4.3.2 Response content format negotiation

The MnS Consumer shall engage in proactive content negotiation as defined in clause 3.4.1 of RFC 7231 [2] by including the "Accept" request header field in HTTP requests that expect a message body in the response. The "Accept" header field indicates to the MnS Producer the media types acceptable to the MnS Consumer.

If the MnS Producer cannot provide any of the acceptable resource representations, it shall respond either with a "406 Not Acceptable" error code or provide a representation for the resource that is not specified in the "Accept" header field.

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, when being used as a target URI in HTTP requests, the 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 one or more optional path segments for structuring the resource hierarchy on a HTTP server. The DN or parts thereof shall not be mapped to this path component.
{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, containing zero, 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 TS 32.300 [3], the “LocalDN” as defined in clause 7.3 of TS 32.300 [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 colon (":"), the two slash characters ("//"), the "{authority}" part, a single slash character ("/") and the "{root}" part.

```
{MnSRoot} := {scheme}://{URI-DN-prefix}/{root}
```

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 form of the URI structure when defining Management Services.

The path segment "MnSVersion" allows access to resources with different MnS versions, for example:

```
http://operatorA.com/ProvMnS/v1500/SubNetwork=south/.../Cell=1
http://operatorA.com/ProvMnS/v1600/SubNetwork=south/.../Cell=1
```

Note that both URIs, though different as to the path segment indicating the version number of the ProvMnS, identify the same resource that is identified by the canonical URI:

```
http://operatorA.com/SubNetwork=south/.../Cell=1
```

and whose DN is:

```
DC=operatorA.com,SubNetwork=south,...,Cell=1
```

The optional path component "/{root}" may be used to separate the name space for 3GPP management from the name space for other domains:

```
http://operatorA.com/3gppManagement/ProvMnS/v1600/SubNetwork=south/.../Cell=1
```

or to provide dedicated URIs on the same host for different tasks:

```
http://operatorA.com/3gppManagement/cm/ProvMnS/v1600/SubNetwork=south/.../Cell=1
http://operatorA.com/3gppManagement/fm/ProvMnS/v1600/SubNetwork=south/.../Cell=1
```

Note that when different hosts are used for different management tasks, like in

```
http://cm.operatorA.com/3gppManagement/ProvMnS/v1600/SubNetwork=south/.../Cell=1
http://fm.operatorA.com/3gppManagement/ProvMnS/v1600/SubNetwork=south/.../Cell=1
```

then also the resources are different and identifierd by the canonical URIs

```
http://cm.operatorA.com/SubNetwork=south/.../Cell=1
http://fm.operatorA.com/SubNetwork=south/.../Cell=1
```

or the DNs

```
DC=cm.operatorA.com,SubNetwork=south,...,Cell=1
DC=fm.operatorA.com,SubNetwork=south,...,Cell=1
```

In the example above, it is assumed that both resources represent the same cell in the network. This information cannot be derived from the DN or canonical URI, though.

4.4.3 URI structure for resources not representing managed object instances

URIs identifying other resources shall follow, when being used as a target URI in HTTP requests, the 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 form of the URI structure when defining Management Services.

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

The resource identified by "../{MnSName}/{MnSVersion}" is called NRM root. It 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.

Attempts to read the NRM root only shall return "204 No Content".

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.

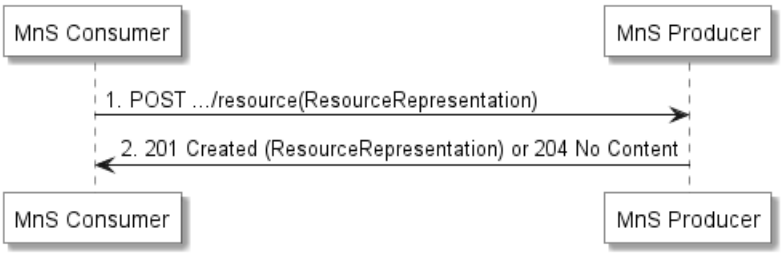

Figure 5.1.1-1: Flow for creating a resource with HTTP POST

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 response message body should carry the representation of the new resource. If the resource representation received is not modified, the MnS Producer may also return "204 No Content", instead of "201 Created". The response message body shall be empty in this case. 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.

Only resources, whose parent resource does exist, can be created (directly under that parent). The MnS Producer shall consider an attempt to create a resource, whose parent resource does not exist, as an error.

Note that the parent resource of resources for top-level (root) managed object instances is the NRM root. The NRM root always exists on MnS producers. This ensures that, when no resources for managed object instances have been created yet, the top-level resources can be created.

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.

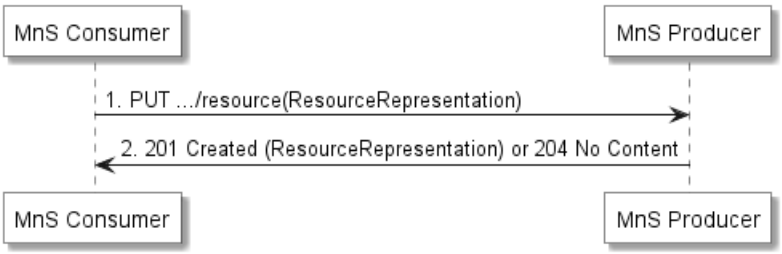

Figure 5.1.2-1: Flow for creating a resource with HTTP PUT

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 response message body shall contain the representation of the new resource. If the resource representation received is not modified, the MnS Producer may also return "204 No Content", instead of "201 Created". The response message body shall be empty in this case. 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.

As to the existence of parent resources for the resources to be created, the considerations set forth in the preceding clause for HTTP POST apply.

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.

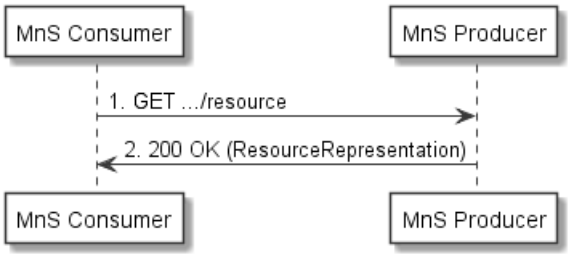

Figure 5.2-1: Flow for reading a resource

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 target URI. The target URI shall have no query and no fragment component. The "Accept" header shall be included in the request and contain the media types acceptable to the MnS Consumer. The message body shall be empty.

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.

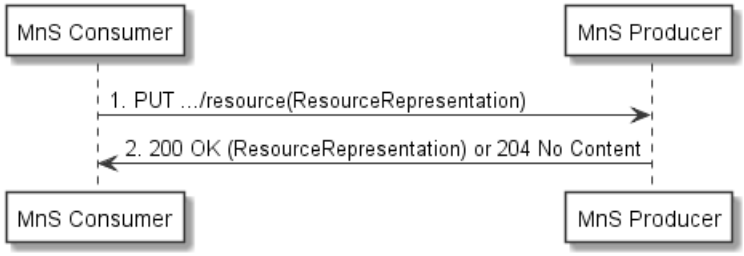

Figure 5.3-1: Flow for updating a resource

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.

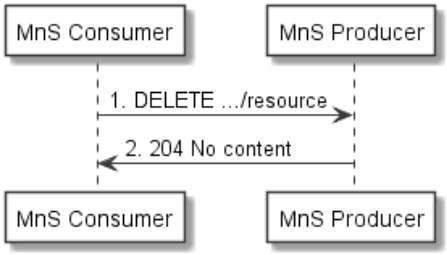

Figure 5.4-1: Flow for deleting a resource

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 response 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.

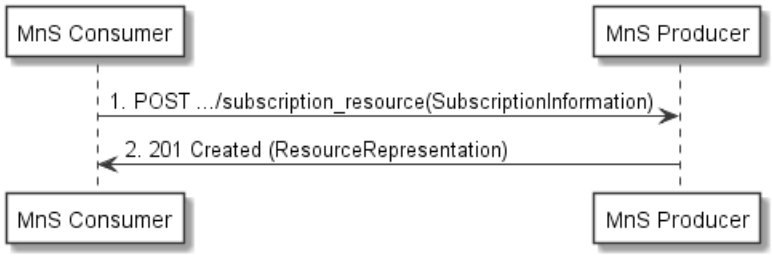

Figure 5.5.2-1: Flow for creating a subscription

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.

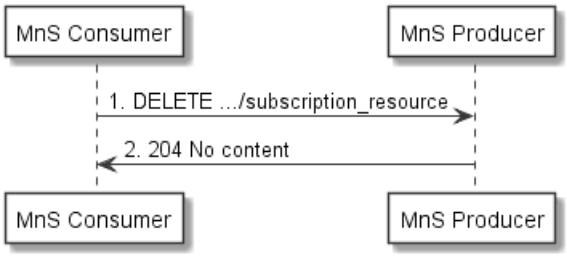

Figure 5.5.3-1: Flow for deleting a subscription

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.

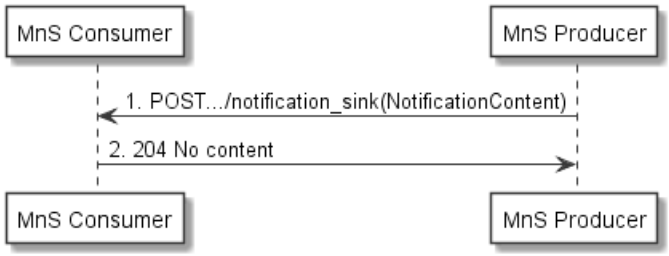

Figure 5.5.4-1: Flow for sending a notification

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.

