ETSI GS NFV-SOL 013 V5.1.1 (2024-07)

Network Functions Virtualisation (NFV) Release 5;
Protocols and Data Models;
Specification of common aspects for RESTful NFV MANO APIs

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
RGS/NFV-SOL013ed511

Keywords
API, NFV, protocol

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

Important notice

The present document can be downloaded from:
https://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at www.etsi.org/deliver.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

If you find a security vulnerability in the present document, please report it through our
Coordinated Vulnerability Disclosure Program:
https://www.etsi.org/standards/coordinated-vulnerability-disclosure

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2024.
All rights reserved.

Contents

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

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

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

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.

1 Scope

The present document specifies common aspects of RESTful protocols and data models for ETSI NFV management and orchestration (MANO) interfaces.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] Void.

[2] IETF RFC 3339: "Date and Time on the Internet: Timestamps".

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

[4] Void.

[5] IETF RFC 5246: "The Transport Layer Security (TLS) Protocol Version 1.2".

[6] IETF RFC 6585: "Additional HTTP Status Codes".

[7] IETF RFC 6749: "The OAuth 2.0 Authorization Framework".

[8] IETF RFC 6750: "The OAuth 2.0 Authorization Framework: Bearer Token Usage".

[9] IETF RFC 8259: "The JavaScript Object Notation (JSON) Data Interchange Format".

[10] Void.

[11] Void.

[12] Void.

[13] Void.

[14] IETF RFC 7617: "The 'Basic' HTTP Authentication Scheme".

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

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

[17] IETF RFC 8288: "Web Linking".

[18] Semantic Versioning 2.0.0.

[19] IETF RFC 4229: "HTTP Header Field Registrations".

[20] ETSI GS NFV-SEC 022: "Network Functions Virtualisation (NFV) Release 4; Security; Access Token Specification for API Access".

[21] ETSI TS 133 210: "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Network Domain Security (NDS); IP network layer security (3GPP TS 33.210)".

[22] IETF RFC 8446: "The Transport Layer Security (TLS) Protocol Version 1.3".

[23] IETF RFC 7515: "JSON Web Signature (JWS)".

[24] IETF RFC 9110: "HTTP Semantics".

[25] IETF RFC 8705: " OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GR NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.2] Void.

[i.3] IANA: "Hypertext Transfer Protocol (HTTP) Status Code Registry".

[i.4] ETSI NFV OpenAPI repository.

[i.5] JSON Schema Validation: "A Vocabulary for Structural Validation of JSON".

[i.6] OpenAPI™ Specification.

[i.7] IANA: "Hash Function Textual Names".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR NFV 003 [i.1] apply.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
EM	Element Manager
ETSI	European Telecommunications Standards Institute
GMT	Greenwich Mean Time
GS	Group Specification

HATEOAS	Hypermedia As The Engine Of Application State
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
MAC	Medium Access Control
MANO	MANagement and Orchestration
MIME	Multipurpose Internet Mail Extensions
NFV	Network Functions Virtualisation
NFVO	NFV Orchestrator
REST	REpresentational State Transfer
RFC	Request For Comments
TLS	Transport Layer Security
URI	Uniform Resource Identifier
VIM	Virtualised Infrastructure Manager
VNF	Virtualised Network Function
VNFM	VNF Manager

4 HTTP usage

4.1 URI structure and supported content formats

This clause specifies the URI prefix and the supported formats applicable to the RESTful NFV-MANO APIs.

All resource URIs of the APIs shall have the following prefix, except the "API versions" resource which shall follow the rules specified in clause 9.3:

{apiRoot}/{apiName}/{apiMajorVersion}/

where:

{apiRoot} indicates the scheme ("https"), the host name and optional port, and an optional sequence of path segments that together represent a prefix path.

EXAMPLE: http://orchestrator.example.com/nfv_apis/abc

{apiName} indicates the interface name in an abbreviated form. The {apiName} of each interface is defined in the clause specifying the corresponding interface.

{apiMajorVersion} indicates the current major version (see clause 9.1) of the API and is defined in the clause specifying the corresponding interface.

For HTTP requests and responses that have a body, the content format JSON (see IETF RFC 8259 [9]) shall be supported. The JSON format shall be signalled by the content type "application/json".

HTTP shall be run as an application protocol over TLS, a combination that is known as HTTPS. All APIs shall use TLS version 1.2 as defined by IETF RFC 5246 [5] or later. Versions of TLS earlier than 1.2 shall neither be supported nor used.

NOTE 1: The HTTP protocol elements mentioned in the RESTful NFV-MANO API specifications originate from the HTTP specification; HTTPS runs the HTTP protocol on top of a TLS layer. The RESTful NFV-MANO specifications therefore use the statement above to mention "HTTP request", "HTTP header", etc., without explicitly calling out whether or not these are run over TLS.

TLS implementations shall meet or exceed the security algorithm, key length and strength requirements specified in clause 6.2.3 (if TLS version 1.2 as defined by IETF RFC 5246 [5] is used) or clause 6.2.2 (if TLS version 1.3 as defined by IETF RFC 8446 [22] is used) of ETSI TS 133 210 [21] (3GPP Release 16 or later).

All resource URIs of the API shall comply with the URI syntax as defined in IETF RFC 3986 [3]. An implementation that dynamically generates resource URI parts (individual path segments, sequences of path segments that are separated by "/", query parameter values) shall ensure that these parts only use the character set that is allowed by IETF RFC 3986 [3] for these parts.

NOTE 2: This means that characters not part of this allowed set are escaped using percent-encoding as defined by IETF RFC 3986 [3].

Unless otherwise specified explicitly, all request URI parameters that are part of the path of the resource URI shall be individual path segments, i.e. shall not contain the "/" character.

NOTE 3: A request URI parameter is denoted by a string in curly brackets, e.g. {subscriptionId}.

4.2 Usage of HTTP header fields

4.2.1 Introduction

HTTP headers are components of the header section of the HTTP request and response messages. They contain the information about the server/client and metadata of the transaction. The use of HTTP header fields shall comply with the provisions defined for those header fields in the specifications referenced from tables 4.2.2-1 and 4.2.3-1. The following clauses describe more details related to selected HTTP header fields.

4.2.2 Request header fields

This clause describes the usage of selected HTTP header fields of the request messages in the RESTful NFV-MANO APIs. The HTTP header fields used in the request messages are specified in table 4.2.2-1.

Table 4.2.2-1: Header fields supported in the request message

Header field name	Reference	Example	Descriptions
Accept	IETF RFC 9110 [24]	application/json	Content-Types that are acceptable for the response. This header field shall be present if the response is expected to have a non-empty message body.
Content-Type	IETF RFC 9110 [24]	application/json	The MIME type of the body of the request. This header field shall be present if the request has a non-empty message body.
Authorization	IETF RFC 9110 [24]	Bearer mF_9.B5f-4.1JqM	The authorization token for the request. Details are specified in clause 8.3.
Range	IETF RFC 9110 [24]	1 000-2 000	Requested range of bytes from a file.
Version	IETF RFC 4229 [19]	1.2.0 or 1.2.0-impl:example.com:myProduct:4	Version of the API requested to use when responding to this request.
If-Unmodified-Since	IETF RFC 9110 [24]	Sat, 29 Oct 1994 19:43:31 GMT	Used to make the request method conditional on the selected resource representation's last modification date being earlier than or equal to the date provided in the field-value. If the condition is not met, the request fails with a "412 Precondition Failed" response.

Header field name	Reference	Example	Descriptions
If-Match	IETF RFC 9110 [24]	"xyzzy" or "xyzzy", "r2d2xxxx", "c3piozzzz" or *	Used to make the request method conditional on the recipient origin server either having at least one current representation of the target resource, when the field-value is "*", or having a current representation of the target resource that has an entity-tag matching a member of the list of entity-tags provided in the field-value. If the condition is not met, the request fails with a "412 Precondition Failed" response.

4.2.3 Response header fields

This clause describes the usage of selected HTTP header fields of the response messages in the RESTful NFV-MANO APIs. The HTTP header fields used in the response messages are specified in table 4.2.3-1.

Table 4.2.3-1: Header fields supported in the response message

Header field name	Reference	Example	Descriptions
Content-Type	IETF RFC 9110 [24]	application/json	The MIME type of the body of the response. This header field shall be present if the response has a non-empty message body.
Location	IETF RFC 9110 [24]	http://www.example.com/vnflcm/v1/vnf_instances/123	Used in redirection, or when a new resource has been created. This header field shall be present if the response status code is 201 or 3xx. In the RESTful NFV-MANO APIs this header field is also used if the response status code is 202 and a new resource was created.
WWW-Authenticate	IETF RFC 9110 [24]	Bearer realm="example"	Challenge if the corresponding HTTP request has not provided authorization, or error details if the corresponding HTTP request has provided an invalid authorization token.
Accept-Ranges	IETF RFC 9110 [24]	bytes	Used by the server to signal whether or not it supports ranges for certain resources.
Content-Range	IETF RFC 9110 [24]	bytes 21 010 - 47 021/47 022	Signals the byte range that is contained in the response, and the total length of the file.
Retry-After	IETF RFC 9110 [24]	Fri, 31 Dec 1999 23:59:59 GMT or 120	Used to indicate how long the user agent ought to wait before making a follow-up request. It can be used with 503 responses. The value of this field can be an HTTP-date or a number of seconds to delay after the response is received.
Link	IETF RFC 8288 [17]	<http://example.com/resources?nextpage_opaque_marker=abc123>; rel="next"	Reference to other resources. Used for paging in the present document, see clause 5.4.2.1.
Version	IETF RFC 4229 [19]	1.2.0 or 1.2.0-impl:example.com:myProduct:4	Version of the API requested to use when responding to this request.

Header field name	Reference	Example	Descriptions
ETag	IETF RFC 9110 [24]	"33a64df551425fcc55e4d42a148795d9f25f89d4" or W/"0815"	Used to provide the current entity-tag for the selected resource representation. It can be sent in "200 OK", "201 Created" and "204 No Content" responses.
Last-Modified	IETF RFC 9110 [24]	Tue, 15 Nov 1994 12:45:26 GMT	Used to provide a timestamp indicating the date and time at which the server believes the selected resource representation was last modified. It can be sent in "200 OK", "201 Created" and "204 No Content" responses.

5 Result set control

5.1 Introduction

This clause specifies procedures that allow to control the size of the result set of GET requests with respect to the number of entries in a response list (using attribute-based filtering) or with respect to the number of attributes returned in a response (using attribute selection).

5.2 Attribute-based filtering

5.2.1 Overview and example (informative)

Attribute-based filtering allows to reduce the number of objects returned by a query operation. Typically, attribute-based filtering is applied to a GET request that reads a resource which represents a list of objects (e.g. child resources). Only those objects that match the filter are returned as part of the resource representation in the message content of the GET response.

Attribute-based filtering can test a simple (scalar) attribute of the resource representation against a constant value, for instance for equality, inequality, greater or smaller than, etc. Attribute-based filtering is requested by adding a set of URI query parameters, the "attribute-based filtering parameters" or "filter" for short, to a resource URI.

The following example illustrates the principle. Assume a resource "container" with the following objects:

EXAMPLE 1: Objects:

```
obj1: {"id":123, "weight":100, "parts":[{"id":1, "color":"red"}, {"id":2, "color":"green"}]}
obj2: {"id":456, "weight":500, "parts":[{"id":3, "color":"green"}, {"id":4, "color":"blue"}]}
```

A GET request on the "container" resource would deliver the following response:

EXAMPLE 2: Unfiltered GET:

```
Request:
GET …/container
Response:
[
    {"id":123, "weight":100, "parts":[{"id":1, "color":"red"}, {"id":2, "color":"green"}]},
    {"id":456, "weight":500, "parts":[{"id":3, "color":"green"}, {"id":4, "color":"blue"}]}
]
```

A GET request with a filter on the "container" resource would deliver the following response:

EXAMPLE 3: GET with filter:

```
Request:
GET …/container?filter=(eq,weight,100)
Response:
[
```

```
    {"id":123, "weight":100, "parts":[{"id":1, "color":"red"}, {"id":2, "color":"green"}]}
]
```

For hierarchically-structured data, filters can also be applied to attributes deeper in the hierarchy. In case of arrays, a filter matches if any of the elements of the array matches. In other words, when applying the filter "(eq,parts/color,green)" to the objects in example 1, the filter matches obj1 when evaluating the second entry in the "parts" array of obj1 and matches obj2 already when evaluating the first entry in the "parts" array of obj2. As the result, both obj1 and obj2 match the filter.

