**3GPP TSG-SA5 Meeting #129-e *S5-201384***

**Online, 24th Feb 2020 - 4th Mar 2020**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *CR-Form-v12.0* | | | | | | | | |
| **CHANGE REQUEST** | | | | | | | | |
|  | | | | | | | | |
|  |  | **CR** |  | **rev** |  | **Current version:** |  |  |
|  | | | | | | | | |
| *For* [***HE******LP***](http://www.3gpp.org/3G_Specs/CRs.htm#_blank)*on using this form: comprehensive instructions can be found at* [*http://www.3gpp.org/Change-Requests*](http://www.3gpp.org/Change-Requests)*.* | | | | | | | | |
|  | | | | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Proposed change affects:*** | UICC apps |  | ME |  | Radio Access Network | **X** | Core Network | **X** |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
| ***Title:*** | Clarify HTTP PATCH methods | | | | | | | | | |
|  |  | | | | | | | | | |
| ***Source to WG:*** | Nokia, Nokia Shanghai Bell | | | | | | | | | |
| ***Source to TSG:*** | S5 | | | | | | | | | |
|  |  | | | | | | | | | |
| ***Work item code:*** |  | | | | |  | ***Date:*** | | | -02-2020 |
|  |  | | | |  | |  | | |  |
| ***Category:*** |  |  | | | | | ***Release:*** | | | Rel- |
|  | *Use one of the following categories:* ***F*** *(correction)* ***A*** *(mirror corresponding to a change in an earlier release)* ***B*** *(addition of feature),* ***C*** *(functional modification of feature)* ***D*** *(editorial modification)*  Detailed explanations of the above categories can be found in 3GPP [TR 21.900](http://www.3gpp.org/ftp/Specs/html-info/21900.htm). | | | | | | | | *Use one of the following releases: Rel-8 (Release 8) Rel-9 (Release 9) Rel-10 (Release 10) Rel-11 (Release 11) Rel-12 (Release 12)* *Rel-13 (Release 13) Rel-14 (Release 14) Rel-15 (Release 15) Rel-16 (Release 16)* | |
|  |  | | | | | | | | | |
| ***Reason for change:*** | | The HTTP PATCH methods need to be clarified. More specifically, the IETF described JSON Patch and JSON Merge Patch do not work well for JSON arrays and manipulating multiple resources representing managed objects instances. 3GPP enhancements are introduced to overcome these shortcomings. | | | | | | | | |
|  | |  | | | | | | | | |
| ***Summary of change:*** | | 3GPP enhancements are introduced to the IETF defined JSON Patch and JSON Merge Patch formats. | | | | | | | | |
|  | |  | | | | | | | | |
| ***Consequences if not approved:*** | | It is not possible to manipulate multiple resources with a single HTTP method request making bulk configuration impossible. | | | | | | | | |
|  | |  | | | | | | | | |
| ***Clauses affected:*** | | 4,3, 6.1.1, 6,3, 6.4.2, 6.4.3, A.1, A.2.1, A.2.2, A.2.3, A.3.1, A.3.2, A.3.3, A.4, A.4.2, A.4.3, A.5, A.6.1, A.6.2, A.6.3, A.6.4 (new), A.7.1, A.7.2 | | | | | | | | |
|  | |  | | | | | | | | |
|  | | **Y** | **N** |  | | | |  | | |
| ***Other specs*** | |  | **X** | Other core specifications | | | | TS/TR ... CR ... | | |
| ***affected:*** | |  | **X** | Test specifications | | | | TS/TR ... CR ... | | |
| ***(show related CRs)*** | |  | **X** | O&M Specifications | | | | TS/TR ... CR ... | | |
|  | |  | | | | | | | | |
| ***Other comments:*** | |  | | | | | | | | |
|  | |  | | | | | | | | |
| ***This CR's revision history:*** | |  | | | | | | | | |

|  |
| --- |
| **First modification** |

## 4.3 Media types

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

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

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

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

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

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

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

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

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

|  |
| --- |
| **Next modification** |

# 6 Advanced design patterns

## 6.1 Design pattern for scoping and filtering

### 6.1.1 Introduction

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

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

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

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

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

### 6.1.2 Query parameters for scoping

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

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

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

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

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

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

|  |  |
| --- | --- |
| **Value** | **Description** |
| BASE\_ONLY | Selects only the base resource. The "scopeLevel" parameter shall be absent or ignored if present. |
| BASE\_ALL | Selects the base resource and all of its subordinate resources (incl. the leaf resources). The "scopeLevel" parameter shall be absent or ignored if present. |
| BASE\_NTH\_LEVEL | Selects all resources on the level, which is indicated by the "scopeLevel" parameter, below the base resource. The base resource is at "scopeLevel" zero. |
| BASE\_SUBTREE | Selects the base resource and all of its subordinate resources down to and including the resources on the level indicated by the "scopeLevel" parameter. The base resource is at "scopeLevel" zero. |

### 6.1.3 Query parameters for filtering

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

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

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

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

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

### 6.1.4 Construction rules for the response message body

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

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

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

## 6.2 Design pattern for attribute and attribute field selection

### 6.2.1 Introduction

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

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

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

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

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

### 6.2.2 Query parameters for attribute and attribute field selection

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

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

## 6.3 Design pattern for partially updating a resource

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

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

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

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

2) Adding a new name/value pair.