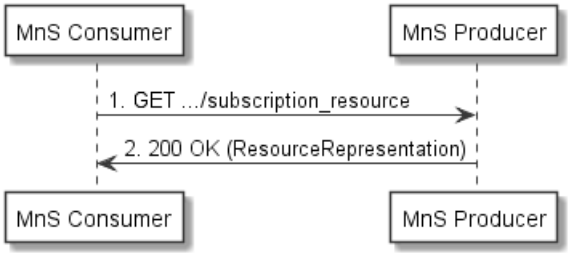

Figure 5.5.5-1: Flow for subscription retrieval

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 expression. The name of the query parameter is "filter".

Jex [21] shall be used for specifying the filter expression.

The Jex expression is applied to a JSON document document constructed based on the following rules:

- The document element is the object identified by the path component of the target URI. If the path component of the target URI identifies the NRM root (see clause 4.4.4), then the element name of the document element shall be "nrmRoot". The "nrmRoot" element contains the element nodes coming from the top-level objects as its children.
- 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.

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 Jex expression needs to be percent-encoded as described in clause 2 and 3.4 of RFC 3986 [4].

Note that NRM objects and NRM attributes are both mapped to element nodes. The children of an element node representing a NRM object are hence the NRM attributes of that NRM object or the name-contained NRM objects. This needs to be taken into account when constructing a location path for selecting element nodes representing NRM objects or NRM attributes.

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.

If no resource is identified in the retrieval request the MnS Producer shall return a "204 No Content" response.

The following media types shall be used to distinguish the flat and the hierarchical response representation:

- application/vnd.3gpp.object-tree-flat+json,
- application/vnd.3gpp.object-tree-hierarchical+json.

The "application/json" media type may be used alternatively and defaults to the hierarchical representation format.

The MnS Consumer shall indicate the acceptable representations in the "Accept" header, as described in clause 4.3.2. One or multiple media types may be specified. If the MnS Producer cannot provide an acceptable representation, a "406 Not Acceptable" error response shall be returned. The MnS Consumer may send a second request with another media type specified in the "Accept" header.

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.

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.

Note that for multi-valued attributes the selection of one or multiple attribute elements is not supported with this pattern. Furthermore, conditional attribute or attribute field selection is not supported.

6.2.3 Construction rules for the response message body

In a first step the resource identified by the target URI, or the set of resources identified by the target URI and the scope and filter parmeters, is determined. Then, in a second step, resources that do not contain at least one attribute identified by the "attributes" parameter or one attribute field identified by the "fields" parameter shall be removed from the output set of the first step. In the last step all attributes and attribute fields not identified by "attributes" and "fields" shall be removed from the remaining resource representations.

This result set is then used to construct the final response using either the hierarchical or the flat construction method, both defined in clause 6.1.4.

If no resource is identified in the retrieval request the MnS producer shall return an error response with "404 Not Found" in the status line.

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 for JSON documents called JSON Merge Patch. It allows to describe a set of modifications to be applied to the target resource representation. The JSON Merge Patch document is a partial representation of the resource to be patched. 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 needs to 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.

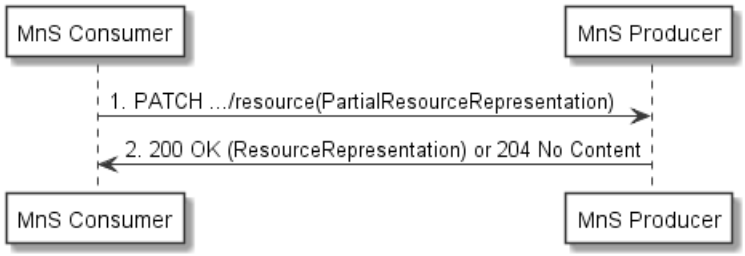

Figure 6.3.2-1: Flow for partially updating a resource with JSON Merge Patch

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 complete 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.

The following examples demonstrate the usage of JSON Merge Patch. 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"
  }
}
```

This PATCH request deletes the attribute.

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

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

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"). When removing a secondary resource, the "value" property is absent. When moving or copying an existing value, the "value" property is absent, too, and the "from" property is present instead. The "from" property identifies the secondary resource, whose value is moved or copied to the location specified by the "path" property. The value of the "from" and "path" property is a JSON Pointer in string representation as defined in section 5 of IETF 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 needs to exist, otherwise the error status code "404 Not Found " shall be returned.

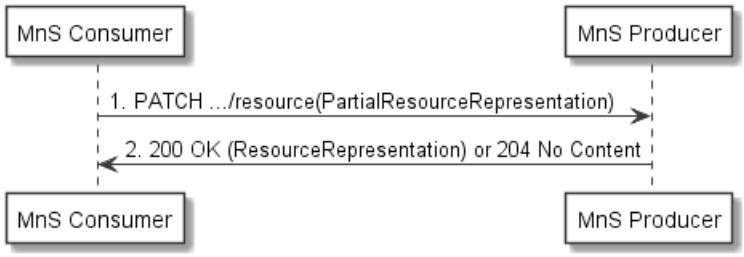

Figure 6.3.3-1: Flow for partially updating a resource with JSON Patch

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 JSON Patch document is described by the folloing JSON schema fragment.

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "op": {
        "enum": [
          "add",
          "replace",
          "remove",
          "copy",
          "move",
          "test"
        ]
      },
      "from": {
        "type": "string"
      },
      "path": {
        "type": "string"
      },
      "value": {}
    },
    "required": [
      "op",
      "path"
    ]
  }
}
```

The schema for the "value" property is the list (constructed with "anyOf") of the NRM schema fragments for all resource representations, and the NRM schema fragments for the values of all attributes and attribute fields. The NRM schema normally contains many NRM schema fragments of these kinds. For that reason it is normally not practicable to list all NRM schema fragments defining the allowed values of the "value" property. In addition, the resource, attribute

or attribute field identified in the "path" property cannt be related by the schema itself to its value schema. For these reasons, the schema "{}" is normally used, which is the shorthand syntax for a schema without any type.

The following example adds a new attribute "attrA" to an "XyzFunction" (assuming "attrA" does not exist yet).

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

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

The following example replaces the value of "attrA" 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": "def"

  }
]
```

It is not an error if the "path" property of an "add" operation specifies an object member that exists already. In this case the value of the specified object member is replaced. The following patch request has hence the same effect as the patch request in the example above. In both cases the value of "attrA" is replaced with "def".

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

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

The following patch document has not the same effect as both examples above. It does not replace the value of "attrA" with a new value. Instead, it replaces the value of the "attributes" object with a value that is an object and has a single member, the "attrA" property (attribute), thereby deleting all other attributes, that may exist when the patch request is received.

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

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

To remove the attribute "attrA" 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": "remove",
    "path": "/attributes/attrA",
  }
]
```

When the attribute to be added is a JSON array, the "value" property contains an array. In the following example the array has two items of type string.

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

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

To add a new item to an existing array, the "path" property needs to specify the array index where the item is to be added. For example, the following PATCH request adds the array item "xyz" after the first array item.

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

[
  {
    "op": "add",
    "path": "/attributes/attrB/1",
    "value": "xyz"
  }
]
```

Note that the "test" operation can be used to construct conditional patch requests. In the following example the "attrA" value is replaced only with "ghi" if the current value is "def", otherwise the test operation fails and the complete patch request is not applied.

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

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

Conditional patch requests based on the "test" operation are limited to conditions related to secondary rersources (attributes) of the target resource. It is not possible to point to secondary resources outside of the target resource using the "path" property.

Multiple test operations can be combined to construct requests with multiple conditions. All conditions need to evaluate to true for the patch document to be applied. In other words, the test operations are linked with a logical "and" operator.

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 HTTP GET request. This clause specifies methods allowing to manipulate (create, update, delete) multiple resources with a single request.

The specified methods use the HTTP PATCH method and provide extensions to the JSON Merge Patch and JSON Patch formats. As described in clause 6.3, JSON Merge Patch and JSON Patch are used for partial updates of a single resource. The extensions specified in the following clauses are designed to allow for efficient manipulation of multiple resources with a single HTTP PATCH request. The target resource and all its descendant resources are ascessible with a single request. The extended patch formats are called 3GPP JSON Merge Patch and 3GPP JSON Patch.

Note that the HTTP PATCH method is atomic as explained in clause 6.3.1.

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. In other words, the rules of the hierarchical response construction method (clause 6.1.4) apply also when constructing the 3GPP JSON Merge Patch document.

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 needs to 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 "vnd.3gpp.merge-patch+json". This media type is defined by 3GPP. It is not registered with IANA. Patch documents using this media type need to 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 and created 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 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 and delete descendant (primary) 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 a generalized offset to the target resource specified by the request URI, that allows to define not only secondary resources (within the target URI), as does JSON Patch, but also other (primary) resources and secondary resources within these other (primary) resources. The offset is relative to the target URI. It has a first component pointing (together with the target URI) to a resource below the target resource, or to the target resource, if the first component is absent. The second component points to a secondary resource within the resource identified by the first component and the request URI. When multiple root resources are to be patched, the target URI shall identify the NRM root.

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 resource 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".

Note that the MnS producer has normally a choice as to the target resource. For example, assume the resource with the URI

"http://example.com/3gpp/ProvMnS/v 1700/ManagedElement=ME1/XyzFunction=XYZF1"

is patched. Then the target resource is either the patched resource

"/example.com/3gpp/ProvMnS/v 1700/ManagedElement=ME1/XyzFunction=XYZF1",

or the parent resource of the patched resource, in this case the root resource,

"/example.com/3gpp/ProvMnS/v1700/ManagedElement=ME1",

or the NRM root.

"/example.com/3gpp/ProvMnS/v1700".

Setting the target resource always to the NRM root is hence a possible implementation option for MnS Consumers.

The target resource identified by the target URI needs to exist, otherwise the error status code "404 Not Found " shall be returned.

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 "replace" operation is not applicable when the "path" identifies a (primary) resource.

The media type of 3GPP JSON Patch is "3gpp-patch+json". This media type is defined by 3GPP and is not registered with IANA. Patch documents using this media type need to 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 (patch items) to be applied to the identified resources. The "Accept" header shall be included in the request and specify the media types acceptable to the MnS Consumer for "200 OK" or "204 No Content" responses.

2) The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated and created 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/vnd.3gpp.json-patch+json
Accept: application/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. The "merge" operation does not allow to delete secondary resources.

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/vnd.3gpp.json-patch+json
Accept: application/json
[
  {
    "op": "merge",
    "path": "#/attributes",
    "value": {
      "userLabel": "Berlin NW-1",
      "plmnId": {
        "mcc": 654
      }
    }
  }
]
```