If a filter contains multiple sub-parts that only differ in the leaf attribute (i.e. they share the same attribute prefix), they are evaluated together per array entry when traversing an array. As an example, the two expressions in the filter "(eq,parts/color,green);(eq,parts/id,3)" would be evaluated together for each entry in the array "parts". As the result, obj2 matches the filter.

5.2.2 Specification

An attribute-based filter shall be represented by a URI query parameter named "filter". The value of this parameter shall consist of one or more strings formatted according to "simpleFilterExpr", concatenated using the ";" character:

```
simpleFilterExprOne     := <opOne>","<attrName>["/"<attrName>]*","<value>
simpleFilterExprMulti   := <opMulti>","<attrName>["/"<attrName>]*","<value>[","<value>]*
simpleFilterExpr        := "("<simpleFilterExprOne>")" | "("<simpleFilterExprMulti>")"
filterExpr              := <simpleFilterExpr>[";"<simpleFilterExpr>]*
filter                  := "filter"=<filterExpr>
opOne                   := "eq" | "neq" | "gt" | "lt" | "gte" | "lte"
opMulti                 := "in" | "nin" | "cont" | "ncont"
attrName                := string
value                   := string
```

where:

```
*   zero or more occurrences
[]  grouping of expressions to be used with *
""  quotation marks for marking string constants
<>  name separator
|   separator of alternatives
```

"AttrName" is the name of one attribute in the data type that defines the representation of the resource. The slash ("/") character in "simpleFilterExprOne" and " simpleFilterExprMulti" allows concatenation of <attrName> entries to filter by attributes deeper in the hierarchy of a structured document. The special attribute name "@key" refers to the key of a map.

EXAMPLE: Referencing the key of a map:

```
GET …/resource?filter=(eq,mymap/@key,abc123)
```

The elements "opOne" and "opMulti" stand for the comparison operators (accepting one comparison value or a list of such values). If the expression has concatenated <attrName> entries, it means that the operator is applied to the attribute addressed by the last <attrName> entry included in the concatenation. All simple filter expressions are combined by the "AND" logical operator, denoted by ";".

In a concatenation of <attrName> entries in a <simpleFilterExprOne> or <simpleFilterExprMulti>, the rightmost <attrName> entry is called "leaf attribute". The concatenation of all "attrName" entries except the leaf attribute is called the "attribute prefix". If an attribute referenced in an expression is an array, an object that contains a corresponding array shall be considered to match the expression if any of the elements in the array matches all expressions that have the same attribute prefix.

The leaf attribute of a <simpleFilterExprOne> or <simpleFilterExprMulti> shall not be structured but shall be of a simple (scalar) type such as String, Number, Boolean or DateTime, or shall be an array of simple (scalar) values. Attempting to apply a filter with a structured leaf attribute shall be rejected with "400 Bad request". A <filterExpr> shall not contain any invalid <simpleFilterExpr> entry.

The operators listed in table 5.2.2-1 shall be supported.

Table 5.2.2-1: Operators for attribute-based filtering

Operator with parameters	Meaning
eq,<attrName>,<value>	Attribute **equal** to <value>
neq,<attrName>,<value>	Attribute **not equal** to <value>
in,<attrName>,<value>[,<value>]*	Attribute equal to one of the values in the list ("**in set**" relationship)
nin,<attrName>,<value>[,<value>]*	Attribute not equal to any of the values in the list ("**not in set**" relationship)
gt,<attrName>,<value>	Attribute **greater than** <value>
gte,<attrName>,<value>	Attribute **greater than or equal** to <value>
lt,<attrName>,<value>	Attribute **less than** <value>
lte,<attrName>,<value>	Attribute **less than or equal** to <value>
cont,<attrName>,<value>[,<value>]*	String attribute **contains** (at least) one of the values in the list
ncont,<attrName>,<value>[,<value>]*	String attribute **does not contain** any of the values in the list

Table 5.2.2-2: Applicability of the operators to data types

Operator	String	Number	DateTime	Enumeration	Boolean
eq	x	x	-	x	x
neq	x	x	-	x	x
in	x	x	-	x	-
nin	x	x	-	x	-
gt	x	x	x	-	-
gte	x	x	x	-	-
lt	x	x	x	-	-
lte	x	x	x	-	-
cont	x	-	-	-	-
ncont	x	-	-	-	-

Table 5.2.2-2 defines which operators are applicable for which data types. All combinations marked with a "x" shall be supported.

All objects that match the filter shall be returned as response to a GET request that contains a filter.

A <value> entry shall contain a scalar value of type Number, String, Boolean, Enum or DateTime. The content of a <value> entry shall be formatted the same way as the representation of the related attribute in the resource representation: The syntax of DateTime <value> entries shall follow the "date-time" production of IETF RFC 3339 [2]. The syntax of Boolean and Number <value> entries shall follow IETF RFC 8259 [9].

A <value> entry of type String shall be enclosed in single quotes (') if it contains any of the characters ")", "'" or ",", and may be enclosed in single quotes otherwise. Any single quote (') character contained in a <value> entry shall be represented as a sequence of two single quote characters.

The "/" and "~" characters in <attrName> shall be escaped according to the rules defined in section 3 of IETF RFC 6901 [16]. If the "," character appears in <attrName> it shall be escaped by replacing it with "~a". If the "@" character appears in <attrName> other than in the keyword "@key", it shall be escaped by replacing it with "~b".

In the resulting <filterExpr>, percent-encoding as defined in IETF RFC 3986 [3] shall be applied to the characters that are not allowed in a URI query part according to Appendix A of IETF RFC 3986 [3], and to the ampersand "&" character.

NOTE: In addition to the statement on percent-encoding above, it is reminded that the percent "%" character is always percent-encoded when used in parts of a URI, according to IETF RFC 3986 [3].

Attribute-based filters are supported for certain resources. Details are defined in the clauses specifying the actual resources.

5.3 Attribute selectors

5.3.1 Overview and example (informative)

Certain resource representations can become quite big, in particular, if the resource is a container for multiple sub-resources, or if the resource representation itself contains a deeply-nested structure. In these cases, it can be desired to reduce the amount of data exchanged over the interface and processed by the API consumer application. On the other hand, it can also be desirable that a "drill-deep" for selected parts of the omitted data can be initiated quickly.

An attribute selector allows the API consumer to choose which attributes it wants to be contained in the response. Only attributes that are not required to be present, i.e. those with a lower bound of zero on their cardinality (e.g. 0..1, 0..N) and that are not conditionally mandatory, are allowed to be omitted as part of the selection process. Attributes can be marked for inclusion or exclusion.

If an attribute is omitted, a link to a resource may be added where the information of that attribute can be fetched. Such approach is known as HATEOAS which is a common pattern in REST, and enables drilling down on selected issues without having to repeat a request that may create a potentially big response.

5.3.2 Specification

5.3.2.1 GET request

The URI query parameters for attribute selection are defined in table 5.3.2.1-1.

In the provisions below, "complex attributes" are assumed to be those attributes that are structured or that are arrays.

Table 5.3.2.1-1: Attribute selector parameters

Parameter	Definition
all_fields	This URI query parameter requests that all complex attributes are included in the response, including those suppressed by exclude_default. It is inverse to the "exclude_default" parameter. The API producer shall support this parameter for certain resources. Details are defined in the clauses specifying the actual resources.
fields	This URI query parameter requests that only the listed complex attributes are included in the response. The parameter shall be formatted as a list of attribute names. An attribute name shall either be the name of an attribute, or a path consisting of the names of multiple attributes with parent-child relationship, separated by "/". Attribute names in the list shall be separated by comma (","). Valid attribute names for a particular GET request are the names of all complex attributes in the expected response that have a lower cardinality bound of 0 and that are not conditionally mandatory. The API producer should support this parameter for certain resources. Details are defined in the clauses specifying the actual resources.
exclude_fields	This URI query parameter requests that the listed complex attributes are excluded from the response. For the format, eligible attributes and support by the API producer, the provisions defined for the "fields" parameter shall apply.
exclude_default	Presence of this URI query parameter requests that a default set of complex attributes shall be excluded from the response. The default set is defined per resource in the applicable RESTful NFV-MANO API specification. Not every resource will necessarily have such a default set. Only complex attributes with a lower cardinality bound of zero that are not conditionally mandatory can be included in the set. The API producer shall support this parameter for certain resources. Details are defined in the clauses in the applicable RESTful NFV-MANO API specification defining the actual resources. This parameter is a flag, i.e. it has no value. If a resource supports attribute selectors and none of the attribute selector parameters is specified in a GET request, the "exclude_default" parameter shall be assumed as the default.

The "/" and "~" characters in attribute names in an attribute selector shall be escaped according to the rules defined in section 3 of IETF RFC 6901 [16]. The "," character in attribute names in an attribute selector shall be escaped by replacing it with "~a". Further, percent-encoding as defined in IETF RFC 3986 [3] shall be applied to the characters that are not allowed in a URI query part according to Appendix A of IETF RFC 3986 [3], and to the ampersand "&" character.

5.3.2.2 GET response

Table 5.3.2.2-1 defines the valid parameter combinations in a GET request and their effect on the GET response.

Table 5.3.2.2-1: Valid combinations of attribute selector parameters

Parameter combination	The GET response shall include…
(none)	… same as "exclude_default".
all_fields	… all attributes.
fields=<list>	… all attributes except all complex attributes with minimum cardinality of zero that are not conditionally mandatory, and that are not provided in <list>.
exclude_fields=<list>	… all attributes except those complex attributes with a minimum cardinality of zero that are not conditionally mandatory, and that are provided in <list>.
exclude_default	… all attributes except those complex attributes with a minimum cardinality of zero that are not conditionally mandatory, and that are part of the "default exclude set" defined in the applicable RESTful NFV-MANO API specification for the particular resource.
exclude_default and fields=<list>	… all attributes except those complex attributes with a minimum cardinality of zero that are not conditionally mandatory and that are part of the "default exclude set" defined in the applicable RESTful NFV-MANO API specification for the particular resource, but that are not part of <list>.

If complex attributes were omitted in a GET response, the response may contain a number of links that allow to obtain directly the content of the omitted attributes. Such links shall be embedded into a structure named "_links" at the same level as the omitted attribute. That structure shall contain one entry for each link, named as the omitted attribute, and containing an "href" attribute that contains the URI of a resource that can be read with GET to obtain the content of the omitted attribute. A link shall not be present if the attribute is not present in the underlying resource representation. The resource URI structure of such links is not standardized but may be chosen by the API producer implementation. Performing a GET request on such a link shall return a representation that contains the content of the omitted attribute.

EXAMPLE:

```
"_links" : {
      "vnfcs" : {"href" : ".../vnflcm/v1/vnf_instances/1234/vnfcs"},
      "extVirtualLinks" : {"href" : ".../vnflcm/v1/_dynamic/7d6bef4e-d86b-4abc-97ed-
   9dc9b951f206"}
}
```

5.4 Handling of large query results

5.4.1 Overview

If the response to a query to a container resource (i.e. a resource that contains child resources whose representations will be returned when responding to a GET request) will become so large that the response will adversely affect the performance of the server, the server either rejects the request with a 400 Bad Request response, or the server provides a paged response, i.e. it returns only a subset of the query result in the response, and also provides information how to obtain the remainder of the query result.