3) Removing an existing name/value pair.

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

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

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

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

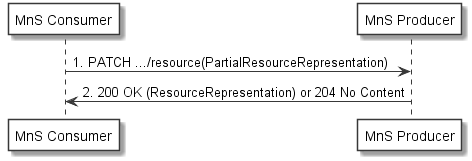


Figure 6.3-1: Flow for partially updating a resource

The procedure flow is as follows:

1) The MnS Consumer sends a HTTP PATCH request to the MnS Producer. The resource to be updated is identified with the target URI. The message body carries a JSON Patch or JSON Merge Patch document describing a set of modification instructions to be applied to the target resource.

2) The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated resource in the message body or "204 No Content" shall be returned. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

## 6.4 Design pattern for patching multiple resources

### 6.4.1 Introduction

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

### 6.4.2 3GPP JSON Merge Patch

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

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

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

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

The procedure is as follows:

1. The MnS Consumer sends a HTTP PATCH request to the MnS Producer. The message body carries a 3GPP JSON Merge Patch document describing a set of modification instructions to be applied to the identified resources.
2. The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated resources, constructed according to the hierarchical response construction method described in clause 6.1.1, in the message body or "204 No Content" shall be returned. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

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

### 6.4.3 3GPP JSON Patch

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

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

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

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

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

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

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

The procedure is as follows:

1. The MnS Consumer sends a HTTP PATCH request to the MnS Producer. The message body carries a 3GPP JSON Patch document describing a set of modification instructions to be applied to the identified resources.
2. The MnS Producer returns the HTTP PATCH response to the MnS Consumer. On success, "200 OK" together with the representation of the updated resources, constructed according to the hierarchical response construction method described in clause 6.1.1, in the message body or "204 No Content" shall be returned. On failure, the appropriate error code shall be returned. The response message body may provide additional error information.

|  |
| --- |
| **Next modification** |

Annex A (informative):  
Examples

# A.1 Example information model

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

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

The corresponding JSON schema is

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

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

# A.2 Retrieval of resources

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

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

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

The response includes the resource representation

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "id": "XYZF1",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  } |

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

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "XyzFunction": [  {  "id": "XYZF1",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  ]  } |

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

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "XyzFunction": {  "id": "XYZF1",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  } |

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

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "data": {  "XyzFunction": {  "id": "XYZF1",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  }  } |

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

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

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

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

Alternatively one might send as well

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

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

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "SubNetwork": {  "id": "SN1",  "attributes": {  "userLabel": "Berlin NW",  "plmn-id": {  "mnc": 789  }  }  }  } |

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

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

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

Alternatively, the following XPath expression can be used.

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

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

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "SubNetwork": {  "id": "SN1",  "ManagedElement": [  {  "id": "ME1",  "attributes": {  "userLabel": "Berlin NW 1",  "vendorname": "Company XY",  "location": "TV Tower"  }  },  {  "id": "ME2",  "attributes": {  "userLabel": "Berlin NW 2",  "vendorname": "Company XY",  "location": "Grunewald"  }  }  ]  }  } |

The following example returns the containment tree only

|  |
| --- |
| GET /SubNetwork=SN1?scope=BASE\_ALL&attributes= HTTP/1.1  Host: example.org  Accept: application/json |

|  |
| --- |
| HTTP/1.1 200 OK  Date: Tue, 06 Aug 2019 16:50:26 GMT  Content-Type: application/json  {  "SubNetwork": {  "id": "SN1",  "ManagedElement": [  {  "id": "ME1",  "XyzFunction": [  {  "id": "XYZF1"  },  {  "id": "XYZF2"  }  ]  },  {  "id": "ME2"  }  ]  }  } |

# A.3 Creation of resources

## A.3.1 Creation of a resource with HTTP PUT

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

|  |
| --- |
| PUT /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1  Host: example.org  Content-Type: application/json  {  "XyzFunction": [  {  "id": "XYZF1",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  ]  } |

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

|  |
| --- |
| HTTP/1.1 201 Created  Date: Tue, 06 Aug 2019 16:50:26 GMT  Location: http://example.org/ SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1  Content-Type: application/json  {  "XyzFunction": [  {  "id": "XYZF1",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  ]  } |

## A.3.2 Creation of a resource with HTTP POST

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

|  |
| --- |
| POST /SubNetwork=SN1/ManagedElement=ME1 HTTP/1.1  Host: example.org  Content-Type: application/json  {  "XyzFunction": [  {  "id": "null",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  ]  } |

|  |
| --- |
| HTTP/1.1 201 Created  Date: Tue, 06 Aug 2019 16:50:26 GMT  Location: http://example.org/ SubNetwork=SN1/ManagedElement=ME1/XyzFunction=123e4567-e89b  Content-Type: application/json  {  "XyzFunction": [  {  "id": "123e4567-e89b",  "attributes": {  "attrA": "xyz",  "attrB": 551  }  }  ]  } |