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

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/vnd.3gpp.json-patch+json
Accept: application/json
[
  {
    "op": "merge",
    "path": "",
    "value": {
      "attributes": {
        "userLabel": "Berlin NW-1",
        "plmnId": {
          "mcc": 654
        }
      },
```

```
      "ManagedElement": [
        {
          ...
        }
      ]
    }
  }
]
```

In the same way as JSON Patch allows to construct conditional patch requests using the "test" operation, 3GPP JSON Patch can be used to construct condititional patch requests where the condition is expressed with the "test" operation. In contrast to JSON Patch, however, the condition may be based on attribute values outside of the patched resource.

For example, the following patch document replaces the value of "attrA", which is an attribute of a "XyzFunction" resource whereas the condition relates to an attribute in the "SubNetwork" resource.

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/json-patch+json
Accept: application/json
[
  {
    "op": "test",
    "path": "#/attributes/userLabel",
    "value": "Berlin NW"
  },
  {
    "op": "replace",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF1#/attributes/attrA",
    "value": "ghi"
  }
]
```

6.5 Design pattern for 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.

6.6 Design pattern for error responses

6.6.1 Introduction

If an error occurs on a MnS Producer during the processing of an HTTP request, the MnS Producer does not apply the request and returns an error response to the MnS Consumer.

This clause describes first HTTP status codes to be used in error responses and then different error response message body formats.

Note that the case of partial success, i.e. the case where some parts of the request are applied and some are not, is not covered by this clause.

6.6.2 HTTP error codes

A status code of the classes 4xx (Client Error) or 5xx (Server Error) is returned to the MnS Consumer in the error response status line. A complete list of error status codes is maintained by IANA.

Tables 6.6.2-1 and 6.6.2-2 list the status codes that shall be supported by MnS Producer and MnS Consumer implementations compliant to this specification.

Table 6.6.2-1: Supported 4xx client error status codes

Error status code	Reference	Description
400 Bad Request	IETF RFC 7231 [2]	indicates that the server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).
403 Forbidden	IETF RFC 7231 [2]	indicates that the server understood the request but refuses to authorize it.
404 Not Found	IETF RFC 7231 [2]	indicates that the origin server did not find a current representation for the target resource or is not willing to disclose that one exists.
405 Method Not Allowed	IETF RFC 7231 [2]	indicates that the method received in the request-line is known by the origin server but not supported by the target resource.
406 Not Acceptable	IETF RFC 7231 [2]	indicates that the target resource does not have a current representation that would be acceptable to the user agent, according to the proactive negotiation header fields received in the request (Section 5.3), and the server is unwilling to supply a default representation.
408 Request Timeout	IETF RFC 7231 [2]	indicates that the server did not receive a complete request message within the time that it was prepared to wait.
410 Gone	IETF RFC 7231 [2]	indicates that access to the target resource is no longer available at the origin server and that this condition is likely to be permanent.
411 Length Required	IETF RFC 7231 [2]	indicates that the server refuses to accept the request without a defined Content-Length field containing the length of the message body in the request message.
413 Payload Too Large	IETF RFC 7231 [2]	indicates that the server is refusing to process a request because the request payload is larger than the server is willing or able to process.
414 URI Too Long	IETF RFC 7231 [2]	indicates that the server is refusing to service the request because the request-target is longer than the server is willing to interpret.
415 Unsupported Media Type	IETF RFC 7231 [2]	indicates that the origin server is refusing to service the request because the payload is in a format not supported by this method on the target resource.
422 Unprocessable Entity	IETF RFC 4918 [17]	indicates the server understands the content type of the request entity (hence a 415(Unsupported Media Type) status code is inappropriate), and the syntax of the request entity is correct (thus a 400 (Bad Request) status code is inappropriate) but was unable to process the contained instructions.
426 Upgrade Required	IETF RFC 7231 [2]	indicates that the server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol.
429 Too Many Requests	IETF RFC 6585 [18]	indicates that the user has sent too many requests in a given amount of time ("rate limiting").
451 Unavailable For Legal Reasons	IETF RFC 7725 [20]	Identifies the entity that blocks access to a resource following receipt of a legal demand.

Table 6.6.2-2: Supported 5xx server error status codes

Error status code	Reference	Description
500 Internal Server Error	IETF RFC 7231 [2]	Indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented	IETF RFC 7231 [2]	indicates that the server does not support the functionality required to fulfill the request.
502 Bad Gateway	IETF RFC 7231 [2]	indicates that the server, while acting as a gateway or proxy, received an invalid response from an inbound server it accessed while attempting to fulfill the request.
503 Service Unavailable	IETF RFC 7231 [2]	indicates that the server is currently unable to handle the request due to a temporary overload or scheduled maintenance, which will likely be alleviated after some delay.
504 Gateway Timeout	IETF RFC 7231 [2]	indicates that the server, while acting as a gateway or proxy, did not receive a timely response from an upstream server it needed to access in order to complete the request.
505 HTTP Version Not Supported	IETF RFC 7231 [2]	indicates that the server does not support, or refuses to support, the major version of HTTP that was used in the request message.

A MnS Producer may use other error response codes as well. However, there is no guarantee that a MnS Consumer understands the semantics beyond what is specified in clause 6 of IETF RFC 7231 [2]: "A client MUST understand the class of any status code, as indicated by the first digit, and treat an unrecognized status code as being equivalent to the x00 status code of that class".

6.6.3 Error response body

6.6.3.1 Overview

HTTP status codes provide high level error information. This is often not sufficient, for example in situations where the MnS Producer wants to aid the MnS Consumer in generating a valid request. In these cases, the MnS Producer needs to include an error response body in the response, that contains more details on the error than the error code can provide.

The error response body specified in the present document is an extension of the problem details object defined in IETF RFC 7807 [19]. The following three properties of the problem details object are re-used for describing a problem:

- The optional "status" property that contains the status code for the error.
- The mandatory "type" property that provides high level error information.
- The optional "title" that provides a short, human-readable summary of the problem type. It shall not change from occurrence to occurrence of the problem.

Potential support for the "details" and "instance" properties is outside the provisions of the present document.

The three re-used properties are extended in the present document with the following property:

- The optional "reason" property" that provides more details on the error conditions than the "type" property.

The "status", "type", "title" and "reason" property are called generic problem details properties. They are applicable to all HTTP methods and request media types. In addition, the following method specific properties are defined:

- The optional "badQueryParams" property that provides information about bad query parameters in GET requests.
- The mandatory "badOp" property that specifies the operation in JSON Patch and 3GPP JSON Patch requests, that cannot be satisfied.
- The optional "badAttributes" property provides information about bad attributes in PUT, POST, JSON Merge Patch and 3GPP JSON Merge Patch requests.
- The optional "badObjects" property provides information about bad objects in 3GPP JSON Merge Patch requests.

A single request may have more than one problem. This situation may occur for example when a GET request has multiple bad query parameters, or when a PATCH request contains multiple bad operations. For that reason the optional "otherProblems" property is provided that allows to return one ore more additional problem detail descriptions.

A MnS Consumer cannot assume that the returned list of problems is exhaustive and includes all problems in the request. A MnS producer may stop processing the request upon detection of the first problem and return an error response.

If all problems have the same error status code, that code shall be used in the status line of the error response. The "status" property of each problem description may repeat that code. However, if the problems have different error codes, the "207 Multi-Status" (IETF RFC4918 [???]) code shall be used in the response status line. The "status" property related to each problem shall contain the specific status code.

The concrete format of the error response body depends on the request. The media type for all error response formats is "application/vnd.3gpp.error+json". The following clauses provide the details.

6.6.3.2 Error response format for GET requests

Each problem is described by the generic problem detail properties, and the additional "badQueryParams" property. The "type" property shall be present. The "status" property shall be present only under the conditions specified in clause X.2.1.

A MnS Consumer cannot assume that the returned list of bad query parameters in "badQueryParams" includes all bad parameters in the request. A MnS Producer may stop processing the request upon detection of the first bad query parameter and return an error response.

The JSON schema for the error response body is as follows.

```
{
  "type": "object",
  "properties": {
    "status": {"type": "string"},
    "type": {"type": "string"},
    "reason": {"type": "string"},
    "title": {"type": "string"},
    "badQueryParams": {"type": "array","items": {"type": "string"}},
    "otherProblems": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "status": {"type": "string"},
          "type": {"type": "string"},
          "reason": {"type": "string"},
          "title": {"type": "string"},
          "badQueryParams": {"type": "array","items": {"type": "string"}}
        }
      }
    },
    "required": ["type"]
  },
  "required": ["type"]
}
```

6.6.3.3 Error response format for PUT, POST, DELETE, JSON Merge Patch and 3GPP JSON Merge Patch requests

The error response is a JSON array of JSON objects with the generic problem details, and the "badAttributes" and "badObjects" properties. The "type" property shall be present. The "status" property shall be present only under the conditions specified in clause 6.6.3. The "badObjects" property is applicable only for 3GPP JSON Merge Patch.

The value of "badAttributes" or "badObjects" is a pointer referencing the bad node. The pointer is is a relative URI and constructed according to the rules defined in clause 6.4.3 for the "path" property of 3GPP JSON Patch.

A MnS Consumer cannot assume that the returned list of bad attributes in "badAttributes" or bad objects in "badObjects" includes all bad attributes or bad objects in the request. A MnS Producer may stop processing the request upon detection of the first bad attribute or object and return an error response.