When returning a paged response, depending on the underlying storage organization, it might be problematic for the server to determine the actual size of the result; however, it is usually possible to determine whether there will be additional results returned when knowing, for the last entry in the returned page, the position in the overall query result or some other property that has ordering semantics. For example, the time of creation of a resource has such an ordering property. When using such an (implementation-specific) property, the API producer can correctly handle deletions of child resources that happen between sending the first page of the query result, and sending the next page. It cannot be guaranteed that child resources inserted between returning subsequent pages can be considered in the query result, however, it shall be guaranteed that this does not lead to skipping of entries that have existed prior to insertion.

At minimum, a paged response needs to contain information telling the API consumer that the response is paged, and how to obtain the next page of information. For that purpose, a link to obtain the next page is returned in an HTTP header, containing a parameter that is opaque to the API consumer, but that allows the API producer to determine the start of the next page.

NOTE: In the present document, this functionality is designed for overload protection only. Additional functionality, such as configuring the page size by the API consumer, determining the size of the overall query result or the number of pages, and determining the previous page, is left outside the scope of the present document.

5.4.2 Specification

5.4.2.1 Alternatives

For each container resource (i.e. a resource that contains child resources whose representations will be returned when responding to a GET request), the API producer shall support one of the following two behaviours specified below to handle the case that a response to a query (GET request) will become so large that the response will adversely affect performance:

1) Return an error response, as defined in clause 5.4.2.2.

2) Return the result in a paged manner, as defined in clause 5.4.2.3.

5.4.2.2 Error response

In this alternative, the server shall reject the request with a 400 Bad Request response, shall include the "ProblemDetails" message content, and shall provide in the "detail" attribute more information about the error.

This error code indicates to the API consumer that with the given attribute-based filtering query (or absence thereof), the response would have been so big that performance is adversely affected. The client can obtain a query result by specifying a (more restrictive) attribute-based filtering query (see clause 5.2).

5.4.2.3 Paged response

In this alternative, the API producer shall provide a response that contains a first page (subset) of the results to the query, and shall include a LINK HTTP header (see IETF RFC 8288 [17]) with the "rel" attribute set to "next", which communicates a URI that allows to obtain the next page of results to the original query.

The API consumer can send a GET request to the URI communicated in the LINK header to obtain the next page of results. The response which returns that next page shall contain the LINK header to point to the next page, as specified above, unless there are no further pages available in which case the LINK header shall be omitted.

To allow the API producer to determine the start of the next page, the LINK header shall contain the URI query parameter "nextpage_opaque_marker" whose value is chosen by the API producer. This parameter has no meaning for the API consumer, but is echoed back by the API consumer to the API producer when requesting the next page. The URI in the link header may include further parameters, such as those passed in the original request.

The size of each page may be chosen by the API provider, and may vary from page to page. The maximum page size is determined by means outside the scope of the present document.

The response need not contain entries that correspond to child resources which were created after the original query was issued.

6 Error reporting

6.1 Introduction

In RESTful interfaces, application errors are mapped to HTTP errors. Since HTTP error information is generally not enough to discover the root cause of the error, additional application specific error information is typically delivered. The following clauses define such a mechanism to be used by the RESTful NFV-MANO APIs.

6.2 General mechanism

When an error occurs that prevents the API producer from successfully fulfilling the request, the HTTP response shall include in the response a status code in the range 400..499 (client error) or 500..599 (server error) as defined by the HTTP specification (see IETF RFC 9110 [24], as well as by IETF RFC 6585 [6]). In addition, the response body should contain a JSON representation of a "ProblemDetails" data structure according to IETF RFC 7807 [15] that provides additional details of the error. In that case, as defined by IETF RFC 7807 [15], the "Content-Type" HTTP header shall be set to "application/problem+json".

6.3 Type: ProblemDetails

The definition of the general "ProblemDetails" data structure from IETF RFC 7807 [15] is reproduced in table 6.3-1. Compared to the general framework defined in IETF RFC 7807 [15], the "status" and "detail" attributes are mandated to be included, to ensure that the response contains additional textual information about an error. IETF RFC 7807 [15] foresees extensibility of the "ProblemDetails" type. It is possible that particular RESTful NFV-MANO API, or particular implementations, specify extensions to define additional attributes that provide more information about the error.

The description column only provides some explanation of the meaning to facilitate understanding of the design. For a full description, see IETF RFC 7807 [15].

Table 6.3-1: Definition of the ProblemDetails data type

Attribute name	Data type	Cardinality	Description
type	Uri	0..1	A URI reference according to IETF RFC 3986 [3] that identifies the problem type. It is encouraged that the URI provides human-readable documentation for the problem (e.g. using HTML) when dereferenced. When this member is not present, its value is assumed to be "about:blank".
title	String	0..1	A short, human-readable summary of the problem type. It should not change from occurrence to occurrence of the problem, except for purposes of localization. If type is given and other than "about:blank", this attribute shall also be provided.
status	Integer	1	The HTTP status code for this occurrence of the problem.
detail	String	1	A human-readable explanation specific to this occurrence of the problem.
instance	Uri	0..1	A URI reference that identifies the specific occurrence of the problem. It may yield further information if dereferenced.
(additional attributes)	Not specified.	0..N	Any number of additional attributes, as defined in a specification or by an implementation.
NOTE: It is expected that the minimum set of information returned in ProblemDetails consists of "status" and "detail". For the definition of specific "type" values as well as extension attributes by implementations, guidance can be found in IETF RFC 7807 [15].			

6.4 Common error situations

The following common error situations are applicable on all REST resources and related HTTP methods defined in the RESTful NFV-MANO API specifications, and shall be handled as defined in the present clause. The full definition of each error code can be obtained from the referenced specification.

In general, error response codes used for application errors should be mapped to the most similar HTTP error status code. If no such code is applicable, one of the codes 400 (Bad Request, for client errors) or 500 (Internal Server Error, for server errors) should be used.

Implementations may use additional error response codes on top of the ones listed in this clause, as long as they are valid HTTP response codes; and should include a ProblemDetails structure in the message content as defined in clause 6.3. A list of all valid HTTP response codes and their specification documents can be obtained from the HTTP status code registry [i.3].

NOTE 1: The error handling defined in this clause only applies to REST resources defined in the RESTful NFV-MANO API specifications. For the token endpoint defined in IETF RFC 6749 [7] and re-used in the present document as defined in clause 8.3, the error handling provisions are defined in clause 8.3.

400 Bad Request: If the request is malformed or syntactically incorrect (e.g. if the request URI contains incorrect query parameters or the message content contains a syntactically incorrect data structure), the API producer shall respond with this response code. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure shall be provided, and should include in the "detail" attribute more information about the source of the problem.

400 Bad Request: If the response to a GET request which queries a container resource would be so big that the performance of the API producer is adversely affected, and the API producer does not support paging for the affected resource, it shall respond with this response code. Clause 5.4.2.2 specifies provisions for the "ProblemDetails" structure provided in the response body.

400 Bad Request: If there is an application error related to the client's input that cannot be easily mapped to any other HTTP response code ("catch all error"), the API producer shall respond with this response code. The "ProblemDetails" structure shall be provided, and shall include in the "detail" attribute more information about the source of the problem.

NOTE 2: It is by design to represent these application error situations with the same HTTP error response code 400.

400 Bad Request: If the request contains a malformed access token, the API producer should respond with this response. The details of the error shall be returned in the WWW-Authenticate HTTP header, as defined in IETF RFC 6750 [8]. The ProblemDetails structure may be provided.

NOTE 3: The use of this HTTP error response code described above is applicable to the use of the OAuth 2.0 for the authorization of API requests and notifications, as defined in clauses 8.3.3 and 8.3.4.

401 Unauthorized: If the request contains no access token even though one is required, or if the request contains an authorization token that is invalid (e.g. expired or revoked), the API producer should respond with this response. The details of the error shall be returned in the WWW-Authenticate HTTP header, as defined in IETF RFC 6750 [8] and IETF RFC 9110 [24]. The ProblemDetails structure may be provided.

403 Forbidden: If the API consumer is not allowed to perform a particular request to a particular resource, the API producer shall respond with this response code. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure shall be provided. It should include in the "detail" attribute information about the source of the problem, and may indicate how to solve it.

404 Not Found: If the API producer did not find a current representation for the resource addressed by the URI passed in the request, or is not willing to disclose that one exists, it shall respond with this response code. A typical reason for this error can e.g. be that resource URI variables were set wrongly. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure may be provided, including in the "detail" attribute information about the source of the problem, e.g. a wrong resource URI variable.

NOTE 4: This response code is not appropriate in case the resource addressed by the URI is a container resource which is designed to contain child resources, but does not contain any child resource at the time the request is received. For a GET request to an existing empty container resource, a typical response contains a 200 OK response code and a message content with an empty array.

405 Method Not Allowed: If a particular HTTP method is not supported for a particular resource, the API producer shall respond with this response code. The "ProblemDetails" structure may be provided.

406 Not Acceptable: If the "Accept" HTTP header does not contain at least one name of a content type that is acceptable to the API producer, the API producer shall respond with this response code. The "ProblemDetails" structure may be provided.

413 Content Too Large: If the message content of a request is larger than the amount of data the API producer is willing or able to process, it shall respond with this response code, following the provisions in IETF RFC 9110 [24] for the use of the "Retry-After" HTTP header and for closing the connection. The "ProblemDetails" structure may be provided.

414 URI Too Long: If the request URI of a request is longer than the API producer is willing or able to process, it shall respond with this response code. This condition can e.g. be caused by passing long queries in the request URI of a GET request. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure may be provided.

422 Unprocessable Content: If the content type of the message content is supported and the message content of a request contains syntactically correct data (e.g. well-formed JSON) but the data cannot be processed (e.g. because it fails validation against a schema), the API producer shall respond with this response code. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure shall be provided, and should include in the "detail" attribute more information about the source of the problem.

NOTE 5: This error response code is only applicable for methods that have a request body.

429 Too Many Requests: If the API consumer has sent too many requests in a defined period of time and the API producer is able to detect that condition ("rate limiting"), the API producer shall respond with this response code, following the provisions in IETF RFC 6585 [6] for the use of the "Retry-After" HTTP header. The "ProblemDetails" structure shall be provided, and shall include in the "detail" attribute more information about the source of the problem.

NOTE 6: The period of time and allowed number of requests are configured within the API producer by means outside the scope of the present document.

500 Internal Server Error: If the Server is unable to process the request, and retrying the same request later might eventually succeed, the server shall respond with this response code. Further, if there is an application error not related to the client's input that cannot be easily mapped to any other HTTP response code ("catch all error"), the API producer shall respond with this response code. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure shall be provided, and shall include in the "detail" attribute more information about the source of the problem.

503 Service Unavailable: If the API producer encounters an internal overload situation of itself or of a system it relies on, it should respond with this response code, following the provisions in IETF RFC 9110 [24] for the use of the "Retry-After" HTTP header and for the alternative to refuse the connection. The "ProblemDetails" structure may be provided.

504 Gateway Timeout: If the API producer encounters a timeout while waiting for a response from an upstream server (i.e. a server that the API producer communicates with when fulfilling a request), it should respond with this response code. More details are defined in IETF RFC 9110 [24]. The "ProblemDetails" structure may be provided.

7 Common data types

7.1 Structured data types

7.1.1 Introduction

This clause defines data structures that are referenced from data structures in multiple interfaces.

7.1.2 Type: Object

An object contains structured data, and shall comply with the provisions of clause 4 of IETF RFC 8259 [9].

7.1.3 Type: Link

This type represents a link to a resource using an absolute URI. It shall comply with the provisions defined in table 7.1.3-1.