## A.3.3 Creation of a resource with JSON Patch

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

|  |
| --- |
| PATCH /SubNetwork=SN1/ManagedElement=ME1 HTTP/1.1  Host: example.org  Content-Type: application/json-patch+json  [  {  "op": "add",  "path": "",  "value": {  "id": "ME1",  "class": "ManagedElement",  "attributes": {  "userLabel": " Berlin NW 1",  "vendorname": "Company XY",  "location": "TV Tower"  }  }  }  ] |

# A.4 Deletion of resources

## A.4.1 Deletion of a resource with HTTP DELETE

The following example deletes an instance of "ManagedElement".

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

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

## A.4.2 Deletion of multiple resources with HTTP DELETE

This example deletes both "XyzFunction" resources.

|  |
| --- |
| DELETE /SubNetwork=SN1?scopeType= BASE\_NTH\_LEVEL&scopeLevel=2 HTTP/1.1  Host: example.org |

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

## A.4.3 Deletion of a resource with JSON Patch

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

|  |
| --- |
| PATCH /SubNetwork=SN1/ManagedElement=ME1 HTTP/1.1  Host: example.org  Content-Type: application/json-patch+json  [  {  "op": "remove",  "path": ""  }  ] |

# A.5 Complete update of a resource

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

|  |
| --- |
| PUT /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1  Host: example.org  Content-Type: application/merge-patch+json  {  "XyzFunction": [  {  "id": "XYZF1",  "attributes": {  "attrA": "newValue",  "attrB": 551  }  }  ]  } |

# A.6 Partial update of a resource

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

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

|  |
| --- |
| PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1  Host: example.org  Content-Type: application/merge-patch+json  {  "XyzFunction": {  "id": "XYZF1",  "attributes": {  "attrA": "def"  }  }  } |

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

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/merge-patch+json  {  "SubNetwork": {  "id": "SN1",  "attributes": {  "plmn-Id": {  "mcc": 654  }  }  }  } |

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

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

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

|  |
| --- |
| PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1  Host: example.org  Content-Type: application/3gpp-merge-patch+json  {  "XyzFunction": [  {  "id": "XYZF1",  "attributes": {  "attrA": "def"  }  }  ]  } |

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/3gpp-merge-patch+json  {  "SubNetwork": {  "id": "SN1",  "attributes": {  "plmn-Id": {  "mcc": 654  }  }  }  } |

## A.6.3 Partial update of a resource with JSON Patch

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

|  |
| --- |
| PATCH /SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1 HTTP/1.1  Host: example.org  Content-Type: application/json-patch+json  [  {  "op": "replace",  "path": "/attributes/attrA",  "value": 654  }  ] |

and

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/json-patch+json  [  {  "op": "replace",  "path": "/attributes/plmn-Id/mcc",  "value": 654  }  ] |

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

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

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/3gpp-json-patch+json  [  {  "op": "replace",  "path": "/ManagedElement=ME1/XyzFunction=XYZF1#attributes/attrA",  "value": 654  }  ] |

and

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

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

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

# A.7 Manipulating multiple resources

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

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

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/3gpp-merge-patch+json  {  "SubNetwork": {  "id": "SN1",  "attributes": {  "userLabel": "Berlin NW-1",  "plmn-id": {  "mcc": 456  }  },  "ManagedElement": [  {  "id": "ME1",  "XyzFunction": [  {  "id": "XYZF3",  "attributes": {  "attrA": "fgh",  "attrB": 555  }  }  ]  },  {  "id": "ME3",  "attributes": {  "userLabel": " Berlin NW 3",  "vendorname": "Company XY",  "location": "Spandau"  }  }  ]  }  } |

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

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/3gpp-merge-patch+json  {  "SubNetwork": {  "id": "SN1",  "ManagedElement": [  {  "id": "ME1",  "XyzFunction": [  {  "id": "XYZF2",  "attributes": null  }  ]  }  ]  }  } |

## A.7.2 Manipulating multiple resources with 3GPP JSON PATCH

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

|  |
| --- |
| PATCH /SubNetwork=SN1 HTTP/1.1  Host: example.org  Content-Type: application/3gpp-json-patch+json  [  {  "op": "replace",  "path": "/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1#/attributes/userLabel",  "value": "Berlin NW-1"  },  {  "op": "replace",  "path": "/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF1#/attributes/plmn-id/mcc",  "value": 654  },  {  "op": "add",  "path": "/SubNetwork=SN1/ManagedElement=ME1/XyzFunction=XYZF3",  "value": {  "id": "XYZF3",  "attributes": {  "attrA": "fgh",  "attrB": 555  }  }  },  {  "op": "add",  "path": "/SubNetwork=SN1/ManagedElement=ME3",  "value": {  "id": "ME3",  "attributes": {  "userLabel": " Berlin NW 3",  "vendorname": "Company XY",  "location": "Spandau"  }  }  }  ] |

|  |
| --- |
| **End of modifications** |