The JSON schema for the error response body is as follows.

```
{
  "type": "object",
  "properties": {
    "status": {"type": "string"},
    "type": {"type": "string"},
    "reason": {"type": "string"},
```

```
    "title": {"type": "string"},
    "badAttributes": {"type": "array","items": {"type": "string"}},
    "badObjects": {"type": "array","items": {"type": "string"}},
    "otherProblems": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "status": {"type": "string"},
          "type": {"type": "string"},
          "reason": {"type": "string"},
          "title": {"type": "string"},
          "badAttributes": {"type": "array","items": {"type": "string"}},
          "badObjects": {"type": "array","items": {"type": "string"}}
        }
      }
    },
    "required": ["type"]
  },
  "required": ["type"]
}
```

6.6.3.4 Error response format for JSON Patch and 3GPP JSON Patch requests

Each problem is described by the generic problem detail properties, and the additional "badOp" property. The "type" and "badOp" properties shall be present. The "status" property shall be present only under the conditions specified in clause 6.6.3.

The patch operation, that cannot be satisfied, is identified with "badOp", whose value is a JSON Pointer identifying the object with the bad patch operation in the request body.

The JSON schema for the error response body is as follows.

```
{
  "type": "object",
  "properties": {
    "status": {"type": "string"},
    "type": {"type": "string"},
    "reason": {"type": "string"},
    "title": {"type": "string"},
    "badOP": {"type": "string"},
    "otherProblems": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "status": {"type": "string"},
          "type": {"type": "string"},
          "reason": {"type": "string"},
          "title": {"type": "string"},
          "badOp": {"type": "string"}
        }
      }
    },
    "required": ["type", "badOp"]
  },
  "required": ["type", "badOp"]
}
```

6.6.4 The "type" property

The "type" property provides high level error information allowing to complement HTTP 4xx and 5xx error codes in case this is necessary or desired. It provides more details on the nature of the problem than the HTTP error codes. Problem types are specified for the following error response codes.

- 400 Bad Request
- 403 Forbidden
- 422 Unprocessable Content

- 500 Internal Server Error
- 503 Service Unavailable

Note that some error codes convey already all information that can be conveyed. For example, the "404 Not Found" status code indicates that the target resource does not exist or has no current representation. It is hard to see which information should be added to make the error response more helpful for the MnS Consumer.

The "type" property is an enumeration of string values. A MnS Producer should use the following values. Other values may be used as well if deemed more appropriate for specific errors.

- VALIDATION_ERROR (HTTP error code: 400 Bad Request): The request message does not validate and cannot be processed. Validation refers to two aspects: Validation of the received request message against the JSON schema definition of the request message, and validation of the information model state after applying the requested changes against the JSON schema definition of the information model, for example, if a new instance of a certain object class is allowed to be contained under the class of the specified parent object.
- REQUEST_OBJECT_TREE_MISMATCH (HTTP error code: 422 Unprocessable Entity): The request message is well formed and understood but cannot be completed due to the current state of the object tree on the MnS Producer. For example, this reason is used when an object is requested to be created below a parent object that does not exist.
- IE_NOT_FOUND (related error code: 400 Bad Request): The information element (object, attribute, attribute field, attribute element) requested to be modified does not exist.
- MODIFICATION_NOT_ALLOWED (HTTP error code: 403 Forbidden): The requested modification is correct and understood but not allowed.
- RETRIEVAL_NOT_ALLOWED (HTTP error code: 403 Forbidden): The retrieval request is well formed and understood but the retrieval of the specified information is not allowed.
- SERVER_LIMITATION (HTTP error code: 500 Internal Server Error): The request is well formed and understood by the MnS Producer, but the MnS Producer cannot satisfy the request due to server limitations.
- SERVICE_DISABLED (HTTP error code: 503 Service Unavailable): The MnS Producer has disabled itself and is currently unable or unwilling to handle the request. This condition may occur, for example, in overload situations.
- APPLICATION_LAYER_ERROR (HTTP error code: 500 Internal Server Error): The request is well formed and understood by the MnS Producer, but the MnS Producer cannot satisfy the request due to application layer issues.

6.6.5 The "reason" property

6.6.5.1 Overview

The "reason" property provides more details on the error conditions than the "type" property. For client-side errors, these reasons may provide hints to the MnS Consumer on how to generate a request without errors. For server-side errors, they may help the MnS Consumer to generate requests that may be satisfied by the MnS Producer.

When multiple reasons apply, the most fundamental reason should be put in the "reasons" property. For example, when a MnS Consumer attempts to replace an invariant attribute, and - in addition - the attribute value is invalid, then only the information that the attribute is invariant shall be contained in the "reason" property.

The "reason" property may be omitted when the MnS Producer does not want to disclose details on the error to the MnS Consumer.

Detailed error reasons are specified by the "reason" property for the following error codes:

- 400 Bad Request
- 403 Forbidden
- 422 Unprocessable Entity

- 500 Internal Server Error

Error reasons depend on the HTTP method, the patch format, and on if attributes or objects are manipulated. The following clauses specify error reasons for the different cases. The provided reasons are not exhaustive. Other values may be used as well. The name style of these enumeration literals shall follow clause 5.3.5.3 of 3GPP TS 32.156 [?].

6.6.5.2 Error reasons for GET

Valid values for the "reason" property for an error response related to HTTP GET are:

- RESPONSE_TOO_LARGE (related type: SERVER_LIMITATION, 500 Internal Server Error): The content requested to be returned exceeds the response body size limit of the MnS Producer.
- NO_DATA_ACCESS (related type: SERVER_LIMITATION, 500 Internal Server Error): The request is correct and understood by the MnS Producer, but the MnS Producer cannot access the requested data.
- QUERY_MALFORMED (related type: VALIDATION_ERROR, 400 Bad Request): The syntax of the query component is malformed. The "badQueryParams" property shall be absent.
- QUERY_PARAM_NAMES_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more query parameter names are invalid. The "badQueryParams" property shall indicate the names of the invalid parameters.
- QUERY_PARAM_VALUES_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more query parameters have an invalid value. The "badQueryParams" property shall indicate the names of the parameters with invalid value.
- QUERY_PARAMS_MISSING (related type: VALIDATION_ERROR, 400 Bad Request): One or more query parameters, that shall be present in the request or that shall be present in case another parameter is present, are missing in the query component. The "badQueryParams" property shall indicate the names of the missing parameters.
- QUERY_PARAMS_INCONSISTENT (related type: VALIDATION_ERROR, 400 Bad Request): Query parameters with mutual dependency constraints do not respect these constraints. The "badQueryParams" property shall indicate the names of the parameters not respecting the dependency constraints.
- ATTRIBUTES_NOT_READABLE (related type: RETRIEVAL_NOT_ALLOWED ,403 Forbidden): One or more attributes or attribute fields identified by the query parameters are not readable, according to the attribute property "isReadable". The "badQueryParams" property shall indicate the names of the parameters identifying attributes that are not readable.
- QUERY_PARAMS_TOO_COMPLEX (related type: SERVER_LIMITATION, 500 Internal Server Error): The query parameters and their values are valid but one or more of them cannot be processed as requested because complexity limits of the MnS Producer are reached, for example, a filter expression is syntactically correct but cannot be evaluated and yields no results since the expression is longer or more complex than the MnS producer can or is willing to process. The "badQueryParams" property shall indicate the names of the parameters that cannot be processed.

It is not an error when query parameters do not identify anything to be returned.

Note that the following query parameters are currently specified in the present document: "scopeType", "scopeLevel", "filter", "attributes", and "fields".

Examples:

Consider the following request:

```
GET /SubNetwork=SN1?scopeType=COMPLETE_SUBTREE&scopeLevel=HIGHEST&\
     attributeFields=userLabel HTTP/1.1
Host: example.org
Accept: application/json
```

The "scopeType" and "scopeLevel" query parameters have invalid values. The query parameter "attributeField" is not defined. All problems have the same HTTP error status code. The error response may look like:

```
HTTP/1.1 400 Bad Request
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "VALIDATION_ERROR",
  "reason": "QUERY_PARAM_VALUES_INVALID",
  "title": "The value of one or more query parameters is invalid.",
  "badQueryParams": ["scopeType", "scopeLevel"],
  "otherProblems": [
    {
      "type": "VALIDATION_ERROR",
      "reason": "QUERY_PARAM_VALUES_INVALID",
      "title": "The name of one or more query parameters is invalid.",
      "badQueryParams": ["attributeFields"]
    }
  ]
}
```

In the next example the "scopeType" and "scopeLevel" query parameters have invalid values and the "fields" value is syntactically correct and valid, but too complex for the MnS Producer to process. In this case the problems have different HTTP error codes. The "207 Multi-Status" code is used in the response status line, and the "status" property of each problem details object contains to status code of that problem.

```
HTTP/1.1 207 Multi-Status
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "status": 400,
  "type": "VALIDATION_ERROR",
  "reason": "QUERY_PARAM_VALUES_INVALID",
  "title": "The value of one or more query parameters is invalid.",
  "badQueryParams": ["attributes", "fields"],
  "problemDetails": [
    {
      "status": 400,
      "type": "VALIDATION_ERROR",
      "reason": "QUERY_PARAM_NAMES_INVALID",
      "title": "The name of one or more query parameters is invalid.",
      "badQueryParams": ["attributeFields"]
    },
    {
      "status": 500,
      "type": "SERVER_LIMITATION",
      "reason": "QUERY_PARAMS_TOO_COMPLEX",
      "title": "The semantics of one or more query parameters is too complex to be
processed.",
      "badQueryParams": ["fields"]
    }
  ]
}
```

6.6.5.3 Error reasons for attribute manipulations

6.6.5.3.1 JSON Patch and 3GPP JSON Patch

This clause specifies reasons for errors that may occur when attempting to manipulate attributes of existing resources with JSON Patch and 3GPP JSON Patch. JSON Patch and 3GPP JSON Patch are used for partial resource updates.