Table 7.1.3-1: Definition of the Link data type

Attribute name	Data type	Cardinality	Description
href	Uri	1	URI of another resource referenced from a resource. Shall be an absolute URI (i.e. a URI that contains {apiRoot}).

7.1.4 Type: NotificationLink

This type represents a link to a resource in a notification, using an absolute or relative URI. It shall comply with the provisions defined in table 7.1.4-1.

Table 7.1.4-1: Definition of the NotificationLink data type

Attribute name	Data type	Cardinality	Description
href	Uri	1	URI of a resource referenced from a notification. Should be an absolute URI (i.e. a URI that contains {apiRoot}), however, may be a relative URI (i.e. a URI where the {apiRoot} part is omitted) if the {apiRoot} information is not available.

7.1.5 Type: KeyValuePairs

This type represents a list of key-value pairs. The order of the pairs in the list is not significant. In JSON, a set of key-value pairs is represented as an object. It shall comply with the provisions defined in clause 4 of IETF RFC 8259 [9]. In the following example, a list of key-value pairs with four keys ("aString", "aNumber", "anArray" and "anObject") is provided to illustrate that the values associated with different keys can be of different type.

EXAMPLE:

```
{
   "aString" : "ETSI NFV SOL",
   "aNumber" : 0.03,
   "anArray" : [1,2,3],
   "anObject" : {"organization" : "ETSI", "isg" : "NFV", workingGroup" : "SOL"}
}
```

7.1.6 Type: ApiVersionInformation

This type represents API version information. It shall comply with the provisions defined in table 7.1.6-1.

Table 7.1.6-1: ApiVersionInformation data type

Attribute name	Data type	Cardinality	Description
uriPrefix	String	1	Specifies the prefix of the resource URI of the "API versions" resource represented by this data structure. Depending on the resource URI, it shall be in one of the two following forms: {apiRoot}/{apiName}/{apiMajorVersion}/ or {apiRoot}/{apiName}/.
apiVersions	Structure (inlined)	1..N	Version(s) supported for the API signalled by the uriPrefix attribute.
>version	String	1	Identifies a supported version. The value of the version attribute shall be a version identifier as specified in clause 9.1.
>isDeprecated	Boolean	0..1	If such information is available, this attribute indicates whether use of the version signalled by the version attribute is deprecated (true) or not (false). See note.
>retirementDate	DateTime	0..1	The date and time after which the API version will no longer be supported. This attribute may be included if the value of the isDeprecated attribute is set to true and shall be absent otherwise.
NOTE: A deprecated version is still supported by the API producer but is recommended not to be used any longer. When a version is no longer supported, it does not appear in the response body.			

7.1.7 Type: Checksum

This type represents the checksum of a package or an artifact file. It shall comply with the provisions defined in table 7.1.7-1.

Table 7.1.7-1: Definition of the Checksum data type

Attribute name	Data type	Cardinality	Description
algorithm	String	1	Name of the algorithm used to generate the checksum. The name matching when processing the attribute value shall be case-insensitive, for example, "SHA-256" or "sha-256", "SHA-512" or "sha-512". See note.
hash	String	1	The hexadecimal value of the checksum.
NOTE: The IANA registry [i.7] provides the list of available hash function textual names. The provisions about which algorithms to use are expected to be defined by the specifications referring to the present data type.			

7.2 Simple data types and enumerations

7.2.1 Introduction

This clause defines simple data types and enumerations that can be referenced from data structures defined in multiple interfaces.

7.2.2 Simple data types

Table 7.2.2-1 lists the simple data types that are referenced from multiple interfaces.

Table 7.2.2-1: Simple data types

Type name	Description
Identifier	An identifier with the intention of being globally unique. Representation: string of variable length. See note 1.
DateTime	A date-time stamp. Representation: String formatted as defined by the date-time production in IETF RFC 3339 [2].
Uri	A string formatted according to IETF RFC 3986 [3].
Boolean	A data type having two values (true and false).
MacAddress	A MAC address. Representation: string that consists of groups of two hexadecimal digits, separated by hyphens or colons.

Type name	Description
IpAddress	An IPV4 or IPV6 address. Representation: In case of an IPV4 address, string that consists of four decimal integers separated by dots, each integer ranging from 0 to 255. In case of an IPV6 address, string that consists of groups of zero to four hexadecimal digits, separated by colons.
Version	A version. Representation: string of variable length.
String	A string as defined in IETF RFC 8259 [9].
Number	A number as defined in IETF RFC 8259 [9].
Integer	An integer, i.e. a number that cannot have a fractional component. See note 2.
UnsignedInt	An unsigned integer, i.e. an integer that cannot assume negative values. See note 2.
NOTE 1: Individual API specifications are assumed to define types for additional identifiers with dedicated scope (e.g. identifiers scoped by the VIM). NOTE 2: In the JSON instance data model, only the concept of a "number" is used to represent numerical data. Numbers in JSON can be integral, i.e. have no fractional part, or can include a fractional part. The additional numeric types defined in the present document represent constraints on the general "number" type present in JSON instances which can be enforced e.g. during parsing when processing the JSON instance or expressed as constraints in modelling languages such as JSON Schema [i.5] or OpenAPI [i.6].	

7.2.3 Enumerations

Void.

8 Authorization of API requests and notifications

8.1 Introduction

The RESTful NFV-MANO APIs are only allowed to be accessed following successful authorization and authentication. The authorization and authentication mechanisms are based on OAuth 2.0 as specified in clause 8.3.3.

The following terms (set in italics below) are used as defined by IETF RFC 6749 [7]:

- *client;*
- *resource server;*
- *authorization server;*
- *token endpoint;*
- *access token.*

The description below is based on the "client credentials" grant type as defined by IETF RFC 6749 [7].

For API calls, the producer functional block of an API in NFV terms corresponds to the "*resource server*", and the consumer functional block of an API corresponds to the "*client*" as defined by IETF RFC 6749 [7]. For sending a notification, these roles are reversed: the producer (notification sender) corresponds to the "*client*" and the consumer (notification receiver) corresponds to the "*resource server*".

Before invoking an HTTP method on a REST resource provided by a *resource server*, a functional block (referred to as "*client*" from now on) first obtains authorization from another functional block fulfilling the role of the "*authorization server*". The present document makes no assumption about which functional block in the architecture plays the role of the *authorization server*. It is however assumed that the address of the *token endpoint* exposed by the *authorization server* and further specified in the clauses below is provisioned to the *client* together with additional authorization-related configuration parameters, such as valid client credentials. The *client* requests an *access token* from the *token endpoint*. As part of setting up the TLS tunnel for the *access token request*, the *client* and *authorization server* perform mutual authentication based on X.509 certificates. As part of the *access token request*, the client presents its client identifier.

In addition, as an alternative to using mutual authentication based on X.509 certificates, client password may be used in the access token request toward the authorization server (as defined by IETF RFC 6749 [7]), only to support legacy implementations (version 3.4.1 or earlier version of the present document). Since allowing indiscriminate, client-choice use of client identifier/password authentication has the potential to open the network to bid-down attacks from malicious clients, the authorization server shall only allow this option for known legacy OAuth clients. The choice of whether to allow tokens obtained using this legacy mechanism for a specific API end point shall be agreed between the API producer and the authorization server, by means outside the scope of the present document. As it is insecure, this option shall neither be used nor supported except for interworking with legacy.

NOTE: Support for legacy interworking using client password will be dropped in future versions of the present document.

The *authorization server* responds with an *access token* which the *client* will present to the *resource server* with every HTTP method invocation. An *access token* represents a particular access right (defining the particular set of protected resources to access in a particular manner) with a defined duration. The token is opaque to the *client*, and can typically be used by the *authorization server* and the *resource server* as an identifier to retrieve authorization information, such as information that identifies the client, its role and access rights. An *access token* expires after a certain time, or can be revoked. If that happens, the *client* can try to obtain a new *access token* from the *authorization server*.

In order to ensure that no third party can eavesdrop on sensitive information such as client credentials or access tokens, TLS is used to protect the transport of HTTP messages. If mutual authentication using TLS protocol is used, then the producer/server is authenticated to the consumer/client, but also the consumer/client is authenticated by the producer/server at the same time. To facilitate this mutual authentication, the server shall request a client certificate. This can be done using a client certificate in TLS as described in IETF RFC 5246 [5] (in case of TLS1.2) or IETF RFC 8446 [22] (in case of TLS1.3).

8.2 Flows (informative)

8.2.1 General

This clause outlines the methods for authentication and authorization. Clause 8.2.2 presents an approach for authorizing API requests using OAuth 2.0 access tokens. Clause 8.2.5 outlines a method to authorize notifications using OAuth2.0.

8.2.2 Authorization of API requests using OAuth 2.0 access tokens

The flow below illustrates the authorization of API requests that the API consumer sends to the API producer. For authorization of API requests, using OAuth 2.0 certificate-bound access tokens is with higher security than using OAuth 2.0 access tokens. The present document recommends authorization of API requests using OAuth 2.0 certificate-bound access tokens (see clause 8.2.7). Using OAuth 2.0 access tokens described in this clause is deprecated and should only be used to interact with legacy.

NOTE 1: Typical choices for the implementation of the authorization server include the authorization server as a component of the API producer, or as an external component.

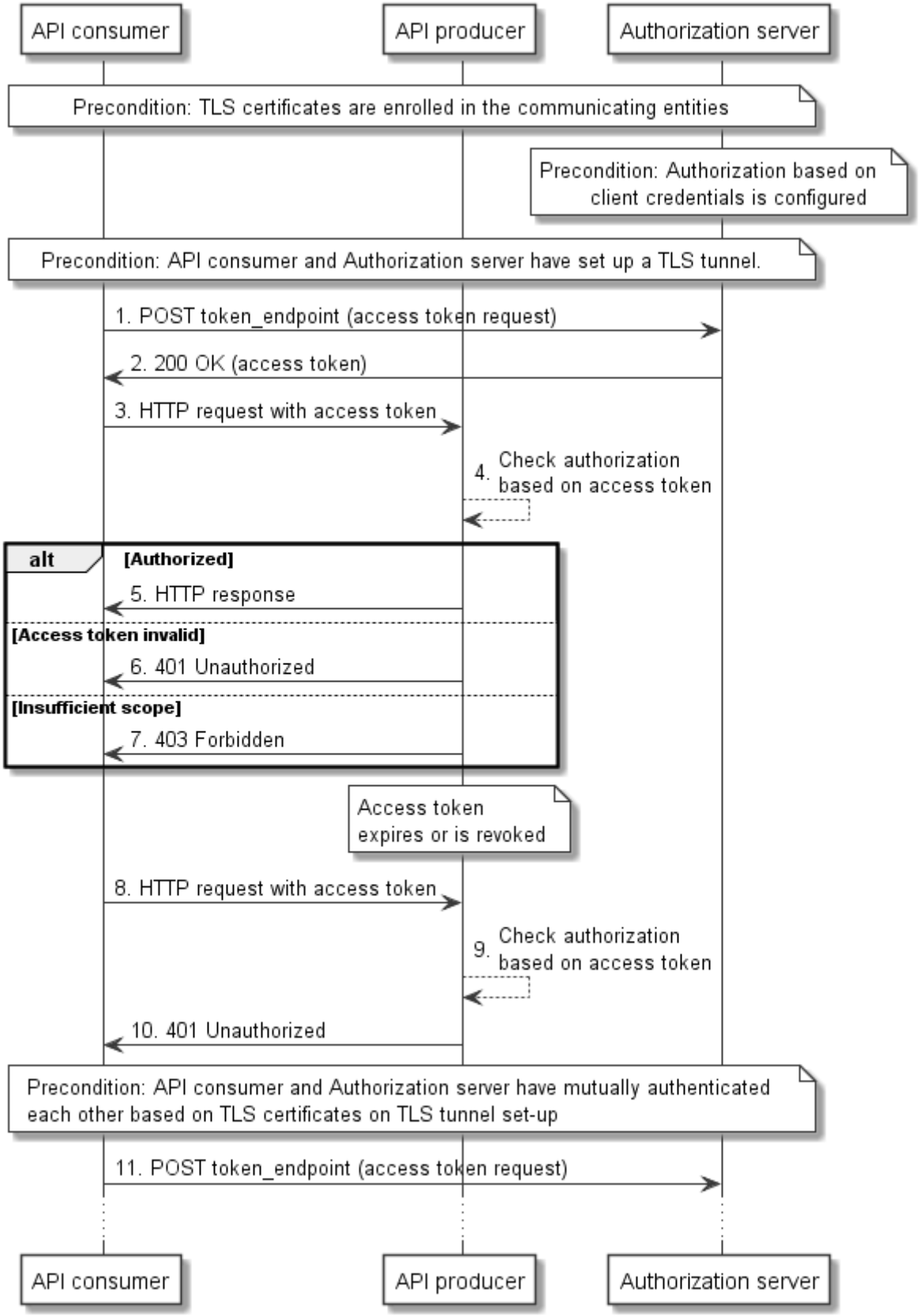

Figure 8.2.2-1: Authorization of API requests using OAuth 2.0 access tokens

Preconditions:

- Certificates are enrolled in the communicating entities as shown in figure 8.2.2-1.
- Authorization server is configured with the authorization policy and access rights against the client credentials.