This specification defines the following error reasons for use with JSON Patch and 3GPP JSON Patch:

- NEW_ATTRIBUTE_VALUE_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): The attribute, attribute field or attribute element, as specified in the "path" property, cannot be added, or its value cannot be replaced, as requested, because the value, as specified in the "value" property, is invalid. Valid values are determined by the attribute properties "type", "allowedValues", "multiplicity", "isOrdered", "isUnique", and "isNullable".

- NEW_ATTRIBUTE_NAME_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): The attribute or attribute field cannot be added as requested, because its name, as specified in the "path" property, is invalid.

- NEW_ATTRIBUTE_PARENT_NOT_FOUND (related type: REQUEST_OBJECTS_MISMATCH, 422 Unprocessable Entity): The attribute or attribute field cannot be added as requested, because its parent, as specified in the "path" property, does not exist.
- ATTRIBUTE_NOT_FOUND (related type: IE_NOT_FOUND, 400 Bad Request): The attribute or attribute field cannot be removed, moved, copied, or is value cannot be replaced, as requested, because the "path" or "from" property identifies an attribute or attribute field, that does not exist.
- ATTRIBUTE_ELEMENT_NOT_FOUND (related type: IE_NOT_FOUND, 400 Bad Request): The attribute element cannot be replaced, removed, moved, or copied, because the "path" or "from" property identifies an attribute element, that does not exist.
- ATTRIBUTE_INDEX_BAD (related type: IE_NOT_FOUND, 400 Bad Request): The attribute element cannot be added at the specified array location as requested, because the array element index specified in the "path" property is greater than the number of elements in the array.
- FINAL_MV_ATTRIBUTE_VALUE_INVALID (related type: REQUEST_OBJECTS_MISMATCH, 422 Unprocessable Entity): The attribute element, as specified in the "value" property cannot be added to or removed from the multi-valued attribute as requested, because this would result in an invalid value, according to the attribute properties "multiplicity" or "isUnique". The attribute element itself is valid.
- ATTRIBUTE_NOT_WRITABLE (related type: MODIFICATION_NOT_ALLOWED, 403 Forbidden): The attribute or attribute field cannot be added, removed, or moved, or its value cannot be replaced, as requested, because the attribute or attribute field is not writable by MnS Consumers, according to the attribute property "isWritable".
- ATTRIBUTE_INVARIANT (related type: MODIFICATION_NOT_ALLOWED, 403 Forbidden): The attribute or attribute field cannot be added, removed, or moved, or its value cannot be replaced, as requested, because the attribute or attribute field is invariant, according to the attribute property "isInvariant".
- OP_UNKNOWN (related type: VALIDATION_ERROR, 400 Bad Request): The patch operation specified by the "op" property is not known by the MnS producer. This situation may occur, for example, when a patch operation is not supported or wrongly spelled.

Examples:

In this example the attribute field "attrB" is requested to be replaced with a new value.

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

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

When "attrB" is invariant and its value cannot be replaced after object creation, the error response may look like:

```
HTTP/1.1 403 Not Forbidden
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "MODIFICATION_NOT_ALLOWED",
  "reason": "ATTRIBUTE_INVARIANT",
  "title": "The attribute, whose value is requested to be replaced, is invariant.",
  "badOp": "/0"
}
```

6.6.5.3.2 JSON Merge Patch, 3GPP JSON Merge Patch and PUT

This clause specifies reasons for errors that may occur when attempting to manipulate attributes of existing resources with JSON Merge Patch, 3GPP JSON Merge Patch and PUT. JSON Merge Patch and 3GPP Merge JSON Patch are used for partial resource updates. PUT is used for complete resource updates.

The following error reasons are defined for use with JSON Merge Patch, 3GPP JSON Merge Patch, and PUT:

- NEW_ATTRIBUTE_VALUE_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more attributes or attribute fields cannot be added, or their values cannot be replaced, as requested, because the received value is invalid. Valid values are determined by the attribute properties "type", "allowedValues", "multiplicity", "isOrdered", "isUnique", and "isNullable". The "badAttributes" property provides the path to these attributes and attribute fields.

- NEW_ATTRIBUTE_NAME_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more attributes or attribute fields cannot be added as requested, because the received attribute or attribute field name is invalid. The "badAttributes" property provides the path to these attributes and attribute fields.

- ATTRIBUTE_NOT_WRITABLE (related type: MODIFICATION_NOT_ALLOWED, 403 Forbidden): One or more attributes or attribute fields cannot be added or removed, or their values cannot be replaced, as requested, because the attributes or attribute fields are not writable by MnS Consumers, according to the attribute property "isWritable". The "badAttributes" property provides the path to these attributes and attribute fields.

- ATTRIBUTE_INVARIANT (related type: MODIFICATION_NOT_ALLOWED, 403 Forbidden): One or more attributes or attribute fields cannot be added or removed, or their values cannot be replaced, as requested, because the attributes or attribute fields are invariant, according to the attribute property "isInvariant". The "badAttributes" property provides the path to these attributes and attribute fields.

The following additional error reasons are defined for use with JSON Merge Patch and 3GPP JSON Merge Patch:

- ATTRIBUTE_NOT_FOUND (related type: IE_NOT_FOUND, 400 Bad Request): One or more attribute or attribute fields cannot be removed as requested, because they do not exist. The "badAttributes" property provides the path to these attributes and attribute fields.

Examples:

In this example the MnS Consumer requests to replace the current value of "attrB" with "def".

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

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

When "attrB" is invariant the MnS Producer might respond as follows.

```
HTTP/1.1 403 Forbidden
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "MODIFICATION_NOT_ALLOWED",
  "reason": "ATTRIBUTE_INVARIANT",
  "title": "The attribute field, whose value is requested to be replaced, is invariant.",
  "badAttributes": ["#/attributes/attrA/attrB"]
}
```

6.6.5.4 Error reasons for object manipulations

The following reasons are defined for errors that may occur when attempting to create objects with PUT, POST. 3GPP JSON Merge Patch and 3GPP JSON Patch, or when attempting to delete objects with DELETE, 3GPP JSON Merge Patch and 3GPP JSON Patch:

- OBJECT_CREATION_NOT_ALLOWED (related type: MODIFICATION_NOT_ALLOWED, 403 Forbidden): One or more objects cannot be created as requested because objects of this class cannot be created by MnS Consumers.
- OBJECT_DELETION_NOT_ALLOWED (related type: MODIFICATION_NOT_ALLOWED, 403 Forbidden): One or more objects cannot be deleted as requested, because objects of this class cannot be deleted by MnS Consumers.
- NEW_OBJECT_CLASS_NAME_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more objects cannot be created as requested, because the receive object class name is unknown to the MnS Producer.
- NEW_OBJECT_REPRESENTATION_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more objects cannot be created as requested, because the received object representation does not validate.
- NEW_OBJECT_CONTAINMENT_INVALID (related type: VALIDATION_ERROR, 400 Bad Request): One or more objects cannot be created under the specified parent as requested, because this containment is not allowed.
- NEW_OBJECTS_ID_EXISTS (related type: REQUEST_OBJECTS_MISMATCH, 422 Unprocessable Content): One or more objects cannot be created as requested, because the received "id" exists already under the specified parent.
- NEW_OBJECTS_PARENT_NOT_FOUND (related type: REQUEST_OBJECTS_MISMATCH, 422 Unprocessable Content): One or more objects cannot be created as requested, because their specified parents do not exist.
- NEW_OBJECT_ATTRIBUTE_VALUE_MISSING (related type: VALIDATION_ERROR, 400 Bad Request): One or more objects cannot be created as requested, because attribute or attribute field values, that shall be provided in the creation request, are not provided.
- OBJECTS_CARDINALITY_INVALID (related type: REQUEST_OBJECTS_MISMATCH, 422 Unprocessable Content): One or more objects cannot be created or deleted as requested, because this would result in violating cardinality constraints.
- OBJECT_NOT_A_LEAF (related type: REQUEST_OBJECTS_MISMATCH, 422 Unprocessable Content): One or more objects cannot be deleted as requested, because they are not leaf objects.
- OBJECT_NOT_FOUND (related type: IE_NOT_FOUND, 400 Bad Request): One or more objects cannot be deleted as requested, because they do not exist.
- OP_UNKNOWN (related type: VALIDATION_ERROR, 400 Bad Request): The patch operation specified by the "op" property is not known by the MnS Producer. This situation may occur, for example, when a patch operation is not supported or wrongly spelled.

The error reason "NEW_OBJECT_REPRESENTATION_INVALID" provides no information on why the representation of the resource requested to be created is invalid. A MnS Producer may decide to provide more details by specifying the error reasons related to attributes defined in clause X.4.3.2 instead of the general reason "NEW_OBJECT_REPRESENTATION_INVALID". The attributes or attribute fields with problems are specified by the "badAttributes" property.

PUT example:

In this example a MnS Producer requests the creation of a resource using PUT.

```
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
  }
}
```

When the resource representation provided in the request is invalid the MnS Producer may send the following error response.

```
HTTP/1.1 400 Bad Request
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "VALIDATION_ERROR",
  "reason": "NEW_OBJECT_REPRESENTATION_INVALID",
  "title": "The object cannot be created because its representation is invalid."
}
```

The MnS Producer may also choose to provide more details on why the resource representation is invalid. For example, when the attribute name "attrB" is invalid, the MnS Producer may return the following error response.

```
HTTP/1.1 400 Bad Request
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "VALIDATION_ERROR",
  "reason": "NEW_ATTRIBUTE_NAME_INVALID",
  "title": "The object representation is invalid because an attribute name is invalid.",
  "badAttributes": ["#/attributes/attrB"]
}
```

It is possible that the request fails for multiple reasons. For example, the object representation might be invalid, and the "id" of the resource requested to be created does already exist.

```
HTTP/1.1 207 Multi-Status
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "status": 400,
  "type": "VALIDATION_ERROR",
  "reason": "NEW_OBJECT_REPRESENTATION_INVALID",
  "title": "The object cannot be created because its representation is invalid.",
  "otherProblems": [
    {
      "status": 422,
      "type": "REQUEST_OBJECTS_MISMATCH",
      "reason": "NEW_OBJECTS_ID_EXISTS",
      "title": "The object cannot be created because the object id exists already."
    }
  ]
}
```

DELETE example:

In this example a MnS Producer requests the deletion of a resource using DELETE.

```
DELETE /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF3 HTTP/1.1
Host: example.org
```

When the object to be deleted does not exist the MnS Producer may send

```
HTTP/1.1 404 Not Found
Date: Tue, 06 Aug 2019 16:50:26 GMT
```

When the object does exist but cannot be deleted, because it is not a leaf, the arror response may be as follows.

```
HTTP/1.1 422 Unprocessable Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json
```

```
{
  "type": "REQUEST_OBJECTS_MISMATCH",
  "reason": "OBJECT_NOT_A_LEAF",
  "title": "The object cannot be deleted because it is not a leaf.",
}
```

The MnS Producer can also return multiple reasons why a request fails. For example, when the object requested to be deleted is not a leaf, and could not be deleted even if it were a leaf due to cardinality constraints, the MnS Producer may return the following.

```
HTTP/1.1 422 Unprocessable Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "status": 422,
  "type": "REQUEST_OBJECTS_MISMATCH",
  "reason": "OBJECT_NOT_A_LEAF",
  "title": "The object cannot be deleted because it is not a leaf.",
  "otherProblems": [
    {
      "status": 422,
      "type": "REQUEST_OBJECTS_MISMATCH",
      "reason": "OBJECTS_CARDINALITY_INVALID",
      "title": "The object cannot be created because of cardinality constraints."
    }
  ]
}
```

In the previous example all problems have the same error code. For that reason the error codes can be omitted in the response body.

```
HTTP/1.1 422 Unprocessable Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "REQUEST_OBJECTS_MISMATCH",
  "reason": "OBJECT_NOT_A_LEAF",
  "title": "The object cannot be deleted because it is not a leaf.",
  "otherProblems": [
    {
      "type": "REQUEST_OBJECTS_MISMATCH",
      "reason": "OBJECTS_CARDINALITY_INVALID",
      "title": "The object cannot be created because of cardinality constraints."
    }
  ]
}
```

3GPP JSON Patch example:

Assume the following patch is applied to an object tree, that has one "SubNetwork" instance only. The first operation requests to create a "ManagedElement". This operation is successful. The second operation requests to create a "HuhuFunction" object under the new object. The "HuhuFunction" is not known to the MnS Producer. This operation fails. The third operation fails as well, since it requests to create a new object under an object that does not exist.

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

[
  {
    "op": "add",
    "path": "/ManagedElement=ME1",
    "value": {
      "id": "ME3",
      "objectClass": "ManagedElement",
      "attributes": {
        "userLabel": " Berlin NW 3",
        "vendorName": "Company XY",
        "location": "Spandau"
      }
    }
  },
  {
```

```
    "op": "add",
    "path": "/ManagedElement=ME1/HuhuFunction=HUHUF1",
    "value": {
      "id": "XYZF1",
      "objectClass": "XyzFunction",
      "attributes": {
        "attrA": "xyz",
        "attrB": 771
      }
    }
  },
  {
    "op": "add",
    "path": "/ManagedElement=ME3/XyzFunction=XYZF1",
    "value": {
      "id": "XYZF2",
      "objectClass": "XyzFunction",
      "attributes": {
        "attrA": "abc",
        "attrB": 772
      }
    }
  }
]
```

The error response may look like:

```
HTTP/1.1 207 Multi-Status
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "status": 400,
  "type": "VALIDATION_ERROR",
  "reason": "NEW_OBJECT_CLASS_NAME_INVALID",
  "title": "The class of the new object to be created is invalid.",
  "badOp": "/1",
  "otherProblems": [
    {
      "status": 422,
      "type": "REQUEST_OBJECTS_MISMATCH",
      "reason": "NEW_OBJECTS_PARENT_NOT_FOUND",
      "title": "The parent object of the new object to be created does not exist."
      "badOp": "/2"
    }
  ]
}
```

3GPP JSON Merge Patch example:

Assume the "ManagedElement" with the identifier "ME3" does not exist. Then the following message requests to create two new objects under a non-existent object. This request cannot be satisfied.

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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME3",
      "XyzFunction": [
        {
          "id": "XYZF1",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "xyz",
            "attrB": 771
          }
        },
        {
          "id": "XYZF2",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "abc",
```

```
            "attrB": 772
          }
        }
      ]
    }
  ]
}
```

The error message may look like:

```
HTTP/1.1 422 Unprocessable Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "REQUEST_OBJECTS_MISMATCH",
  "reason": "NEW_OBJECT_PARENT_NOT_FOUND",
  "title": "The object, below which new objects are requested to be created, does not exist.",
  "badObjects": [
    "/ManagedElement=ME3/XyzFunction=XYZF1",
    "/ManagedElement=ME3/XyzFunction=XYZF2"
  ]
}
```

6.6.6 Error reasons for application layer errors

Error reasons for the error type "APPLICATION_LAYER_ERROR" are very dependent on the specific application. Therefore, it is almost impossible to define application layer error reasons that are applicable to more than one application.

This specification defines the following values for the "reason" property:

- RESOURCE_LOCKED (related type: RETRIEVAL_NOT_ALLOWED ,403 Forbidden): The resource was locked by administrative action and cannot be accessed.
- SERVICE_LOCKED (HTTP error code: 503 Service Unavailable): The MnS Producer has been locked by administrative action and is currently unable to handle the request. This condition may occur, for example, due to scheduled maintenance. The "reason" property shall be absent.

Examples:

In the following example a MnS Consumer requests the creation of a "PerfMetricJob" instance indicating that "metric1" and "metric2" shall be collected for "obj1" and "obj2" with a granularity period if 5min.

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

[
  {
    "op": "add",
    "path": "/PerfMetricJob=PMJ1",
    "value": {
      "PerfMetricJob": [
        {
          "id": "PMJ1",
          "objectClass": "PerfMetricJob",
          "objectInstance": "SubNetwork=SN1,PerfMetricJob=PMJ1",
          "attributes": {
            "granularityPeriod": "5",
            "perfMetrics": [
              "metric1",
              "metric2"
            ],
            "objectInstances": [
              "obj1",
              "obj2"
            ]
          }
        }
      ]
    }
```

```
  }
]
```

When the requested granularity period is not supported, the "PerfMetricJob" instance is not created. The MnS Producer might answer with the following error response.

```
HTTP/1.1 400 Bad Request
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.error+json

{
  "type": "APPLICATION_LAYER_ERROR",
  "reason": "GRANULARITY_PERIOD_NOT_SUPPORTED",
  "title": "The requested granularity period for metric collection is not supported."
}
```

6.6.7 Security considerations

When the MnS Consumer is not trustworthy or the MnS Producer does not want to disclose error details, just the "type" property may be included in the error response. The response body may be omitted also completely, and just the error status code be returned in the response status line.

6.7 Design pattern for conditional data node selection

Scoping with the query parameters "scopeType" and "scopeLevel", and filtering with the query parameter "filter" allows for conditional object selection. The query parameters "attributes" and "fields" allow for (unconditional) selection of attributes and attribute fields.

For multi-valued attributes, where the attribute elements itself are big complex data types, it may be desirable to select also attribute elements based on conditions. For example, assume an alarm list object that has a multi-valued attribute containing alarm records. For retrieving only alarm records with a certain perceived severity it needs to be possible to filter on the perceived severity and return only the alarm records matching that filter criteria. But also attributes or attribute fields may need to be selected based on conditions. For example, assume a managed object that can be locked and disabled. When locked or disabled some state attributes are not updated any more and do not reflect the current state. Reading of these state attributes is hence only meaningful when the object is neither locked nor disabled.

Conditional attribute data node selection is similar to conditional object selection with the "filter" query parameter. Instead of specifying a query parameters for conditional object selection and another parameter for conditional attribute data node selection, these selection mechanisms may be combined into a single query parameter.

The name of this query parameter is "dataNodeSelector". Jex [21] shall be used for specifying the selection expression.

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

```
            "granularityPeriod": "5",
            "perfMetrics": [
              "Metric1",
              "Metric2"
            ],
            "objectInstances": [
              "Obj1",
              "Obj2"
            ]
          }
        }
      ],
      "ThresholdMonitor": [
        {
          "id": "TM1",
          "objectClass": "ThresholdMonitor",
          "objectInstance": "DC=example.org,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" ?>
<nrmRoot>
  <SubNetwork>
      <id>SN1</id>
      <objectClass>SubNetwork</objectClass>
      <objectInstance>DC=example.org,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>DC=example.org,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> DC=example.org,SubNetwork=SN1,\
                              ManagedElement=ME1,XyzFunction=XYZF1</objectInstance>
              <attributes>
                  <attrA>xyz</attrA>
                  <attrB>551</attrB>
              </attributes>
          </XyzFunction>
          <XyzFunction>
              <id>XYZF2</id>
              <objectClass>XyzFunction</objectClass>
              <objectInstance> DC=example.org,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>
</nrmRoot>
```

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.

Furthermore, the value of query parameters is not always percent-encoded, as defined in clause 2 and 3.4 of RFC 3986 [4], for better readability.

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 a JSON object with 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
  }
}
```

The MnS Consumer might request also to return a response constructed according to the flat response construction method. In this case the "Accept" header contains the " application/vnd.3gpp.object-tree-flat+json" media type.

```
GET /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Accept: application/vnd.3gpp.object-tree-flat+json
```

The response is a JSON array with a single item, which is a JSON object with the resource representation. Note that the resource representation contains the "objectClass" and "objectInstance" in this case.