- As a precondition for step 1 to succeed, a TLS channel has been set up between API consumer and authorization server. Unless the API consumer is allowed to use client password, the API consumer and the authorization server have mutually authenticated based on TLS certificates during TLS tunnel set-up.

NOTE 2: As an alternative to mutual authentication using TLS certificates, client password may be supported for legacy OAuth clients as defined in clause 8.1.

The flow consists of the following steps:

1) To obtain an access token, the API consumer sends via POST an access token request to the token endpoint of the authorization server and includes its client identifier.

2) The authorization server responds to the API consumer with an access token, and possibly additional information such as expiry time.

3) The API consumer sends an HTTP request to a resource provided by the API producer and includes the received access token.

4) The API producer checks the token for validity. This assumes that it has received information about the valid access tokens, and additional related information (e.g. time of validity, client identity, client access rights) from the authorization server. Such exchange is outside the scope of the present document, and assumed to be trivial if deployments choose to include the authorization server as a component into the API producer.

5) In case the token is valid and refers to access rights that allow accessing the actual resource with the actual request and its parameters, the API producer returns the HTTP response.

6) In case the token is invalid or expired, the API producer returns a "401 Unauthorized" response.

7) In case the access rights are insufficient to access the resource or to use the parameters, the API producer returns a "403 Forbidden" response.

8) The API consumer sends an HTTP request to the API producer and includes in the request the access token.

9) The API producer checks the token for validity, and establishes that it has expired, or has been revoked by the authorization server using means outside the scope of the present document.

10) The API producer responds with a "401 Unauthorized" response, indicating that the access token is invalid.

11) The API consumer attempts to obtain a new access token, as defined in step 1) (including the precondition). This may eventually succeed or fail, depending on whether access is allowed for that API consumer any longer.

NOTE 3: All the communication presented in this flow diagram is done over encrypted tunnel using TLS as described in clause 4.1.

8.2.3 Authorization of API requests using TLS certificates

The authorization of API requests binding authorization policy directly to TLS certificates that was allowed in previous versions of the present document up to version 3.4.1 shall neither be used nor supported.

8.2.4 Authorization of notifications using the HTTP Basic authentication scheme

The authorization of notifications based on the HTTP Basic authentication scheme (see IETF RFC 7617 [14]) that was allowed in previous versions of the present document up to version 3.4.1 shall neither be used nor supported.

8.2.5 Authorization of notifications using OAuth 2.0 access tokens

The flow below illustrates the authorization of notifications that the API producer sends to the API consumer using OAuth 2.0. In this flow, the authorization server can be a different entity than the authorization server in clause 8.2.2.

NOTE 1: Typical choices for the implementation of the authorization server include the authorization server as a component of the API consumer, or as an external component.

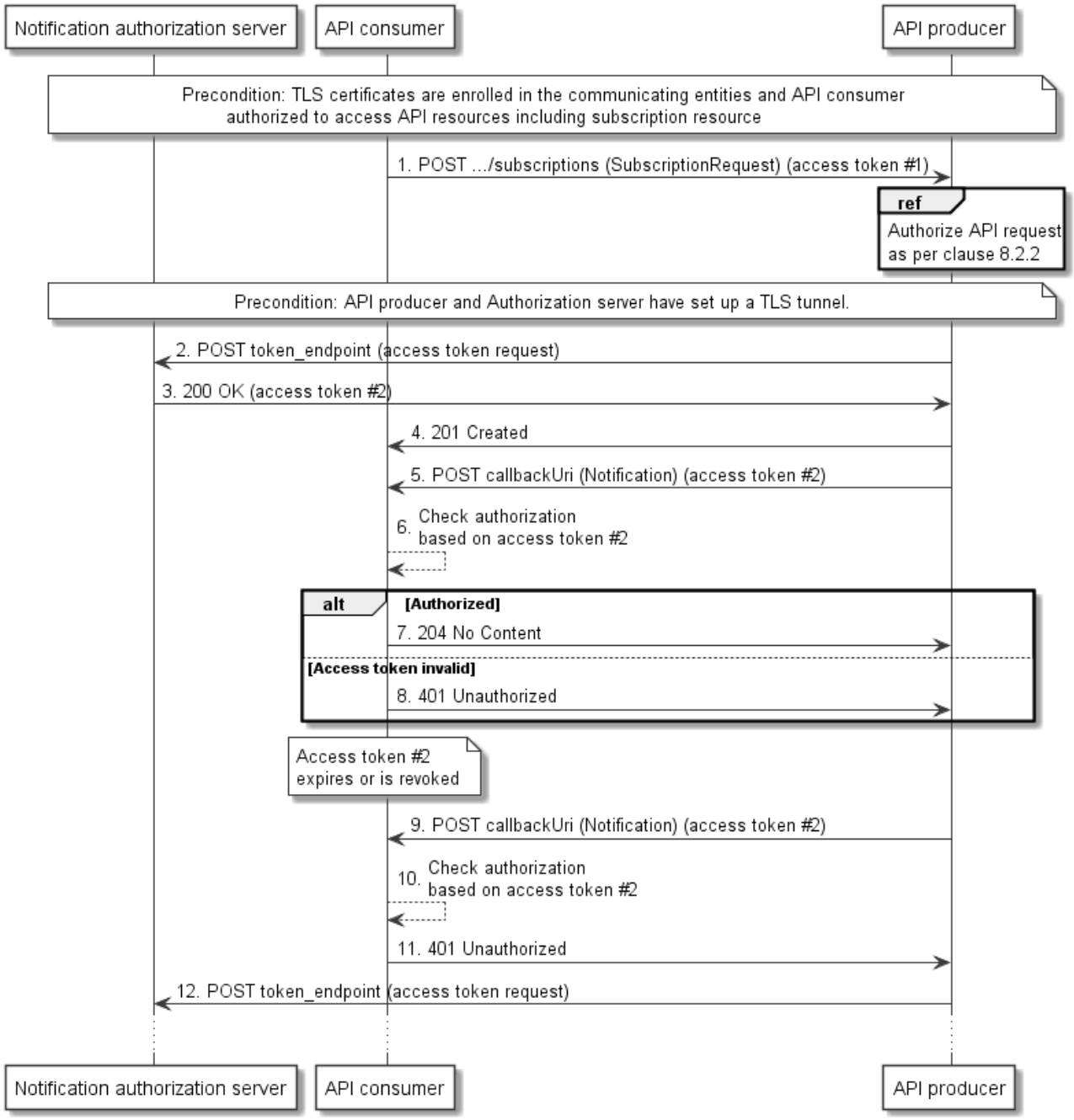

Figure 8.2.5-1: Authorization of notifications using OAuth 2.0

Preconditions:

- The API consumer is authorized to access the "subscriptions" resource provided by the API producer, using the procedure illustrated in clause 8.2.2.
- To ensure secure communication, it is a precondition that the TLS certificates are enrolled in the communicating entities.
- It is a precondition for any API consumer functional block that can subscribe for notifications from an API producer that, a client X.509 certificate of the API producer has been established in the API consumer (as the API producer acts as a client), unless the API producer is allowed to use the client password (see clause 8.1).

The flow consists of the following steps:

1) The API consumer sends a request to create a new subscription resource to the API producer and includes in the request a valid access token #1 to prove that it is authorized to access the API. Also, it includes in the subscription request parameters that the API producer can use to obtain authorization to send notifications to the API consumer, such as client credentials and a token endpoint. Note that these are typically different from the credentials and token endpoint used in the flow in clause 8.2.2. Subsequently, the API producer authorizes the API consumer (see clause 8.2.2).

2) Subsequently, and prior to sending any notification to the API consumer, the API producer obtains authorization for sending notifications by requesting an access token from the authorization server, using the end point and notification client credentials that were sent in the subscription request, or provisioned otherwise. As a precondition for this step to succeed, a TLS channel has been set up between API producer and notification authorization server. Unless the API consumer is allowed to use client password, the API producer and the notification authorization server have mutually authenticated based on TLS certificates during TLS tunnel set-up.

NOTE 2: As an alternative to mutual authentication using TLS certificates, client password may be supported for legacy OAuth clients as defined in clause 8.1.

3) The authorization server responds to the API producer with an access token, hereafter called access token #2, and possibly additional information such as expiry time.

4) The API producer creates the subscription resource and responds with "201 Created".

5) The API producer sends an HTTP POST request with a notification to the callback URI registered by the API consumer during subscription, and includes the received access token #2.

6) The API consumer checks the token for validity. This assumes that it has received information about the valid access tokens, and additional related information (e.g. time of validity, client identity, client access rights) from the authorization server. Such exchange is outside the scope of the present document, and assumed to be trivial if deployments choose to include the authorization server as a component into the API consumer.

7) In case the token #2 is valid, the API consumer returns a "204 No Content" HTTP response to indicate successful delivery of the notification.

8) In case the token #2 is invalid or expired, the API consumer returns a "401 Unauthorized" response.

9) The API producer sends another notification in an HTTP POST request to the API consumer and includes in the request the access token #2.

10) The API consumer checks the token #2 for validity, and establishes that it has expired, or has been revoked by the authorization server using means outside the scope of the present document.

11) The API consumer responds with a "401 Unauthorized" response, indicating that the access token #2 is invalid.

12) The API producer attempts to obtain a new access token as defined in step 2 (including the precondition). This may eventually succeed or fail, depending on whether access is allowed for that API producer any longer.

NOTE 3: All the communication presented in this flow diagram is done over encrypted tunnel using TLS as described in clause 4.1.

ERROR CONDITIONS:

- If the API producer cannot obtain an access token in step 3 it rejects the subscription, instead of executing step 4).

8.2.6 Authorization of notifications using TLS certificates

The authorization of notifications binding authorization policy directly to TLS certificates that was allowed in previous versions of the present document up to version 3.4.1 shall neither be used nor supported.

8.2.7 Authorization of API requests using OAuth 2.0 certificate-bound access tokens

The flow below illustrates the authorization of API requests that the API consumer sends to the API producer over a mutually authenticated TLS connection using OAuth 2.0 certificate-bound access tokens, as described in IETF RFC 8705 [25].

NOTE 1: Typical choices for the implementation of the authorization server include the authorization server as a component of the API producer, or as an external component.