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/ vnd.3gpp.object-tree-flat+json

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

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 the next 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
```

The base object and all objects at scope level "1", irrespective of their object class, are included in the response. The acceptable response media type specified by the "Accept" header field is "application/json", which indicates to the MnS producer to use the hierarchical response construction method

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

{
  "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 MnS Consumer can request also to return a response constructed according to the flat response construction method. In this case the "Accept" header contains the "application/vnd.3gpp.object-tree-flat+json" media type.

```
GET /SubNetwork=SN1?scopeType=BASE_SUBTREE&scopeLevel=1 HTTP/1.1
Host: example.org
Accept: application/vnd.3gpp.object-tree-flat+json
```

The response looks like:

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/vnd.3gpp.object-tree-flat+json

[
  {
    "id": "SN1",
    "objectClass": "SubNetwork",
    "objectInstance": "DC=example.org,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": "DC=example.org,SubNetwork=SN1,ManagedElement=ME2",
    "attributes": {
      "userLabel": "Berlin NW 2",
      "vendorName": "Company XY",
      "location": "Grunewald"
    }
  },
  {
    "id": "PMJ1",
    "objectClass": "PerfMetricJob",
    "objectInstance": "DC=example.org,SubNetwork=SN1,PerfMetricJob=PMJ1",
    "attributes": {
      "granularityPeriod": "5",
      "perfMetrics": [
        "Metric1",
        "Metric2"
      ],
      "objectInstances": [
        "Obj1",
        "Obj2"
      ]
    }
  },
  {
    "id": "TM1",
    "objectClass": "ThresholdMonitor",
    "objectInstance": "DC=example.org,SubNetwork=SN1,ThresholdMonitor=TM1",
    "attributes": {
      "metric": "Metric1",
      "thresholdLevels": [
        {
          "level": "1",
          "thresholdValue": 10
        },
        {
          "level": "2",
          "thresholdValue": 20
        },
        {
          "level": "3",
          "thresholdValue": 30
        }
      ]
    }
  }
]
```


The "objectInstance" of each returned object is present in the response, as required in clause 6.1.4.

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:


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

{
  "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 two "XyzFunction" objects. The "SubNetwork" and "ManagedElement" are present with their "id" only; they provide the containment nodes for the "XyzFunction" objects.

```
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",
          "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 the two "XyzFunction" objects without containment nodes.

```
HTTP/1.1 200 OK
Date: Tue, 06 Aug 2019 16:50:26 GMT
Content-Type: application/json
[
  {
    "id": "XYZF1",
    "objectClass": "XyzFunction",
    "objectInstance": "DC=example.org,SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF1",
    "attributes": {
      "attrA": "xyz",
      "attrB": 551
    }
  },
  {
    "id": "XYZF2",
    "objectClass": "XyzFunction",
    "objectInstance": "DC=example.org,SubNetwork=SN1,ManagedElement=ME1,XyzFunction=XYZF2",
    "attributes": {
      "attrA": "abc",
      "attrB": 552
    }
  }
]
```

The following example selects all objects of any class 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:

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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME2",
      "attributes": {
        "userLabel": "Berlin NW 2",
        "vendorName": "Company XY",
        "location": "Grunewald"
      }
    }
  ]
}
```

The input document to the XPath expression is a document whose root node is the object identified by the path component of the target URI and that includes the object representations of the scoped objects. In this example the root node is the "SubNetwork", but it is not scoped and hence included in the input document with its "id" only, i.e. without the "attributes" node. The input document includes furthermore all scoped objects on level "1" with their complete representations (without name-contained objects). These are the two "ManagedElement" objects, the "PerfMetricJob" object, and the "ThresholdMonitor" object.

```
{
  "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 Note that a valid XML document has one and only one root element. For that reason the "SubNetwork" element needs to be added as root element..

```
<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:

```
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": "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
```

It is possible to combine scoping and filtering with attribute and attribute field selection. The following example returns the containment tree, starting with the "SubNetwork" identified by the target URI.

```
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"
    }
  ]
}
```

The next example scopes the same subtree as in the previous example and requests to return only "vendorName" attributes instead of no attributes at all.

```
GET /ProvMnS/v1700?\
      scopeType=BASE_ALL&\
      attributes=vendorName HTTP/1.1
Host: example.org
Accept: application/json
```

This results, according to clause 6.2.3, in removing from the response all scoped resources that do not have a "vendorName" attribute.

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

{
  "id": "SN1",
  "ManagedElement": [
    {
      "id": "ME1",
      "attributes": {
        "vendorName": "Company XY"
      }
    },
    {
      "id": "ME2",
      "attributes": {
        "vendorName": "Company XY"
      }
    }
  ]
}
```

If the retrieval request identifies resources that do not exist, such as in

```
GET /ProvMnS/v1700?scopeType=BASE_NTH_LEVEL&scopeLevel=3 HTTP/1.1
Host: example.org
Accept: application/json
```

The MnS producer returns a "204 No Content" response.

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

When the MnS Consumer does not know the root objects of the containment tree and wants to retrieve the complete trees starting with the roots, the target URI needs to identify the NRM root, i.e. the resource above the root objects. According to clause 4.4.2, this resource is identified by the path segment "/{MnSName}/{MnSVersion}", for example "/ProvMnS/v1700". In the following example, the "attributes" query parameter is empty and only the name-containment hierarchy (without attributes) is returned.

```
GET /ProvMnS/v1700?scopeType=BASE_ALL&attributes= HTTP/1.1
Host: example.org
Accept: application/json
```

The response is illustrated below.

```
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"
        }
      ]
    }
  ]
}
```

Multiple root resources can be returned as well. For example, assume a NRM with three "SubNetwork" root resources, then the response may look like:

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

{
  "SubNetwork": [
    {
      "id": "SN1",
      ...
    },
    {
      "id": "SN2",
      ...
    },
    {
      "id": "SN3",
      ...
    }
  ]
}
```

Note that when the target URI identifies the NRM root, then the name of the document (root) element, to which an XPath expression is applied, is "nrmRoot". The first step of the location path of an XPath expression is hence "/nrmRoot". For example, the following HTTP GET request returns the "SubNetwork" with the identifier "SN1".

```
GET /ProvMnS/v1700?\
      scopeType=BASE_ALL&\
      filter=/nrmRoot/SubNetwork[id="SN1"]/attributes HTTP/1.1
Host: example.org
Accept: application/json
```

Note the presence of the location step "/attributes". This step is necessary to select only the "attributes" container and hence only the SubNetwork" with the identifier "SN1" without any name-contained objercts.

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

{
  "SubNetwork": [
    {
      "id": "SN1",
      "objectClass": "SubNetwork",
      "objectInstance": "DC=example.org,SubNetwork=SN1",
      "attributes": {
        "userLabel": "Berlin NW",
        "userDefinedNetworkType": "5G",
        "plmnId": {
          "mcc": 456,
          "mnc": 789
        }
      }
```

```
    }
  ]
}
```

Without the location step "/attributes" the complete subtree would be returned.

In all examples above query parameter values are not percent-encoded for better readability. For example, the value of the filter query parameter in the following request

```
GET /ProvMnS/v1700?\
      scopeType=BASE_ALL&\
      filter=/nrmRoot/SubNetwork[id="SN1"]/attributes HTTP/1.1
Host: example.org
Accept: application/json
```

needs to be percent-encoded.

```
GET /ProvMnS/v1700?\
      scopeType=BASE_ALL&\
      filter=%2FnrmRoot%2FSubNetwork%5Bid%3D%22SN1%22%5D%2Fattributes HTTP/1.1
Host: example.org
Accept: application/json
```

A.2.4 Large queries

The following example shows how to construct a GET request using method override.

```
POST /ProvMnS/v1700 HTTP/1.1
Host: example.org
X-HTTP-Method-Override: GET
Content-Type: application/x-www-form-urlencoded
Accept: application/json

scopeType=BASE_ALL&filter=%2FnrmRoot%2FSubNetwork%5Bid%3D%22SN1%22%5D%2Fattributes
```

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
  }
}
```

If the HTTP PUT request succeeds, the status code "201 Created" is returned in the response status line. The location header is present, its value is the URI of the created resource. The response message body contains the complete representation of the new resource. 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
  }
}
```

In this example, the MnS Producer creates the object with the attribute name/value pairs as provided in the request. For that reason, "204 No Content" may be returned in the status line instead of "201 Created". The response message body is absent in this case.

```
HTTP/1.1 204 No Content
Date: Tue, 06 Aug 2019 16:50:26 GMT
Location: http://example.org/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF3
Content-Type: application/json
```

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/v1700 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/vnd.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 MnS Producer might respond as follows to indicate the PATCH request was successful and the received resource representation was not modified.

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

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/vnd.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
          }
        }
      ]
    }
  ]
}
```

The MnS Producer might respond again as follows to indicate the successful creation of the resources.

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

Assume now that for "XyzFunction" a third attribute "attrC" is defined and that this attribute has a default value of "5". The MnS Producer assigns the default value after reception of the PATCH request and before creating the resource when no value is provided for "attrC" in the request. In this case the response includes the modified resource representations.

```
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": "XYZF3",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "def",
            "attrB": 553,
            "attrC": 5
          }
        }
      ]
    },
    {
      "id": "ME2",
      "XyzFunction": [
        {
          "id": "XYZF1",
          "objectClass": "XyzFunction",
          "attributes": {
            "attrA": "def",
            "attrB": 661,
```

```
            "attrC": 5
          }
        }
      ]
    }
  ]
}
```

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 from the target resource 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. The "Accept" header specifies responses with hierarchcal object tree are acceptable.