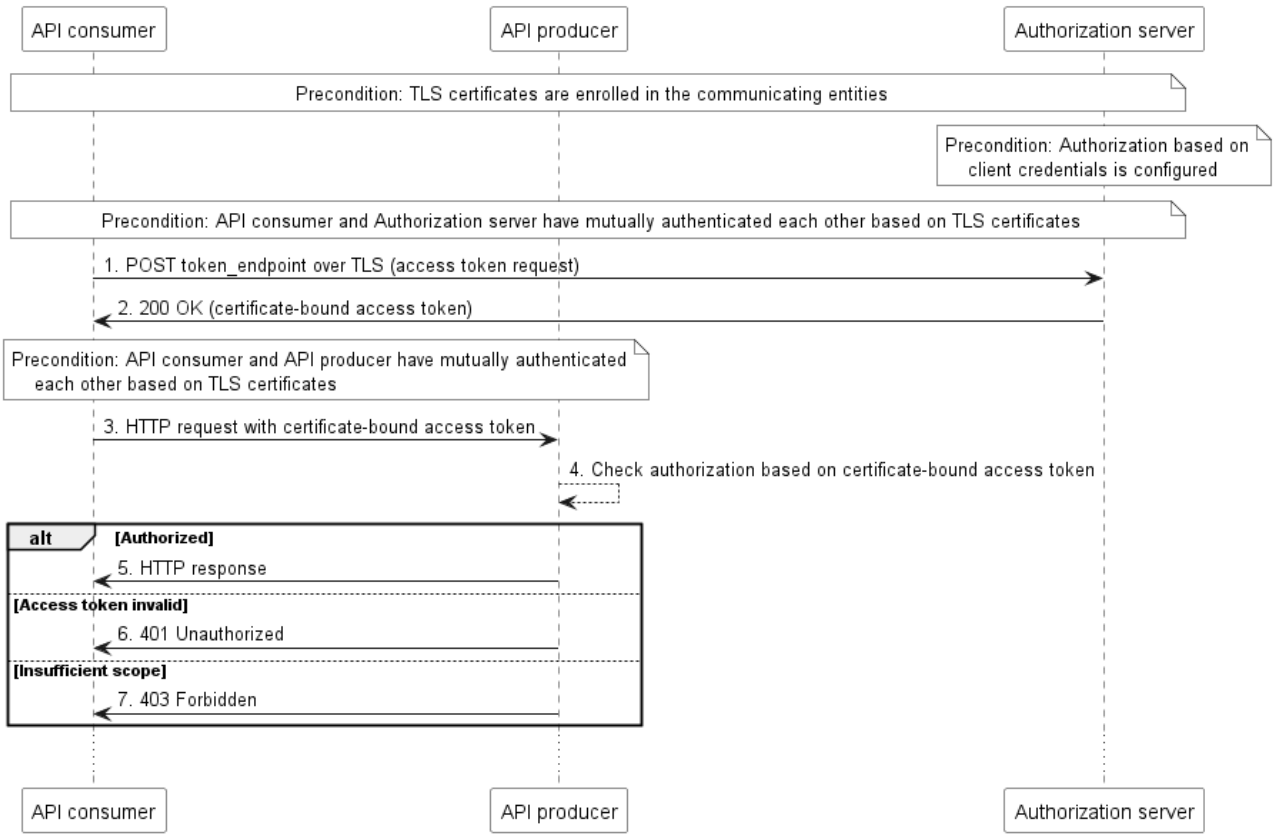

Figure 8.2.7-1: Authorization of API requests using OAuth 2.0 certificate-bound access tokens

Preconditions:

- Certificates are enrolled in the communicating entities as shown in figure 8.2.7-1.
- Authorization server is configured with the authorization policy and access rights against the client credentials.
- As a precondition for step 1 to succeed, API consumer and Authorization server have mutually authenticated each other based on TLS certificates.
- As a precondition for step 3 to succeed, API consumer and API producer have mutually authenticated each other based on TLS certificates.

The flow consists of the following steps:

1) To obtain an access token, the API consumer sends via POST an access token request to the token endpoint of the authorization server and includes its client identifier.

2) The authorization server responds to the API consumer with a certificate-bound access token, and possibly additional information such as expiry time.

3) The API consumer sends an HTTP request to a resource provided by the API producer and includes the received certificate-bound access token.

4) The API producer checks the token for validity by API client's certificate in mutual TLS. This assumes that it has received information about the valid access tokens, and additional related information (e.g. time of validity, client identity, client access rights) from the authorization server. Such exchange is outside the scope of the present document and assumed to be trivial if deployments choose to include the authorization server as a component into the API producer.

5) In case the token is valid and refers to access rights that allow accessing the actual resource with the actual request and its parameters, the API producer returns the HTTP response.

6) In case the token is invalid or expired, the API producer returns a "401 Unauthorized" response.

7) In case the access rights are insufficient to access the resource or to use the parameters, the API producer returns a "403 Forbidden" response.

NOTE 2: All the communication presented in this flow diagram is done over encrypted tunnel using TLS as described in clause 4.1.

8.3 Specification

8.3.1 Introduction

OAuth 2.0 provides a framework for authorization of web applications that has multiple modes and options. This clause profiles the framework for use in the context of the NFV-MANO reference points. Clause 8.3.2 specifies the general mechanism. Two different uses of the general mechanism, actually for API requests and for sending notifications, are defined in clauses 8.3.3 and 8.3.4.

8.3.2 General mechanism

For all requests to any RESTful NFV-MANO API, and for all notifications sent via such an API, authorization as defined below shall be used. Requests and notifications without authorization credentials shall be rejected.

To allow the *client* to obtain an access token, the *authorization server* shall expose a *token endpoint* that shall comply with the provisions defined by the OAuth 2.0 specification for the *client credentials* grant type (see IETF RFC 6749 [7]). A *client* shall use the access token request and response according to this grant type to obtain an *access token* for access to the REST resources defined by the RESTful NFV-MANO API specifications. Access token request and response shall comply with the provisions defined in IETF RFC 6749 [7]. As a precondition for the access token request to succeed, *client* and *authorization server* shall have mutually authenticated based on TLS certificates during TLS tunnel set-up, unless the use of client password is allowed for the *client*. In the case of client password, a TLS tunnel shall be used as well, but mutual authentication is not a requirement.

NOTE: As an alternative to mutual authentication using TLS certificates, client password may be supported for legacy OAuth clients as defined in clause 8.1.

The *access token* shall be a string with the set of allowed characters as defined in IETF RFC 6749 [7], and it shall not be possible for an attacker to easily guess it.

The NFV access token defined in ETSI GS NFV-SEC 022 [20] mitigates the risk of token stealing and also enables strong authentication of the API consumer.

Therefore, in addition to the provisions defined above:

1) The following requirements apply for APIs that are externally-facing from the trust domain that contains the ETSI NFV-MANO functional blocks:

   a) The token endpoint shall further comply with the provisions defined by clause 5.3 of ETSI GS NFV-SEC 022 [20].

   b) Access token request and response shall further comply with those defined by clause 5.3 of ETSI GS NFV-SEC 022 [20].

   c) The *access token* content and format shall comply with the provisions defined by clause 5.4 of ETSI GS NFV-SEC 022 [20] for the NFV access token.

2) The following strong recommendations apply for APIs that are non-externally-facing from the trust domain that contains the ETSI NFV-MANO functional blocks:

   a) The token endpoint should further comply with the provisions defined by clause 5.3 of ETSI GS NFV-SEC 022 [20].

   b) Access token request and response should further comply with those defined by clause 5.3 of ETSI GS NFV-SEC 022 [20].

   c) The *access token* content and format should comply with the provisions defined by clause 5.4 of ETSI GS NFV-SEC 022 [20] for the NFV access token.

A *client* that invokes HTTP requests towards a resource defined in one of the RESTful NFV-MANO API specifications shall include the *access token* in every HTTP request in the "Authorization" HTTP header, using the protocol defined for bearer tokens in IETF RFC 6750 [8]. A *resource server* that receives an HTTP request with an invalid *access token*, or without an *access token*, shall reject the request, and shall signal the error in the HTTP response according to the provisions for the error codes and the "WWW-Authenticate" response HTTP header as defined by IETF RFC 6750 [8].

A *client* that receives a rejection of an *access token* shall follow local policy to obtain a new *access token*. In line with local policy, it may retry the request with the new *access token*.

8.3.3 Authorizing API requests

A consumer of an API that wishes to issue HTTP requests towards resources provided by that API shall act as a *client* according to clause 8.3.2 to obtain an access token, and shall include this access token in every HTTP request, as defined in clause 8.3.2. The respective API producer shall act as a *resource server* as defined in clause 8.3.2.

The API consumer passes an access token when accessing a resource provided by API producer. The API producer checks authorization based on access token. The access token can be obtained from the authorization server based on API producer's credentials such as client ID and client certificate, or client ID and client password. See clause 8.1 for security caveats related to the use of client passwords.

8.3.4 Authorizing the sending of notifications

The procedure in clause 8.2 illustrates how an API consumer can obtain authorization to perform API requests towards the API producer, including subscription requests. Furthermore, the API consumer shall request the API producer (i.e. the subscriber) to authorize for sending the actual notifications matching a subscription, in which case the API producer shall obtain separate authorization to actually *send* the notification to the API consumer.

When an API consumer requests the API producer to authorize for sending notifications to that API consumer, it shall include in the subscription request a data structure that defines the authorization requirements, as defined in table 8.3.4-1.

NOTE: The attribute "paramsBasic" that was included up to version 3.4.1 of the present document has been removed from the "SubscriptionAuthentication" data type because this attribute provides insecure information and is therefore not supported.

In case of authType=OAUTH2_CLIENT_CREDENTIALS, a client certificate is established out of band, to be presented by the API producer when authenticating towards the API consumer's notification authorization server when obtaining the access token.

Table 8.3.4-1: Definition of the SubscriptionAuthentication data type

Attribute name	Data type	Cardinality	Description
authType	Enum (inlined)	1..N	Defines the types of Authentication/Authorization which the API consumer is willing to accept when receiving a notification. Permitted values (see note 3): • OAUTH2_CLIENT_CREDENTIALS: In every HTTP request to the notification endpoint, use an OAuth 2.0 token, obtained using the client credentials grant type after authenticating using client identifier and client password towards the token endpoint. • OAUTH2_CLIENT_CERT: In every HTTP request to the notification endpoint, use an OAuth 2.0 token, obtained using the client credentials grant type after mutually authenticating using client identifier and X.509 certificates towards the token endpoint.
paramsOauth2ClientCert	Structure (inlined)	0..1	Parameters for authentication/authorization using OAUTH2_CLIENT_CERT. Shall be present if authType is "OAUTH2_CLIENT_CERT" and the contained information has not been provisioned out of band. Shall be absent otherwise.
>clientId	String	1	Client identifier to be used in the access token request of the OAuth 2.0 client credentials grant type. The client identifier is unique in the scope of the tokenEndpoint.
>certificateRef	Structure (inlined)	1	Fingerprint of the client certificate. The hash function shall use SHA256 or higher. See note 4.
>>type	String	1	The type of the fingerprint. Permitted values: • x5t#S256: The SHA-256 thumbprint of the X.509 certificate as defined in section 4.1.8 of IETF RFC 7515 [23].
>>value	String	1	The fingerprint value as defined by the type.
>tokenEndpoint	Uri	1	The token endpoint from which the access token can be obtained.
paramsOauth2ClientCredentials	Structure (inlined)	0..1	Parameters for authentication/authorization using OAUTH2_CLIENT_CREDENTIALS. Shall be present if authType is "OAUTH2_CLIENT_CREDENTIALS" and the contained information has not been provisioned out of band. Shall be absent otherwise. See note 2.
>clientId	String	0..1	Client identifier to be used in the access token request of the OAuth 2.0 client credentials grant type. The client identifier is unique in the scope of the tokenEndpoint. Shall be present if it has not been provisioned out of band. See note 1.
>clientPassword	String	0..1	Client password to be used in the access token request of the OAuth 2.0 client credentials grant type. Shall be present if it has not been provisioned out of band. See note 1.
>tokenEndpoint	Uri	0..1	The *token endpoint* from which the access token can be obtained. Shall be present if it has not been provisioned out of band.