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

The MnS Producer might respond as follows.

```
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/vnd.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/vnd.3gpp.json-patch+json
Accept: application/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 "thresholdValue" 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 that 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.

In the examples above the value of "value" is always a simple type (scalar value). When multiple attribute fields of an attribute need to be added or replaced, it is often more compact to add or replace the complete attribute with a single patch operation, instead of each attribute field individually. For example, the following patch

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

[
  {
    "op": "add",
    "path": "/attributes/plmnId/mcc",
    "value": 456
  },
  {
    "op": "add",
    "path": "/attributes/plmnId/mnc",
    "value": 789
  }
]
```

can be replaced by

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

[
  {
    "op": "add",
    "path": "/attributes/plmnId",
    "value": {
       "mcc": 456,
       "mnc": 789
    }
  }
]
```

When adding a member to a JSON object, the JSON object needs to exist. Assume "plmnId" does not exist, but "attributes" does, then the following request is an error, since it attempts to add a "mcc" member to the "plmnId" object, that does not exist

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

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

The MnS Consumer should send the following instead.

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

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

Alternatively, an empty "plmnId" object could be created before adding the "mcc" member.

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

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

Replacing all attribute values of an object is a special case of a partial resource update. The following example demonstrates the usage of a compact format where the "attributes" container is replaced completely. It is not necessary to specify a patch operation for each attribute of the object.

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

[
  {
    "op": "replace",
    "path": "/attributes",
    "value": {
      "attrA": "def",
      "attrB": 123
    }
  }
]
```

Note that clause 4.3 of IETF RFC 6902 [13] does not consider it as an error if an attribute value is replaced with exactly the same value. For that reason it would not be an error if in the example above an attribute value is included in the "value" property that is equal to the value in the current resource representation. A MnS Producer may consider this compact format hence also for the case that not all attributes of an object are requested to be updated with a new value.

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/vnd.3gpp.json-patch+json
Accept: application/json
[
  {
    "op": "replace",
    "path": "#/attributes/attrA",
    "value": "def"
  }
]
```

and

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/vnd.3gpp.json-patch+json
Accept: application/json
[
  {
    "op": "replace",
    "path": "#/attributes/plmnId/mcc",
    "value": 654
  }
]
```

and

```
PATCH /SubNetwork=SN1/ThresholdMonitor=TM1 HTTP/1.1
Host: example.org
Content-Type: application/vnd.3gpp.json-patch+json
Accept: application/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
      }
  }
]
```

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 needs to 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/vnd.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/vnd.3gpp.json-patch+json
Accept: application/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 modifications of the "userLabel" attribute and the "mcc" attribute field can be expressed also by a single "merge" operation instead of two separate "replace" operations.

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/vnd.3gpp.json-patch+json
Accept: application/json
[
  {
    "op": "merge",
    "path": "#/attributes",
    "value": {
      "userLabel": "Berlin NW-1",
      "plmnId": {
        "mcc": 654
      }
    }
  }
]
```

The "copy" operation is useful when complete configurations from existing resources need to be copied to newly created resources.

```
PATCH /SubNetwork=SN1 HTTP/1.1
Host: example.org
Content-Type: application/vnd.3gpp.json-patch+json
Accept: application/json
[
  {
    "op": "add",
    "path": "/ManagedElement=ME1/XyzFunction=XYZF3",
    "value": {
      "id": "XYZF3",
      "objectClass": "XyzFunction",
      "attributes": {
      }
    }
  },
  {
    "op": "copy",
    "from": "/ManagedElement=ME1/XyzFunction=XYZF2/attributes"
    "path": "/ManagedElement=ME1/XyzFunction=XYZF3/attributes"
```

```
  }
]
```

A.8 Partitioning a data model

All objects of the data model in annex A.1 may be accessed and manipulated via a single MnS Producer endpoint, for example

```
http://example.org/3gpp/ProvMnS/v1600
```

An implementation may also provide more than one endpoint for accessing the data model. This may be for allowing MnS Producers supporting different versions of the CRUD operations to access the data model:

```
http://example.org/3gpp/ProvMnS/v1600
http://example.org/3gpp/ProvMnS/v1700
```

Another reason might be to structure the total data model into subsets of managed objects for different purposes such as configuration management and performance management.

```
http://example.org/3gpp/cm/ProvMnS/v1600
http://example.org/3gpp/pm/ProvMnS/v1600
```

Using the MnS Producer endpoint for configuration management only the objects for configuration management can be accessed. The canonical URIs of these objects are

```
http://example.org/SubNetwork=SN1
http://example.org/SubNetwork=SN1/ManagedElement=ME1
http://example.org/SubNetwork=SN1/ManagedElement=ME2
http://example.org/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1
http://example.org/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF2
```

Using the MnS Producer endpoint for performance management only the objects for performance management can be accessed.

```
http://example.org/SubNetwork=SN1/PerfMetricJob=PMJ1
http://example.org/SubNetwork=SN1/ThresholdMonitor=TM1
```

When trying to access with the MnS Producer for performance management an object pertaining to the subset of managed objects for configuration management, for example,

```
GET /3gpp/fm//SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1
Host: example.org
Accept: application/json
```

an error is raised, for example "403 Forbidden" or "404 Not Found".

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
2022-09	SA#97e	SP-220853	0053	1	A	Align examples for DNs and URIs in clause 4.2 with object class naming conventions	17.2.0
2022-09	SA#97e	SP-220853	0055	1	A	Clarify concept of NRM root	17.2.0
2022-09	SA#97e	SP-220853	0057	1	A	Clarify only leaf resources can be created	17.2.0
2022-09	SA#97e	SP-220853	0059	1	A	Clarify HTTP POST and HTTP PUT response message format	17.2.0
2022-09	SA#97e	SP-220853	0061	-	A	Correct and clarify numerous smaller issues	17.2.0
2022-09	SA#97e	SP-220850	0063	-	A	Clarify use of the JSON Patch test operation	17.2.0
2022-10	SA#97e					Correcting CR implementation error in A.1 and formatting in clause 4.2	17.2.1
2022-10	SA#97e	SP-221170	0065	1	A	Clarify usage of information models	17.3.0
2022-10	SA#97e	SP-221170	0067	2	A	Clarify format of target URIs	17.3.0
2022-10	SA#97e	SP-221170	0069	1	A	Clarify media type related aspects	17.3.0
2022-10	SA#97e	SP-221170	0071	-	A	Clarify some aspects of basic design patterns	17.3.0
2022-10	SA#97e	SP-221170	0073	-	A	Clarify construction rules for GET response message body formats	17.3.0
2022-10	SA#97e	SP-221170	0075	-	A	Clarify design patterns for patching resources	17.3.0
2022-10	SA#97e	SP-221170	0077	1	A	Correct and clarify examples in Annex A	17.3.0
2023-03	SA#99	SP-230198	0079	1	A	Clarify URI concept	17.4.0
2023-03	SA#99	SP-230198	0081	-	A	Correct media type of 3GPP JSON Patch and 3GPP JSON Merge Patch	17.4.0
2023-03	SA#99	SP-230198	0083	-	A	Align and clarify definitions for the Accept header	17.4.0
2023-03	SA#99	SP-230198	0085	1	A	Clarify an object must exist when adding members with JSON Patch	17.4.0
2023-03	SA#99	SP-230198	0087	-	A	Correct objectInstance values in examples	17.4.0
2023-03	SA#99	SP-230198	0089	-	A	Remove HTTP GET response examples not used in TS 28.532	17.4.0
2023-03	SA#99	SP-230198	0091	-	A	Add missing JSON schema fragment for the JSON Patch document	17.4.0
2023-03	SA#99	SP-230198	0093	-	A	Correct format of MnS versions in examples	17.4.0

2023-03	SA#99	SP-230198	0097	-	A	Clarify the JSON Merge Patch document is a partial resource representation	17.4.0
2023-03	SA#99	SP-230198	0098	-	A	Correct attribute value null	17.4.0
2023-03	SA#100	SP-230681	0100	1	A	Clarify usage of filter query parameter	17.5.0
2023-03	SA#100	SP-230648	0102	-	A	Correct JSON schema fragment for JSON Patch documents	17.5.0
2023-03	SA#100	SP-230681	0104	1	A	Clarify usage of the attributes container for object selection	17.5.0
2023-03	SA#100	SP-230681	0106	1	A	Clarify usage of the attributes container for complete resource updates	17.5.0
2023-09	SA#101	SP-230942	0108	-	A	Clarify URI path components need to be percent encoded	17.6.0
2023-09	SA#101	SP-230942	0110	-	A	Add missing example for large queries	17.6.0
2023-09	SA#101	SP-230942	0112	1	A	Clarify an empty result set produced by scoping and filtering is not an error	17.6.0
2023-09	SA#101	SP-230942	0114	-	A	Add missing example for data model partitioning	17.6.0
2024-03	SA#103	SP-240168	0128	-	B	Rel-18 CR 32.158 Add design pattern for error responses	18.0.0
2024-03	SA#103	SP-240168	0129	-	B	Rel-18 CR 32.158 Add design pattern for conditional data node selection	18.0.0
2024-03	SA#103	SP-240168	0135	-	B	Rel-18 CR 32.158 Replace XPath by Jex	18.0.0
2024-06	SA#104	SP-240805	0137	1	A	Rel-18 CR 32.158 Clarify usage of information models	18.1.0
2024-06	SA#104	SP-240810	0138	-	F	Fix reference to Jex in clause 6.7	18.1.0
2024-09	SA#105	SP-241171	0142	1	A	Rel-18 CR TS 32.158 Update the IETF references to the latest IETF draft	18.2.0
2025-09	SA#109	SP-251083	0148		F	Rel-18 CR 32.158 Clarify 3GPP JSON Patch	18.3.0

History

Document history		
V18.0.0	May 2024	Publication
V18.1.0	July 2024	Publication
V18.2.0	October 2024	Publication
V18.3.0	October 2025	Publication