Attribute name	Data type	Cardinality	Description
NOTE 1: The clientId and clientPassword passed in a subscription shall not be the same as the clientId and clientPassword that are used to obtain authorization for API requests. Client credentials may differ between subscriptions. The value of clientPassword should be generated by a random process. NOTE 2: As a less secure alternative to OAUTH2_CLIENT_CERT which uses mutual authentication based on X.509 certificates, this mode which uses client password to authenticate may be used in the access token request toward the authorization server (as defined by IETF RFC 6749 [7]), only to support legacy implementations (version 3.4.1 or earlier version of the present document). See clause 8.1 for more details. NOTE 3: The following values that were included up to version 3.4.1 of the present document have been removed: "BASIC" (to signal the use of the basic HTTP authentication) has been removed because it is insecure. "TLS_CERT" to signal an alternative non-token based authorization method using TLS certificates has been removed because the method is no longer supported. NOTE 4: The client certificate is established by means outside the scope of the present document.			

The authType attribute is used to propose supported authorization methods of the API consumer for the authorization of notifications. The expected behaviour of the authorization methods that can be signalled in the "authType" attribute is defined as follows:

"OAUTH2_CLIENT_CREDENTIALS" or "OAUTH2_CLIENT_CERT":

- The API producer shall, prior to sending any notification, obtain an *access token* from the *token endpoint* using the OAuth 2.0 client credentials grant type as defined in IETF RFC 6749 [7]. Authentication shall follow the provisions in clause 8.3.2, using TLS certificates in case of "OAUTH2_CLIENT_CERT" and client ID + client password in case of "OAUTH2_CLIENT_CREDENTIALS". The API consumer should include expiry information with the token response.

- The API producer shall include that *access token* in every POST request that sends a notification (using the protocol defined for bearer tokens in IETF RFC 6750 [8]).

- If the *access token* is expired, the API consumer shall reject the notification. In that case, the API producer shall obtain a new *access token*, and repeat sending the notification.

- If the token expiry time is known to the API producer, it may obtain proactively a new access token.

8.3.5 Client roles

An *access token* allows the API producer to identify information about the *client* that has obtained the access token, such as client identity, client role or client access rights. By having this property, *access tokens* can be used as a means to distinguish between different roles (and consequently different access rights) to the same set of resources.

The mechanism for this works as follows: By means out of scope of the present document, the role of the client identified by a particular client identifier is provisioned to the authorization server. When that client obtains an access token, it first authenticates towards the token endpoint exposed by the authorization server and provides its client identifier as part of the authentication process. The authorization server can obtain the role of the client by evaluating the data that were provisioned for the client identifier, and associate that information to the access token. By means out of scope of the present document, that association is shared with the API producer. This enables the API producer to detect the role based on the access token.

In ETSI NFV, certain interfaces are exposed on multiple different reference points, i.e. the same interface is exposed to different consumer functional blocks. Depending on the consumer block that originates an HTTP request, not all resources/HTTP methods/request and parameters might be available. From the point of view of the producer functional block, this can be seen as consumers acting in different roles when accessing a particular interface.

Implementations may use the OAuth *access token* to differentiate between these cases, assuming that an *access token* can determine in which role (e.g. VNFM, NFVO, EM, VNF) a consumer functional block acts when accessing a particular interface. This assumes that the role of the consumer functional block is bound to its client credentials. The means of creating this binding is out of scope of the present document (e.g. a configuration step or policy).

8.3.6 Support of authorization

8.3.6.1 Authorization of API requests

The API producer shall support checking the authorization of API requests it receives based on an OAuth 2.0 access token as defined in clause 8.3.3.

The API consumer shall support the authorization of API requests it sends by including an OAuth 2.0 access token in the request as defined in clause 8.3.3.

8.3.6.2 Authorization of notification requests

The API consumer shall support checking the authorization of notification requests it receives based on an OAuth 2.0 access token as defined in clause 8.3.4.

The API producer shall support the authorization of notification requests it sends by including an OAuth 2.0 access token in the request as defined in clause 8.3.4.

8.3.7 Authorization scope values

According to section 3.3 of IETF RFC 6749 [7], in OAuth 2.0 the token endpoint allows the client to specify the scope of an *access token* to indicate the permissions associated with the token.

Each RESTful NFV-MANO API can define a set of scope values and the associated permissions. Such scope values indicate the API for which they are valid, its major version and the actual permission within the API in terms of a represented set of resources and associated methods. Qualifiers can be used to indicate specific facets of the accessed resources (for instance, whether the VNF instance information includes information of VNFC instance or not). Such qualifiers can be used to cater for differences of the same API used on different reference points such as Or-Vnfm vs. Ve-Vnfm. Access permissions can be limited to read-only.

If a set of resources that represent individual subscriptions to notifications and their parent container resource are included in the set of resources of a scope value *besides other resources* and the scope value is tagged as read-only, this allows create and delete operations on the subscription-related resources, but not on the other resources that are part of the scope. However, if a scope value is related to subscription-related resources *only*, tagging it as read-only implies that only read operations are allowed on the subscription-related resources.

A scope value is a string that can be defined for a RESTful NFV-MANO API following the structure specified below.

```
scopeValue        := {apiName}":"{apiMajorVersion}":"<permissionName>(":"<qualifier>)*[":"<access>]
apiName           := string not containing ":"
apiMajorVersion   := "v"(<digit>)+
permissionName    := string not containing ":"
qualifier         := string not containing ":"
access            := "readonly" | "readwrite"
digit             := "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

where:

```
{apiName}         string that indicates the API name as defined in the API's URI structure
{apiMajorVersion} string that indicates the major API version as defined in the API's URI structure
<permissionName>  logical name, defined per API, reflecting the object of the permission (resources
                  with methods or other scopes)
<qualifier>       string, defined per API, that qualifies the permission further, e.g. to express
                  differences between an API's usage on different reference points or by consumers
                  with different privileges
<access>          string, defined per API, that indicates whether the scope value is only for
                  reading, or also permits writing. If two scope values only differ in the presence
                  of the "readonly" suffix, the scope value without suffix implies "readwrite"
                  access. See examples 2 and 3.
```

and:

```
*  zero or more occurrences
+  one or more occurrences
() grouping of expressions to be used with * or +
[] optionality (zero or one occurrence)
"" quotation marks for marking string constants
```

```
<> name separator
{} name separator to reference URI variables
|  separator of alternatives
```

The following examples of scope values are provided to illustrate the specification given above.

EXAMPLE 1: Scope value that allows instantiating a VNF:

```
vnflcm:v2:instantiate
```

EXAMPLE 2: Scope values that allow reading and updating VNF instance information(synonyms):

```
vnflcm:v2:vnf_instance_info
```

```
vnflcm:v2:vnf_instance_info:readwrite
```

EXAMPLE 3: Scope value that only allows reading VNF instance information:

```
vnflcm:v2:vnf_instance_info:readonly
```

EXAMPLE 4: Scope value that allows reading and updating VNF instance information including VNFC information:

```
vnflcm:v2:vnf_instance_info:with_vnfc
```

9 Version management

9.1 Version identifiers and parameters

9.1.1 Version identifiers

API version identifiers shall consist of 3 numerical fields, following a MAJOR.MINOR.PATCH pattern and the rules for Semantic Versioning [18] with the additional clarifications defined in clause 9.2. The fields in an API version identifier shall be separated by dots ".". The last field may be followed by one or more version parameters.

The MAJOR, MINOR and PATCH fields are defined in [18] for Semantic Versioning.

The {apiMajorVersion} segment of the URIs used by an API shall be set to the character "v" followed by value of the MAJOR field of the API version identifier.

EXAMPLE: ".../vnflcm/v1/

The full version identifier (including parameters) is used in ApiVersionInformation (see clause 7.1.6) and in version signalling (see clause 9.4). Furthermore, it also appears in the corresponding OpenAPI file that ETSI publishes as collateral material for each RESTful NFV-MANO API specification [i.4].

9.1.2 Version parameters

Version parameters are separated from the version identifier by a dash "-". Version parameters are separated by semicolons ";".

The present document defines the following version parameters:

- impl

The optional "impl" parameter identifies an implementation and a version of this implementation (e.g. implementation delivered by an open source community or a vendor). The OpenAPI specification that ETSI publishes as collateral material for each RESTful NFV-MANO API specification [i.4] is also considered an implementation under this scheme. The "impl" parameter shall have the following structure: "impl:" <vendor>":"<product>":"<impl_version>, where:

- the <vendor> field shall be a string that contains either an IANA Enterprise Number assigned to that vendor, or an Internet domain name owned by that vendor;

- the <product> field shall contain a string identifying the product, chosen by the vendor;
- the <impl_version> field shall contain a number that defines the version of the implementation. Version numbers of subsequent implementations shall be monotonically increasing.

9.2 Rules for incrementing version identifier fields

9.2.1 General

In the RESTful NFV-MANO APIs, versioning applies to the resources structure (URI structure, URI query parameters, and supported HTTP methods) and the message content. Different criteria are applied to increment MAJOR, MINOR, and PATCH version fields for changes that affect the URI compared to changes that affect the message content.

The fields of an API version identifier are incremented from a previous version to the current version according to the following rules:

- 1st field (MAJOR): This field is always incremented when one or more changes were made to the resources structure of the API that break backward compatibility. This field is also incremented if one or more changes were made to at least one message content of the API that break backward compatibility, unless that change is correcting an error.

NOTE 1: A change that corrects an error that would lead the API producer to always send an error response if a certain valid condition is met is not considered a non-backward compatible change, irrespective of the type of change. Indeed, compatibility between a new version and a previous version can only be assessed for a feature that is properly supported in the previous version.

- 2nd field (MINOR): This field is incremented if one or more technical changes (at least one of which is not an error correction) were made to the API specification but none of them (apart from error corrections to the message content) breaks backward compatibility. It is reset to zero if the MAJOR version identifier is changed.
- 3rd field (PATCH): This field is incremented if one or more error corrections that are visible in communication between API producer and API consumer were made to the API specification but none of them (apart from error corrections to the message content) breaks backward compatibility. It is reset to zero if the MINOR version identifier is changed.

NOTE 2: All the aforementioned types of changes affect the corresponding OpenAPI specification that ETSI publishes as collateral material for each RESTful NFV-MANO API specification [i.4].

9.2.2 Examples of backward and non-backward compatible changes

Examples of backward compatible changes include:

- Adding a new resource.
- Adding a new URI.
- Supporting a new HTTP method for an existing resource.
- Adding new optional URI query parameters.
- Adding new optional attributes to a resource representation in a request.
- Adding new attributes to a resource representation in a response or to a notification message.
- Responding with a new status code of an error class.
- Certain cardinality changes (see note 2).

NOTE 1: Whether attribute cardinality changes are backward compatible depends on the type of change. An example of a backward-compatible cardinality change includes making an attribute in a response required (e.g. changing cardinality from 0..1 to 1).

Examples of non-backward compatible changes to the resources structure include:

- Removing a resource/URI.
- Removing support for an HTTP method.
- Changing a resource URI.
- Adding new mandatory URI query parameters.

Examples of non-backward compatible changes to the message content include:

- Renaming an attribute in a resource representation.
- Adding new mandatory attributes to a resource representation in a request.
- Changing the data type of an attribute.
- Certain cardinality changes (see note 2).

NOTE 2: Whether attribute cardinality changes are backward compatible depends on the type of change. Examples of non-backward compatible cardinality changes include decreasing the upper bound of a cardinality range for attributes sent by the client, changing the meaning of the default behaviour associated to the absence of an attribute of cardinality 0..N, etc.

9.3 Version information retrieval

9.3.1 General

The API producer shall support the following dedicated URIs to enable API consumers to retrieve information about API versions supported by an API producer:

1. {apiRoot}/{apiName}/api_versions
2. {apiRoot}/{apiName}/{apiMajorVersion}/api_versions

To obtain information about the supported API versions, the API consumer shall send a GET request to a URI of one of above forms. The information contained in the GET response depends on the form of URI used in the GET request, as follows:

- If the first form is used, the GET response shall provide the list of supported versions for the API corresponding to the {apiName} indicated in the GET Request URI.
- If the second form is used, the GET response shall provide the list of supported versions for the API corresponding to the {apiName} and the {apiMajorVersion} indicated in the GET Request URI.

If the API producer receives a GET request:

- In case of success, the API producer shall return in the body of a 200 OK response a value of the ApiVersionInformation data type specified in clause 7.1.6.
- In case URI query parameters are provided, the API producer shall return a "400 Bad request" response as defined in clause 6.4.
- In other cases of failure, the API producer shall return appropriate error information as defined in clause 6.4.

9.3.2 Resource structure and methods

Table 9.3.2-1 lists the individual resources defined for supporting API version information retrieval, and the applicable HTTP method. The API producer shall support responding to GET requests on the resources in table 9.3.2-1.

Table 9.3.2-1: Resources and methods overview for API version information retrieval

Resource name	Resource URI	HTTP Method	Meaning
API versions	/{apiName}/api_versions	GET	Version information associated to an API.
API versions	/{apiName}/{apiMajorVersion}/api_versions	GET	Version information associated to a major version of an API.

Figure 9.3.2-1 shows the "API versions" resources in the overall resource URI structure defined for all APIs.

The "API versions" resources, as defined in the present clause, shall be part of the overall resource URI structure of each RESTful NFV-MANO API.

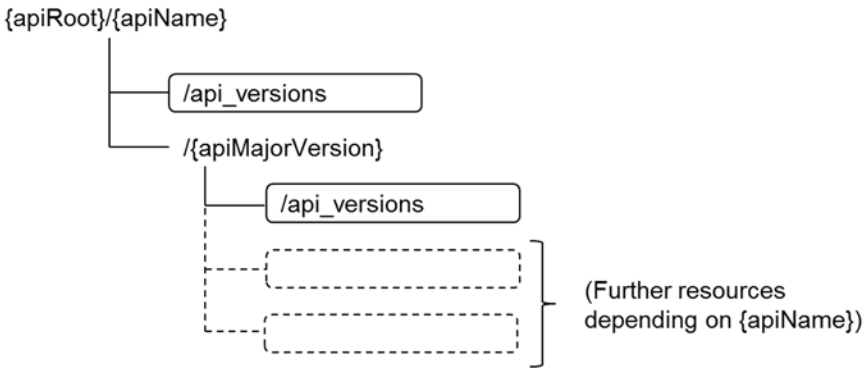

Figure 9.3.2-1: "API versions" resources

9.3.3 Resource: API versions

9.3.3.1 Description

There are two "API versions" resources defined for each API. The client can use these resources to obtain API version information.

9.3.3.2 Resource definition

The resource URI of each of the two "API versions" resources shall be of one of the following forms:

```
1. {apiRoot}/{apiName}/api_versions
2. {apiRoot}/{apiName}/{apiMajorVersion}/api_versions
```

These resources shall support the resource URI variables defined in table 9.3.3.2-1.

Table 9.3.3.2-1: Resource URI variables for these resources

Name	Definition
apiRoot	See clause 4.1.
apiName	See clause 4.1.
apiMajorVersion	See clause 4.1.

9.3.3.3 Resource methods

9.3.3.3.1 POST

This method is not supported. When this method is requested on this resource, the API producer shall return a "405 Method Not Allowed" response as defined in clause 6.4.

9.3.3.3.2 GET

The GET method reads API version information. This method shall follow the provisions specified in table 9.3.3.3.2-1 for request and response data structures, and response codes. URI query parameters are not supported.

Table 9.3.3.3.2-1: Details of the GET request/response on this resource

Request body	Data type	Cardinality	Description	
	n/a			
Response body	**Data type**	**Cardinality**	**Response Codes**	**Description**
	ApiVersionInformation	1	200 OK	API version information was read successfully. The response body shall contain API version information, as defined in clause 7.1.6.
	ProblemDetails	See clause 6.4	4xx/5xx	In addition to the response codes defined above, any common error response code as defined in clause 6.4 may be returned.

9.3.3.3.3 PUT

This method is not supported. When this method is requested on this resource, the API producer shall return a "405 Method Not Allowed" response as defined in clause 6.4.

9.3.3.3.4 PATCH

This method is not supported. When this method is requested on this resource, the API producer shall return a "405 Method Not Allowed" response as defined in clause 6.4.

9.3.3.3.5 DELETE

This method is not supported. When this method is requested on this resource, the API producer shall return a "405 Method Not Allowed" response as defined in clause 6.4.

9.4 Version signalling

The API consumer shall include the "Version" HTTP header (see IETF RFC 4229 [19]) in each HTTP request, excluding the version information retrieval requests specified in clause 9.3 where the "Version" header may be omitted. The "Version" header shall contain the three version identifier fields (MAJOR.MINOR.PATCH) indicating the API version the API consumer intends to use. The "impl" version parameter may be provided, indicating the version of the API producer implementation that the API consumer intends to use.

The API producer shall support receiving and interpreting the "Version" HTTP header. The API producer shall include in the response the "Version" HTTP header signalling the used API version, including the "impl" version parameter if available. If the "impl" version parameter has been omitted in the request, the API producer shall use the combination of MAJOR, MINOR and PATCH as requested and the highest supported value for the "impl_version" field of the "impl" version parameter for that combination, if available.

NOTE: In case multiple versions and/or implementation versions are supported by an API producer, this allows the API consumer to request a particular version.

API consumers conforming to versions of the RESTful NFV-MANO API specifications previous to version 2.5.1 omit this header. If the API producer receives a request without this header:

- If it supports the provisions defined in version 2.4.1 of the applicable RESTful NFV-MANO API specification document, it shall behave as defined in that document, and should indicate this by using MAJOR=1 and MINOR=1 and PATCH=0 in the "Version" HTTP header in the response.
- Otherwise, it shall respond with a 400 Bad Request response and shall include in the response message content a ProblemDetails structure providing more information on the cause of the error in the "detail" attribute.

If the API version signalled in the "Version" request header is not supported by the API producer, the API producer shall respond with a "406 Not Acceptable" error and shall include in the response message content a ProblemDetails structure providing more information on the cause of the error in the "detail" attribute.

Annex A (informative): Change history

Date	Version	Information about changes
November 2018	2.5.2	Initial version based on: - NFVSOL(18)000583r1 Contributions incorporated: - NFVSOL(18)000582r3_SOL013_Content_moved_from_SOL003
February 2019	2.5.3	Contributions incorporated - NFVSOL(18)000677r1_SOL013ed261_Addressing_EN_on_error_codes - NFVSOL(19)000003r1_SOL013_modifications_to_authorization_description - NFVSOL(19)000104r1_SOL013ed261_Version_related_update_for_publication Editorials: - Changed year - Fixed bullet numbering in clause 8 - Fixed mis-formatting and missing table reference in clause 7.2.2
March 2019	2.6.1	Publication by ETSI
August 2019	2.6.2	Contributions incorporated: - NFVSOL(19)000127r3_SOL013ed271_Reference_to_SEC022 - NFVSOL(19)000413_SOL013_moving_OpenAPI_version_conventions_to_SOL015 - NFVSOL(19)000465r1_SOL013_fixing_occurrences_of_the_present_document (with some editorials applied) Editorials: - Generally use "NFV-MANO" - replace occurrences of "ETSI NFV-SOL APIs" in clause 9.4 with "RESTful NFV-MANO APIs"
October 2019	2.6.3	Contributions incorporated: - NFVSOL(19)000501_SOL013ed271_numeric_data_types
December 2019	2.7.1	Publication by ETSI
December 2019	3.0.1	First Release 3 draft created based on V 2.7.1
May 2020	3.0.2	Contributions incorporated: - NFVSOL(20)000237r1_SOL013ed331_Moving_checksum_from_SOL_API_specs
May 2020	3.0.3	Contributions incorporated: - NFVSOL(20)000446_SOL013ed331_forward_mirror_of_445_resolve_bug_report_0007836 - NFVSOL(20)000453_SOL013ed331_Fixing_misalignment_of_normative_statements_for_
June 2020	3.0.4	Contributions incorporated: - NFVSOL(20)000481_SOL013ed331_Nokia_review_comments
September 2020	3.3.1	Version update for publication
November 2020	3.3.2	Contributions incorporated: - BWC: NFVSOL(20)000759r3_SOL013ed341_TLS_version_security_fix
January 2021	3.4.1	Version update for publication
January 2021	3.4.2	Contributions incorporated: - BWC: NFVSOL(20)000736r9_SOL013ed351_changes_from_SEC Editorials: - Add the reference of IETF RFC 7515 [23] in clause 2.1, since it is referred to by NFVSOL(20)000736r9. - Correct typo "sever" as "server" in clause 8.3.5.

Date	Version	Information about changes
February 2021	3.4.3	Contributions incorporated: - BWC: NFVSOL(21)000010_SOL013_CR_to_remove_TLS-based_authorization_alternative - BWC: NFVSOL(21)000014r1_SOL013ed351_version_resource_bugfix - BWC: NFVSOL(21)000043r1_SOL013ed351_Update_the_reference_of_JSON_Schema_to_the_lates Editorials: - For NFVSOL(21)000010 typo, modify "is" to "are" in clause 8.1. - For NFVSOL(21)000014r1 typo, add "on" after "depending" in the description of attribute "uriPrefix" in table 7.1.6-1.
March 2021	3.4.4	Contributions incorporated: - BWC: NFVSOL(21)000026r2_SOL013Ed351_-_Clause_8_4_3_-_Clarification_on_authorizing_th
May 2021	3.4.5	Contributions incorporated: - BWC: NFVSOL(21)000237_SOL013ed351_spelling_of_URI_data_type_name - BWC: NFVSOL(21)000271r1_SOL013ed351_address_editor_s_note
July 2021	3.5.1	Version update for publication
February 2022	4.0.1	Contributions incorporated: - NBWC: NFVSOL(22)000021_SOL013ed431_Make_authorization_mandatory_on_sending_of_notif
April 2022	4.0.2	Contributions incorporated: - BWC: NFVSOL(22)000067r2_SOL013ed431_Addition_of_new_HTTP_header_fields
May 2022	4.0.3	Contributions incorporated: - BWC: NFVSOL(22)000256r2_SOL013ed431_Bugfix_of_chicken-and-egg_problem_with_versionin
July 2022	4.3.1	Version update for publication
October 2022	4.3.2	Contributions incorporated: - BWC: NFVSOL(22)000431r1_SOL013ed441_Mirror_of_424_IETF_RFC_references_update_due_to_
December 2022	4.3.3	Contributions incorporated: - BWC: NFVSOL(22)000507r3_SOL013ed441_Add_OAuth2_0_scope - BWC: NFVSOL(22)000512r2_SOL013ed441_add_IETF_RFC_8705_based_flow Editorials: - Updated the reference links of IETF RFC documents in clause 2.
January 2023	4.3.4	Contributions incorporated: - BWC: NFVSOL(23)000002_SOL013ed441_define__access__in_authorization_scope
January 2023	4.3.5	Contributions incorporated: - BWC:NFVSOL(23)000025_SOL013ed441_clarify_relationship_between_subscription_and
March 2023	4.4.1	Version update for publication
April 2024	5.0.2	Contributions incorporated: - BWC: NFVSOL(24)000116_SOL013ed511_Updating_SEC022_reference

History

Document history		
V5.1.1	July 2024	Publication